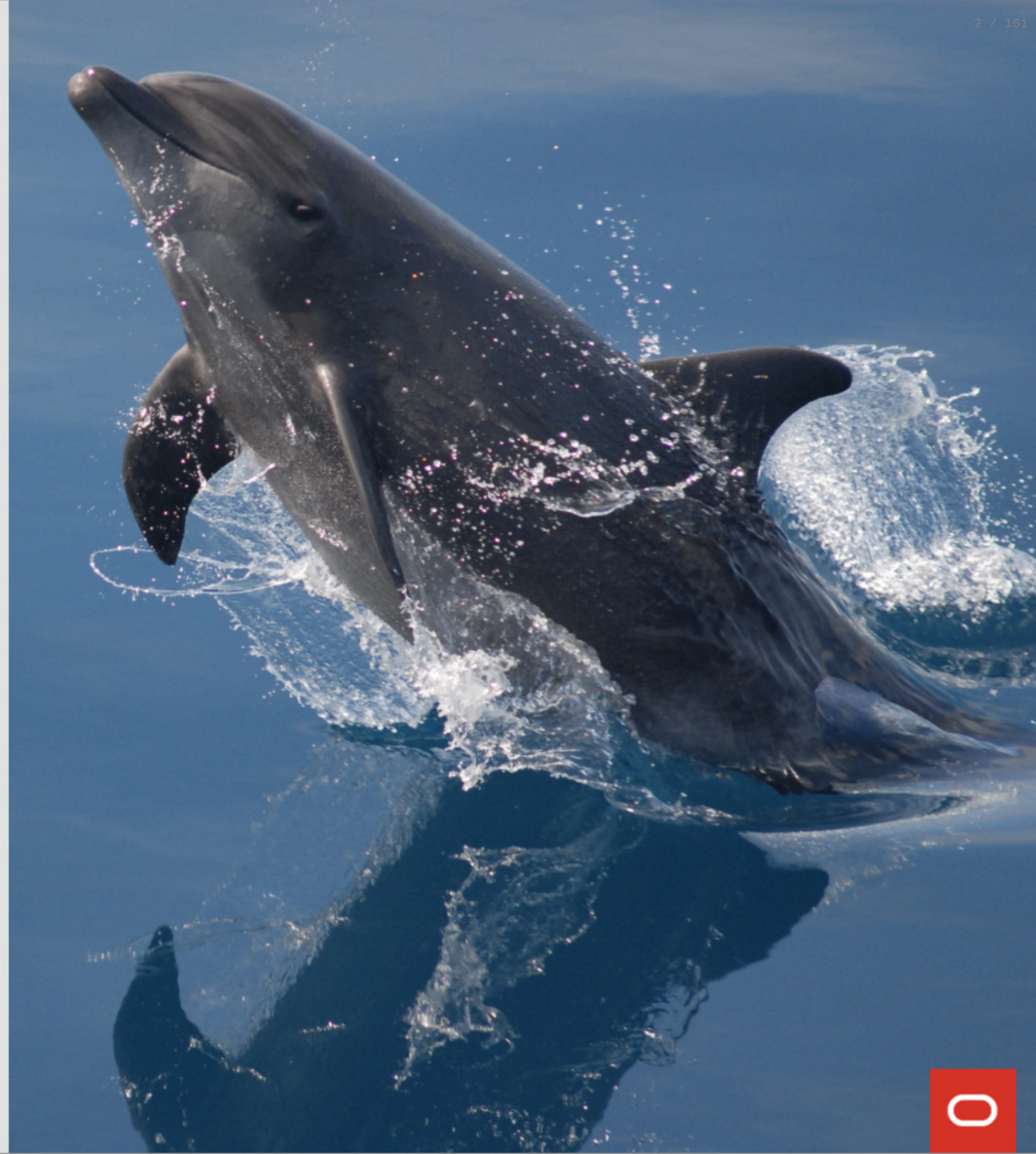
ORACLE

MySQL InnoDB Cluster

Advanced Configuration & Operations

Pedro Gomes
MySQL Senior Software Developer

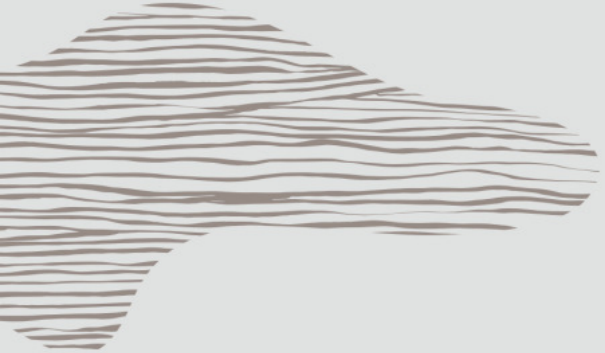
Frédéric Descamps
MySQL Community Manager
EMEA & APAC



Safe Harbor

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, timing and pricing of any features or functionality described for Oracle´s product may change and remains at the sole discretion of Oracle Corporation.

Statement in this presentation relating to Oracle´s future plans, expectations, beliefs, intentions and ptospects are "forward-looking statements" and are subject to material risks and uncertainties. A detailed discussion of these factors and other risks that affect our business is contained in Oracle´s Securities and Exchange Commission (SEC) filings, including our most recent reports on Form 10-K and Form 10-Q under the heading "Risk Factors". These filings are available on the SEC´s website or on Oracle´s website at http://www.oracle.com/investor. All information in this presentation is current as of September 2019 and Oracle undertakes no duty to update any statement in light of new information or future events.

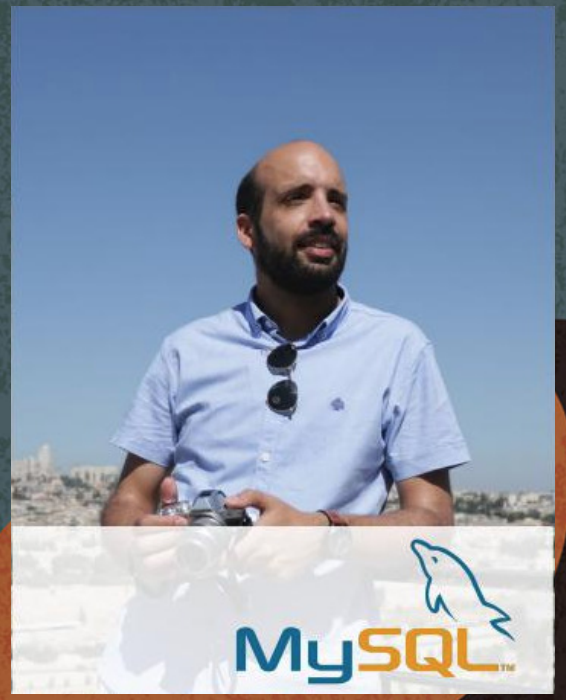


about us

Who are we ?

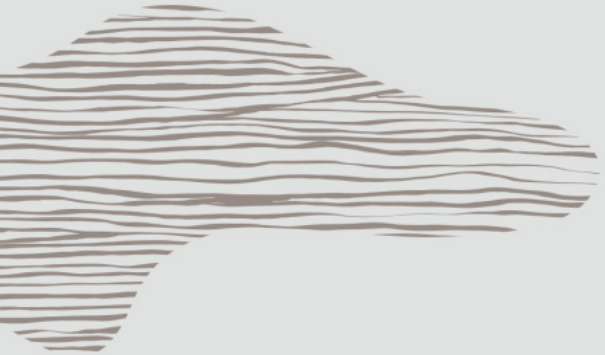
Pedro Gomes

- @distributedpete
- MySQL Replication Developer
- MySQL Developer since 2012
- Portuguese born and raised 🇵🇹
- https://mysqlhighavailability.com



Frédéric Descamps

- @lefred
- MySQL Evangelist
- Managing MySQL since 3.23
- devops believer
- living in Belgium 🇧🇪
- https://lefred.be



MySQL InnoDB Cluster

New Member Provisioning & Recovery

Provisioning VS Recovery

- Provisioning a server is the process of configuring and adding data to a server, which can then potentially be used in a replication topology. Typically, this involves restoring a backup or snapshot into a given instance.
- The distributed recovery is the protocol that runs when a server is added to a cluster, so that the new server finds a state donor, fetches missing state and synchronizes with the rest of the cluster automatically.
- As of MySQL 8.0.17, the distributed recovery protocol for Group Replication is able to automatically orchestrate provisioning, in addition to using binary logs, in the process of synchronizing the new instance with the rest of the group

How to Provision a New Member

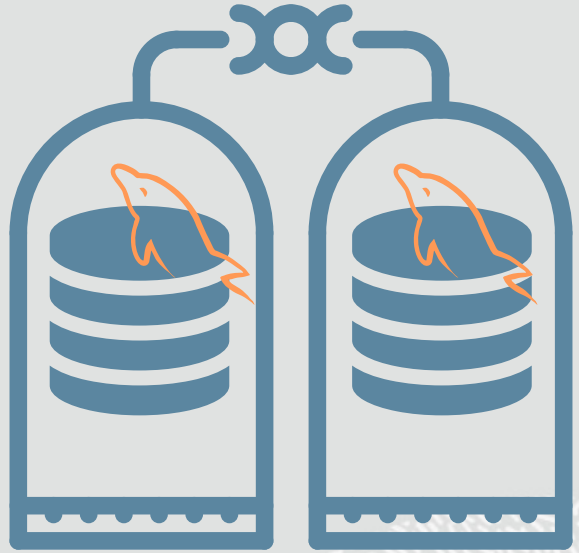
- Incremental Recovery
  - New node for a recent Cluster
  - From a Backup
- Clone (NEW 8.0.17)

Incremental Recovery

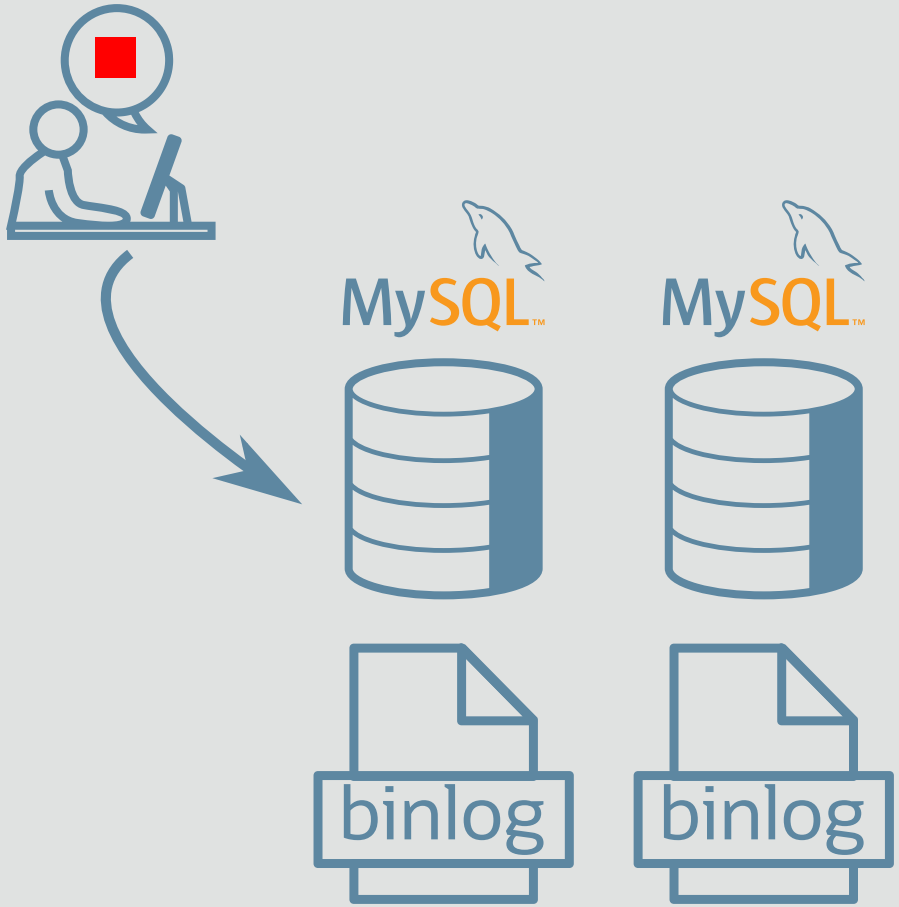
- Based on the good old Asynchronous replication
- Fetches only the missing data based on the member GTIDs info
- Also provides the member with group transactional information

Clone

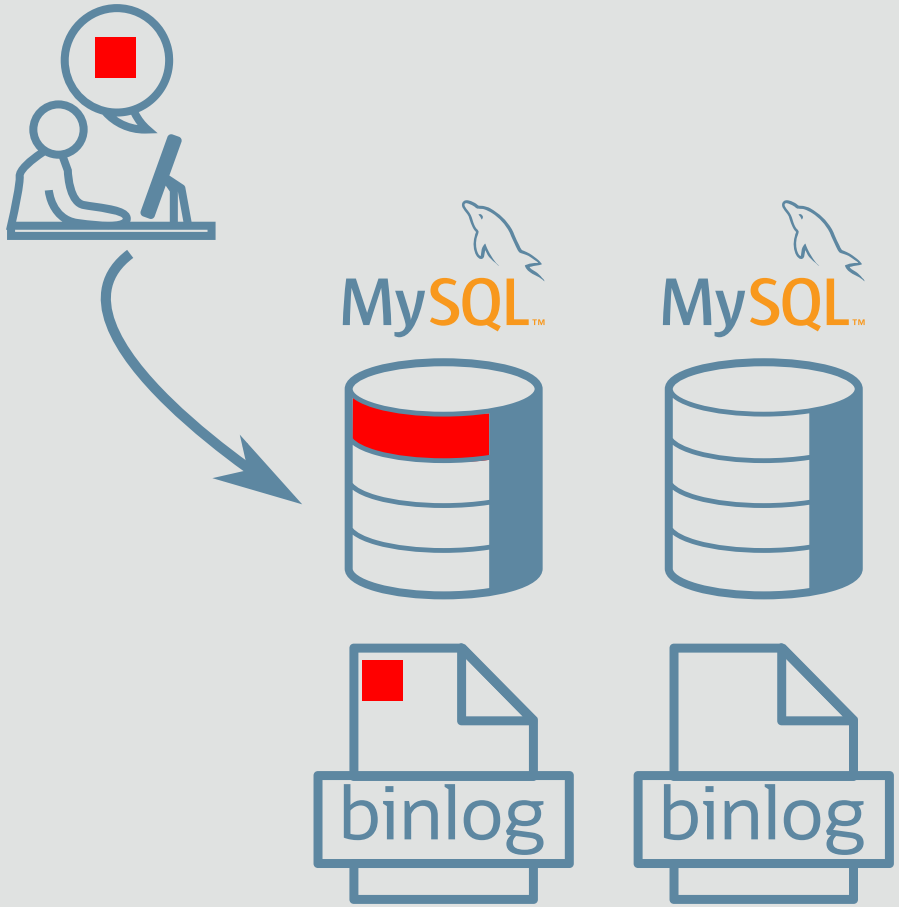
- Based on the brand new Clone plugin
- Fetches a full snapshot of the donor data
- Replaces the server data and requires its restart
- Still requires incremental recovery for group transactional information



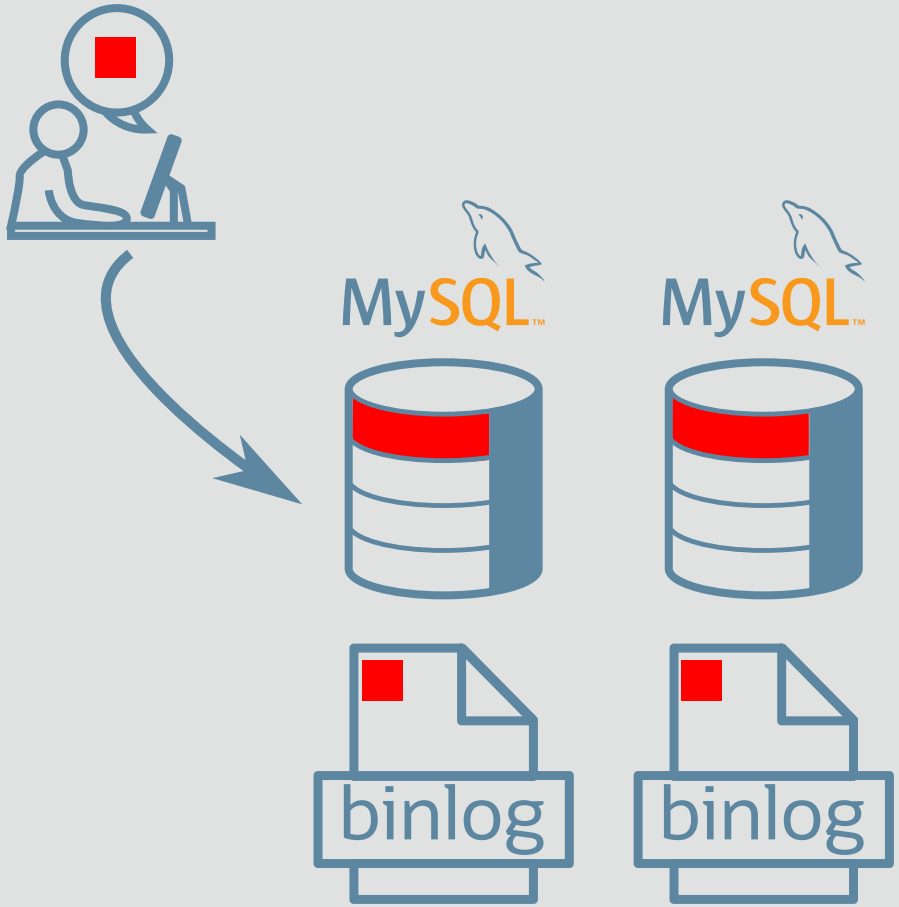
MySQL InnoDB Cluster Provisioning



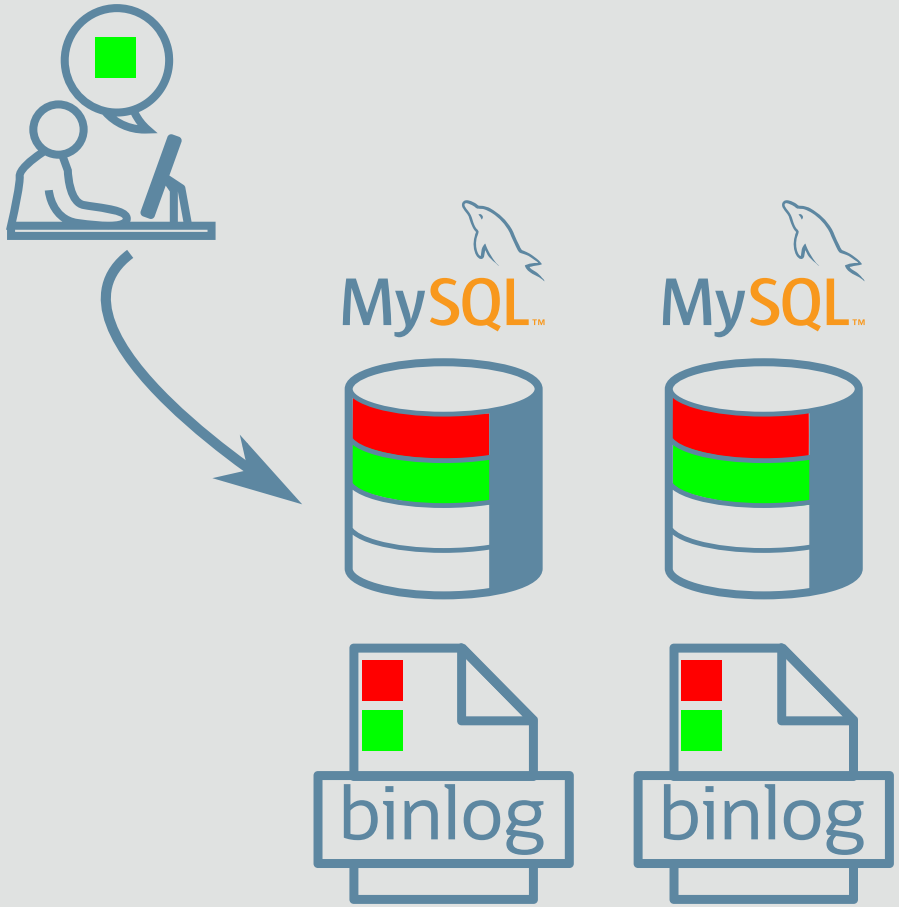
MySQL InnoDB Cluster Provisioning



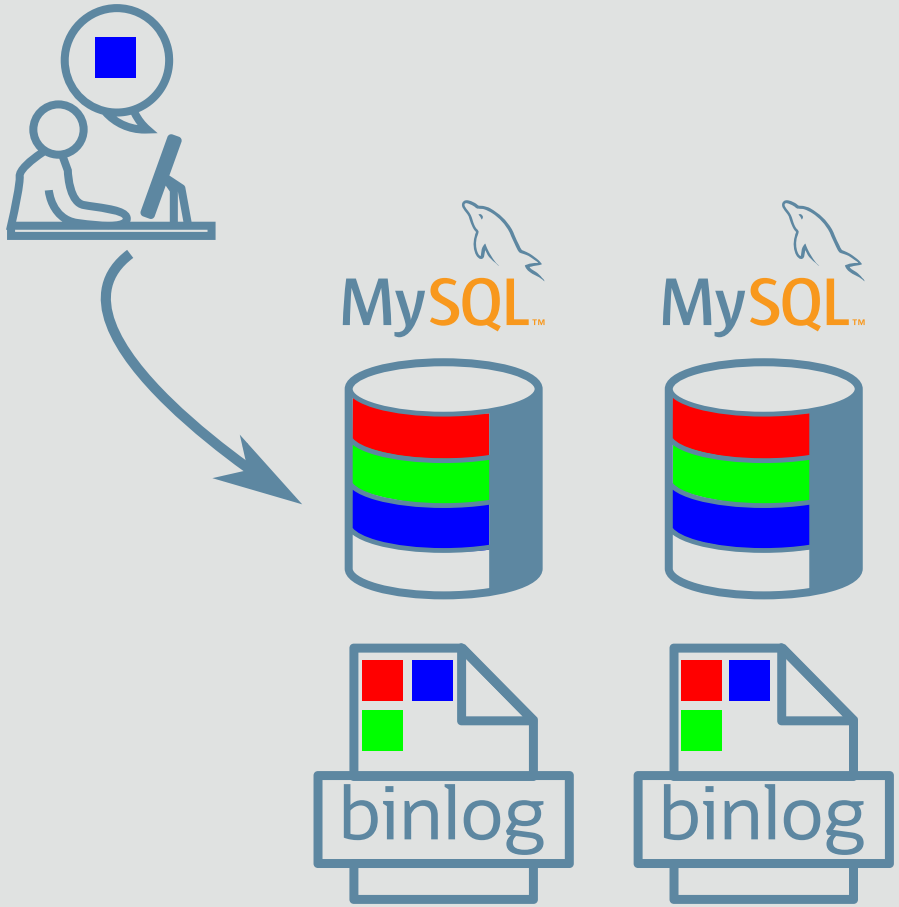
MySQL InnoDB Cluster Provisioning



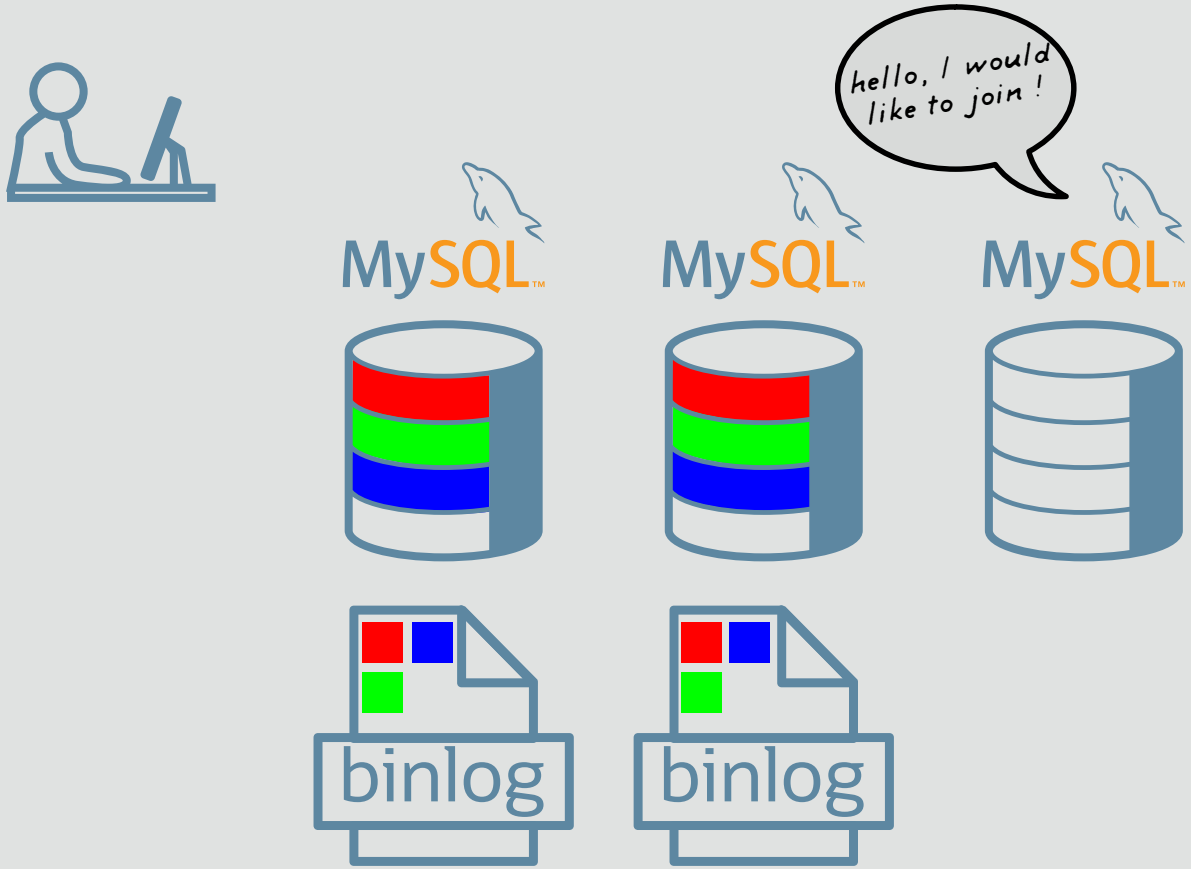
MySQL InnoDB Cluster Provisioning



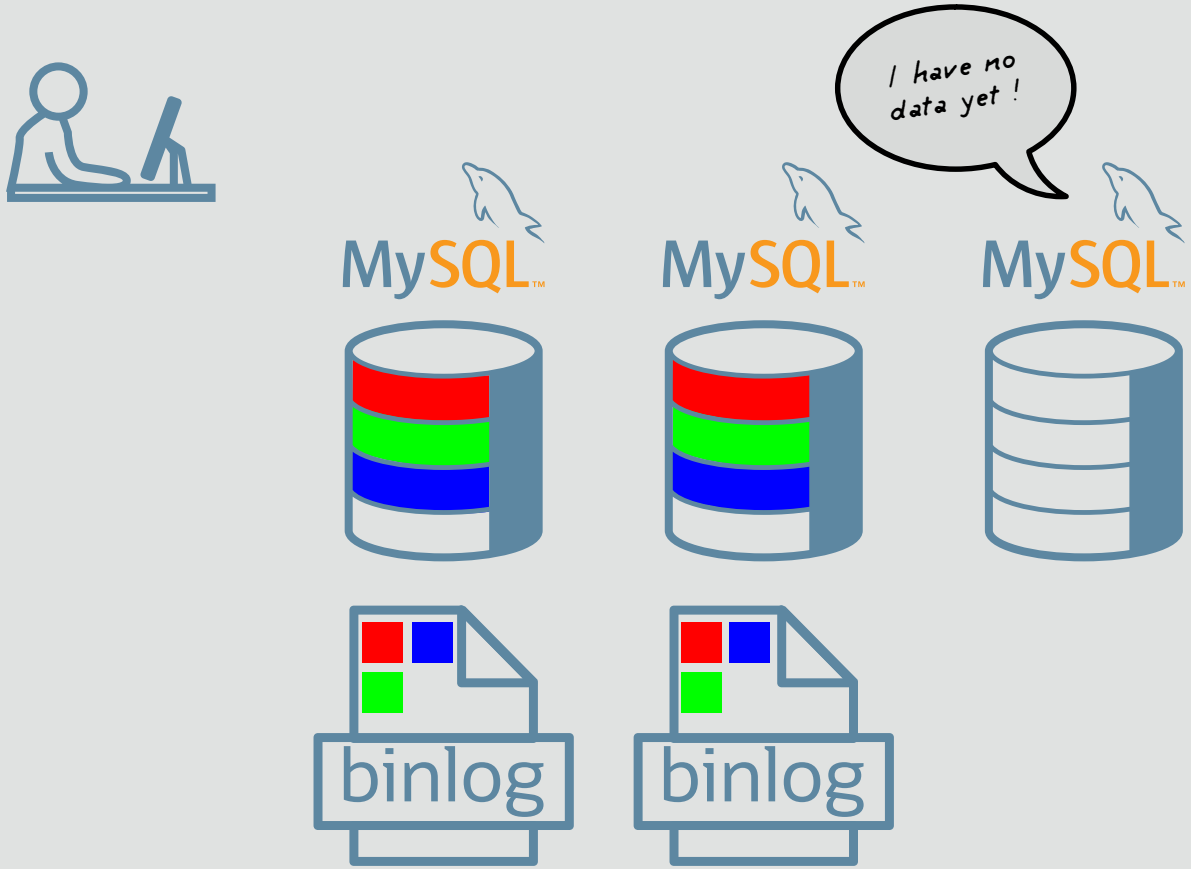
MySQL InnoDB Cluster Provisioning



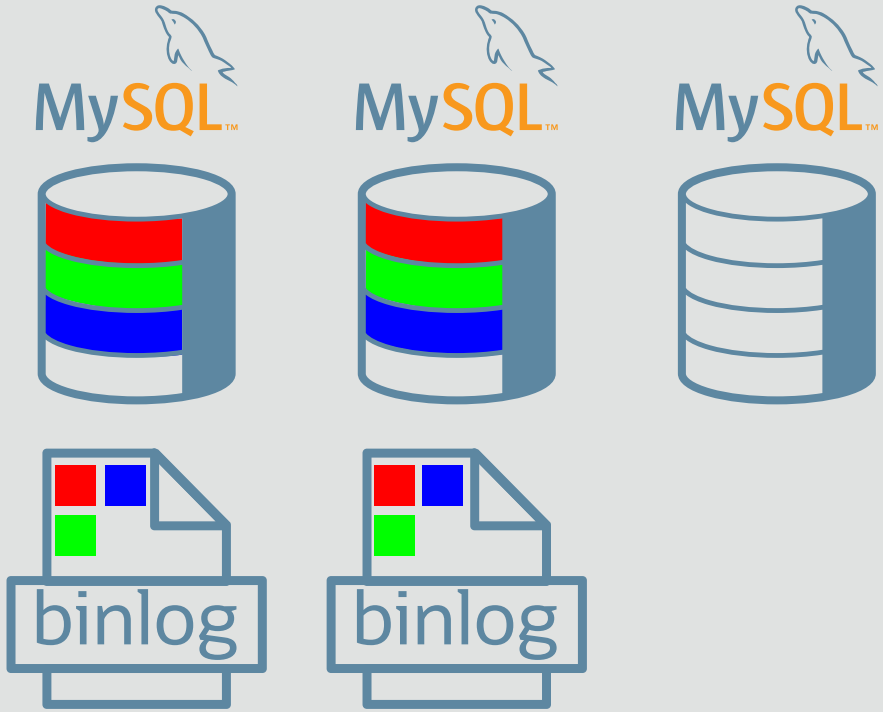
MySQL InnoDB Cluster Provisioning



MySQL InnoDB Cluster Provisioning

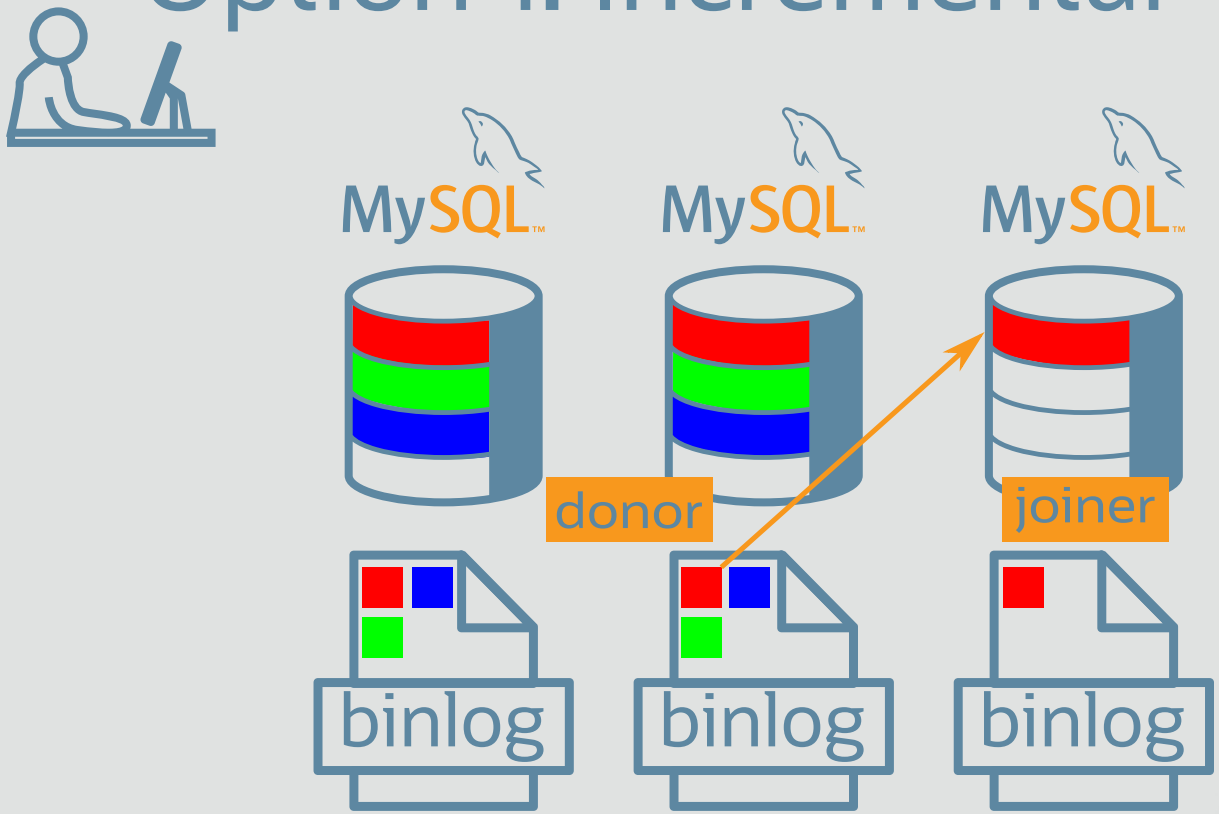


MySQL InnoDB Cluster Provisioning



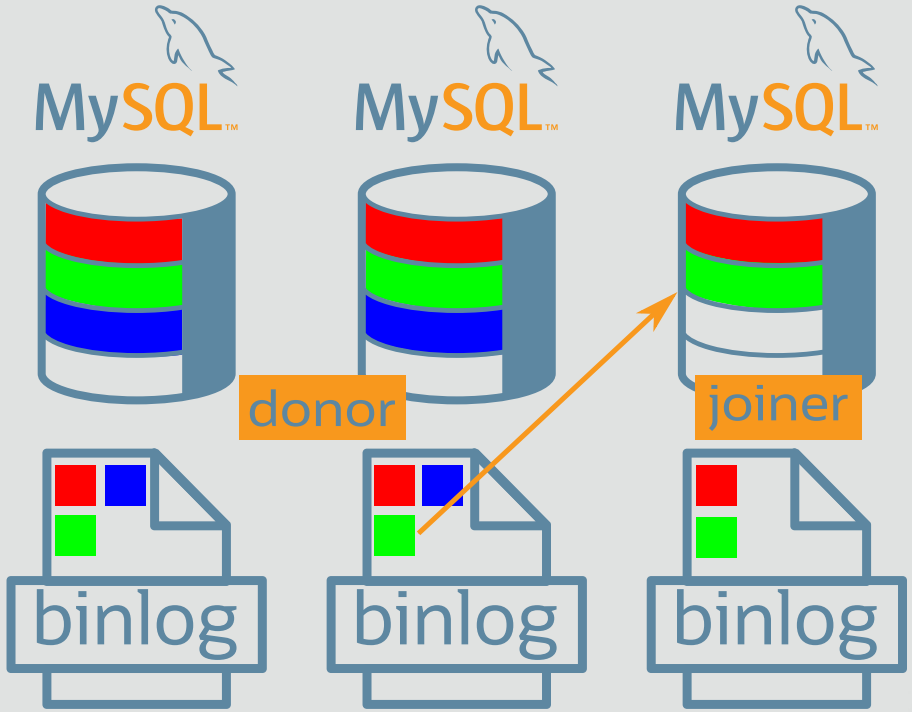
MySQL InnoDB Cluster Provisioning

Option 1: Incremental



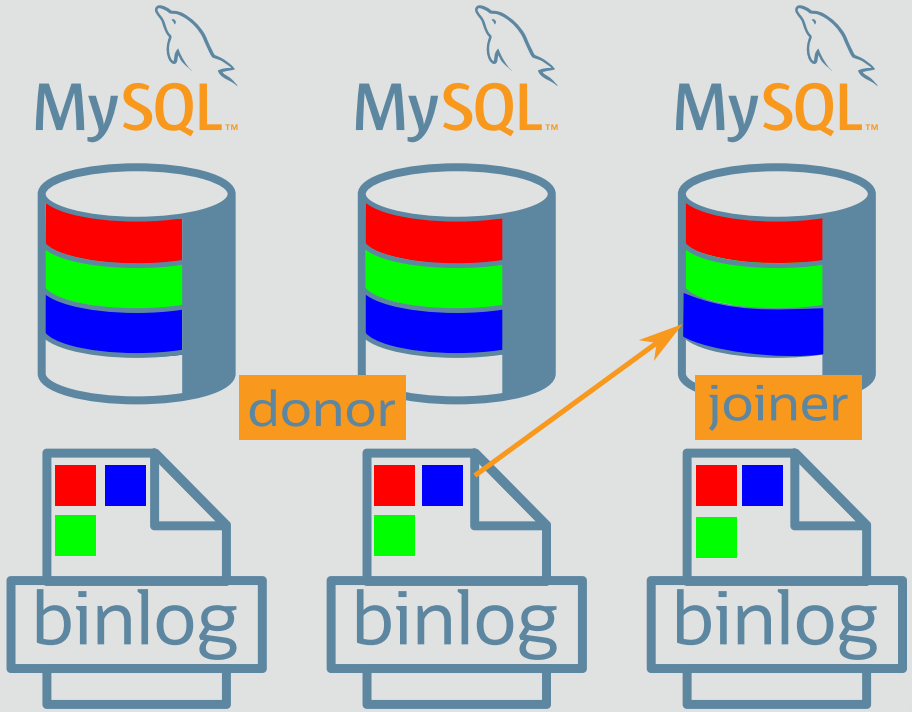
MySQL InnoDB Cluster Provisioning

Option 1: Incremental



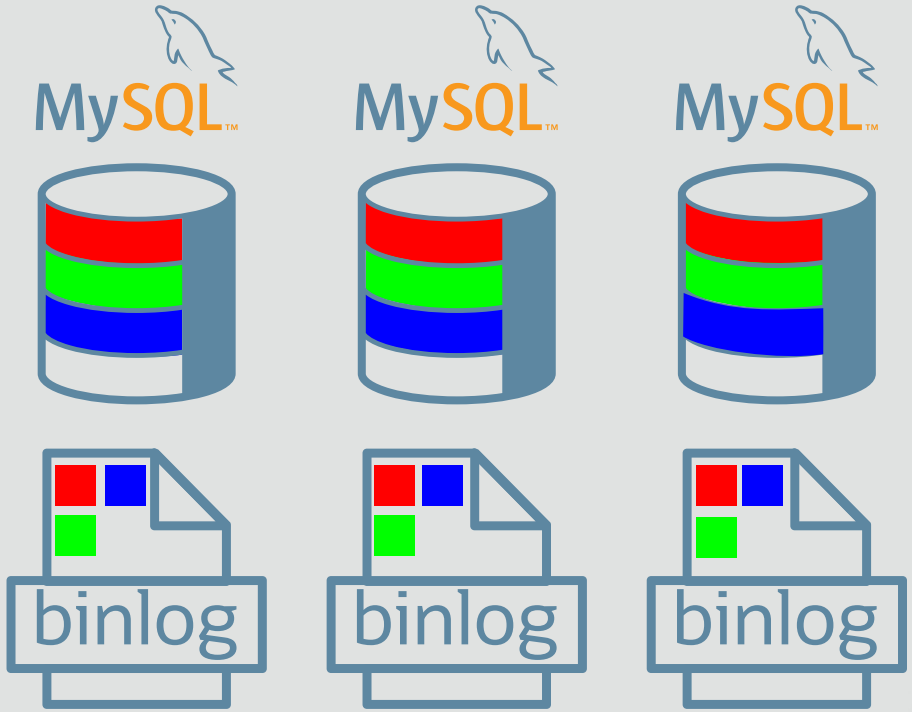
MySQL InnoDB Cluster Provisioning

Option 1: Incremental



MySQL InnoDB Cluster Provisioning

Option 1: Incremental

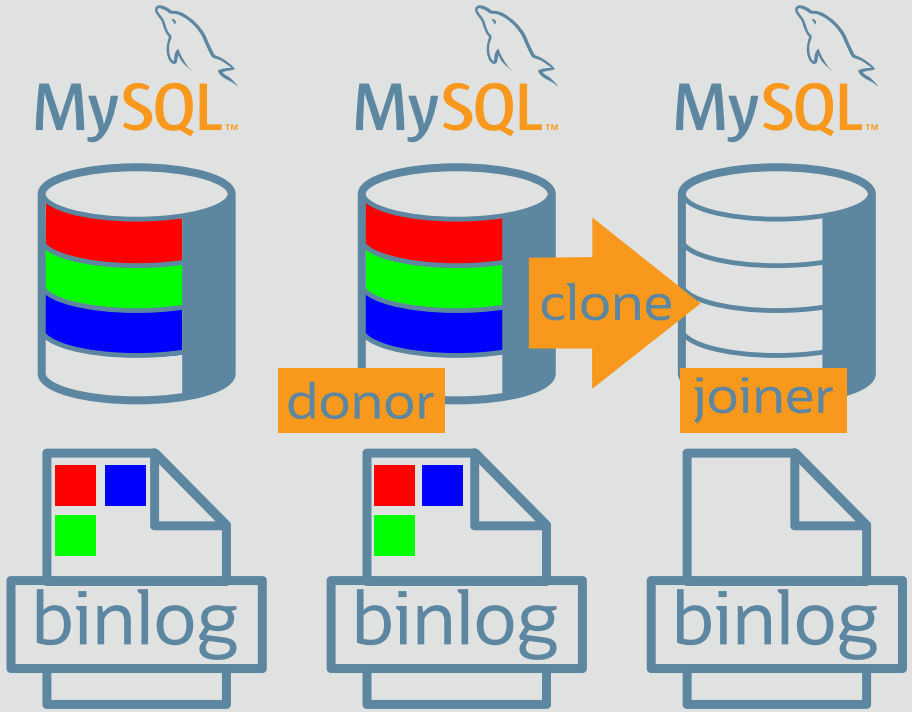


MySQL InnoDB Cluster Provisioning

```
MySQL 8.0.17  mysql1:33060+  2019-09-04 21:30:07
JS cluster.addInstance('clusteradmin@mysql2')

NOTE: The target instance 'mysql2:3306' has not been pre-provisioned (GTID set is
empty). The Shell is unable to decide whether incremental distributed state
recovery can correctly provision it.
The safest and most convenient way to provision a new instance is through
automatic clone provisioning, which will completely overwrite the state of
'mysql2:3306' with a physical snapshot from an existing cluster member. To use
this method by default, set the 'recoveryMethod' option to 'clone'.

The incremental distributed state recovery may be safely used if you are sure
all updates ever executed in the cluster were done with GTIDs enabled, there
are no purged transactions and the new instance contains the same GTID set as
the cluster or a subset of it. To use this method by default, set the
'recoveryMethod' option to 'incremental'.


Please select a recovery method [C]lone/[I]ncremental recovery/[A]bort (default Clone):
```

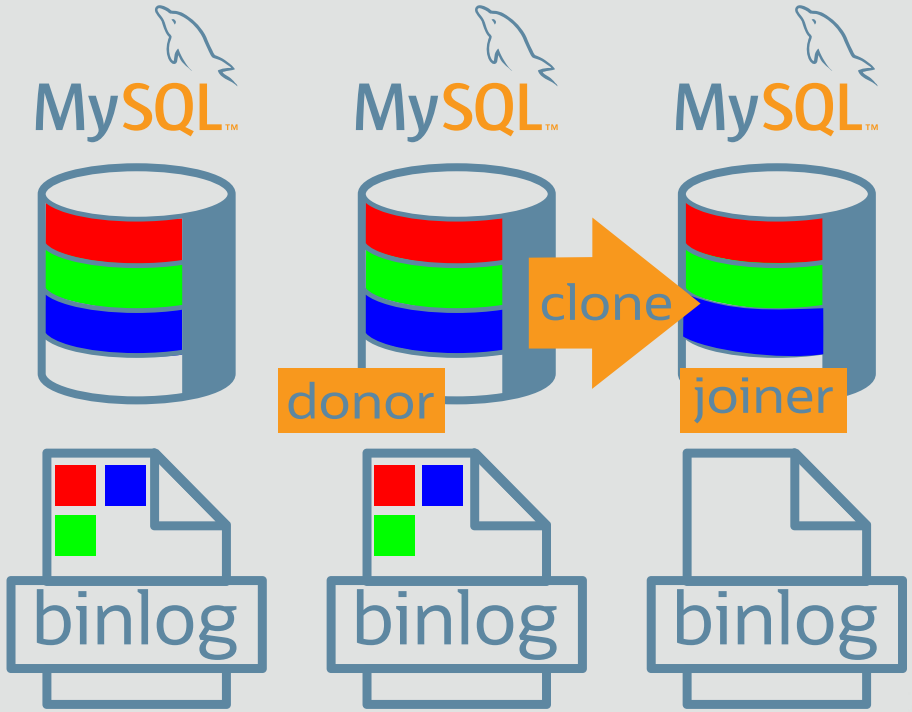
MySQL InnoDB Cluster Provisioning

Option 2: Clone

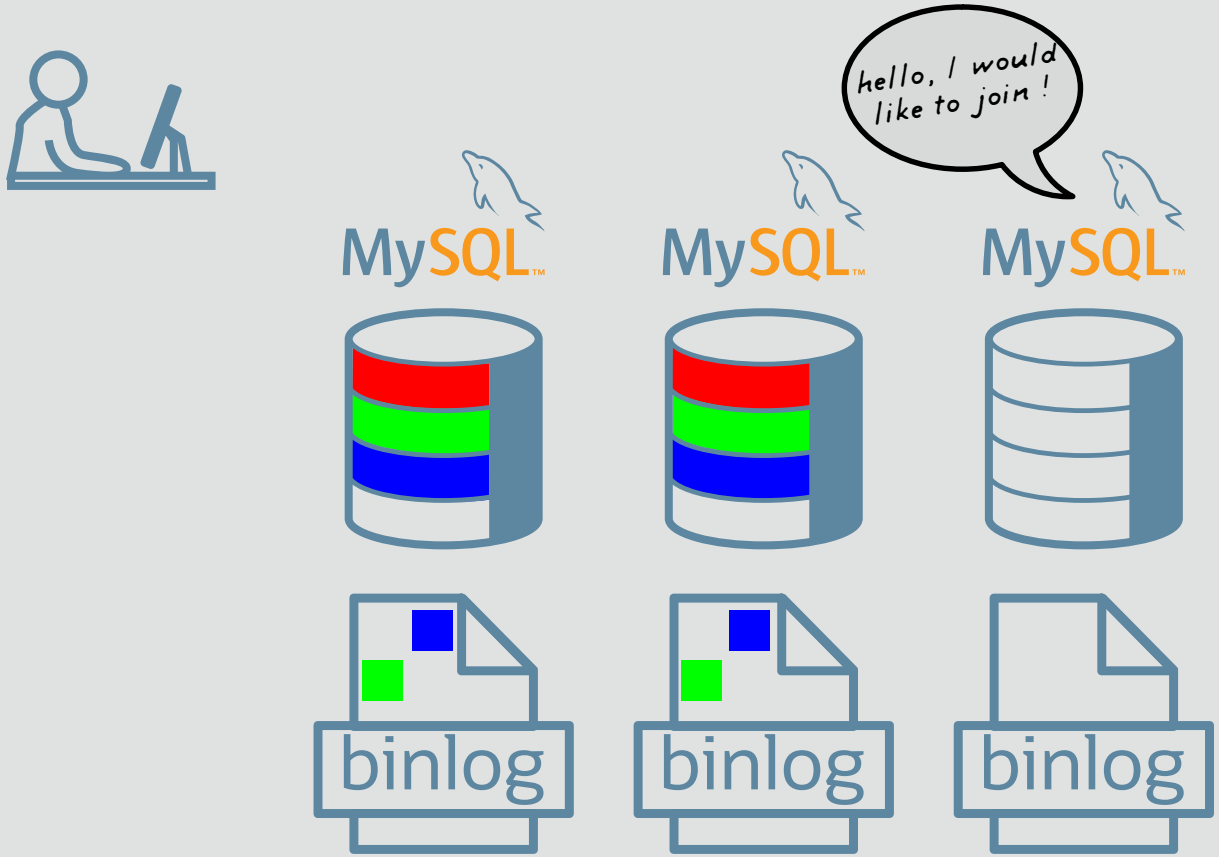


MySQL InnoDB Cluster Provisioning

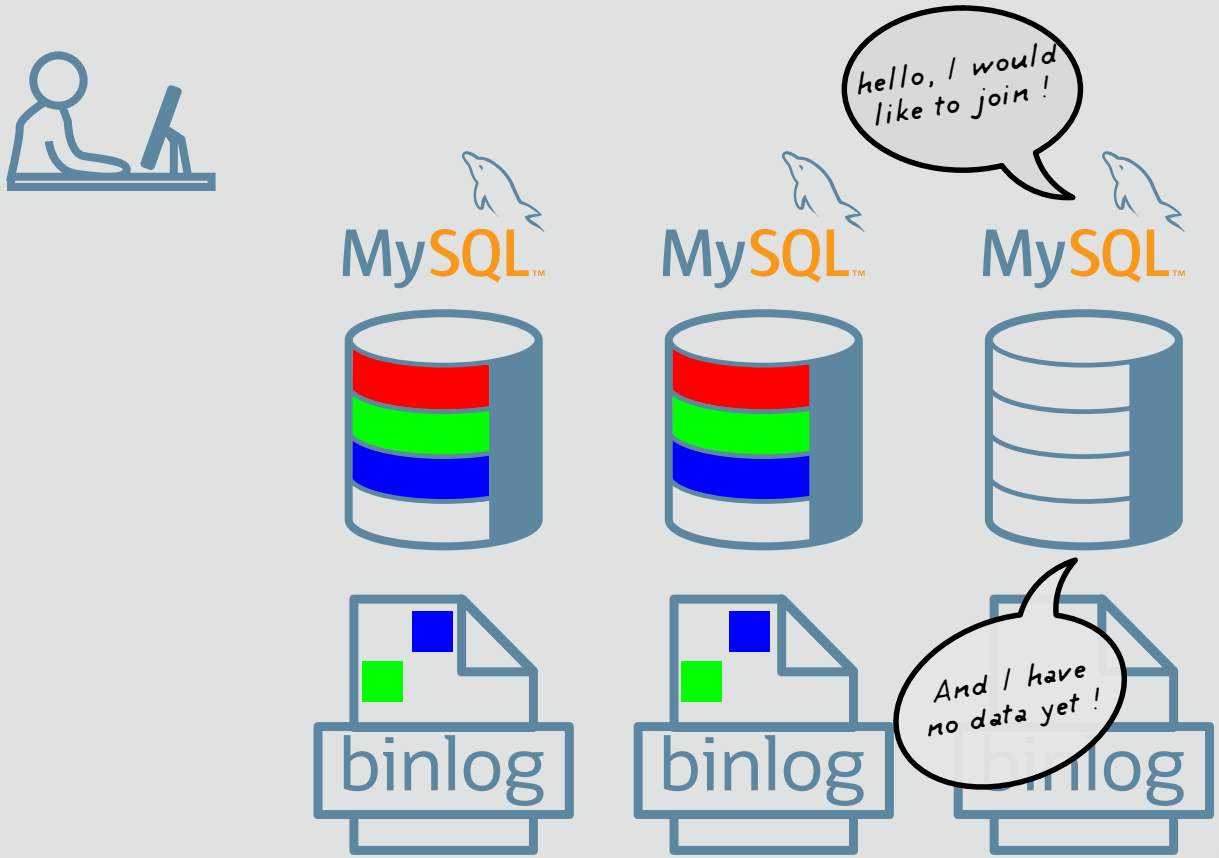
Option 2: Clone



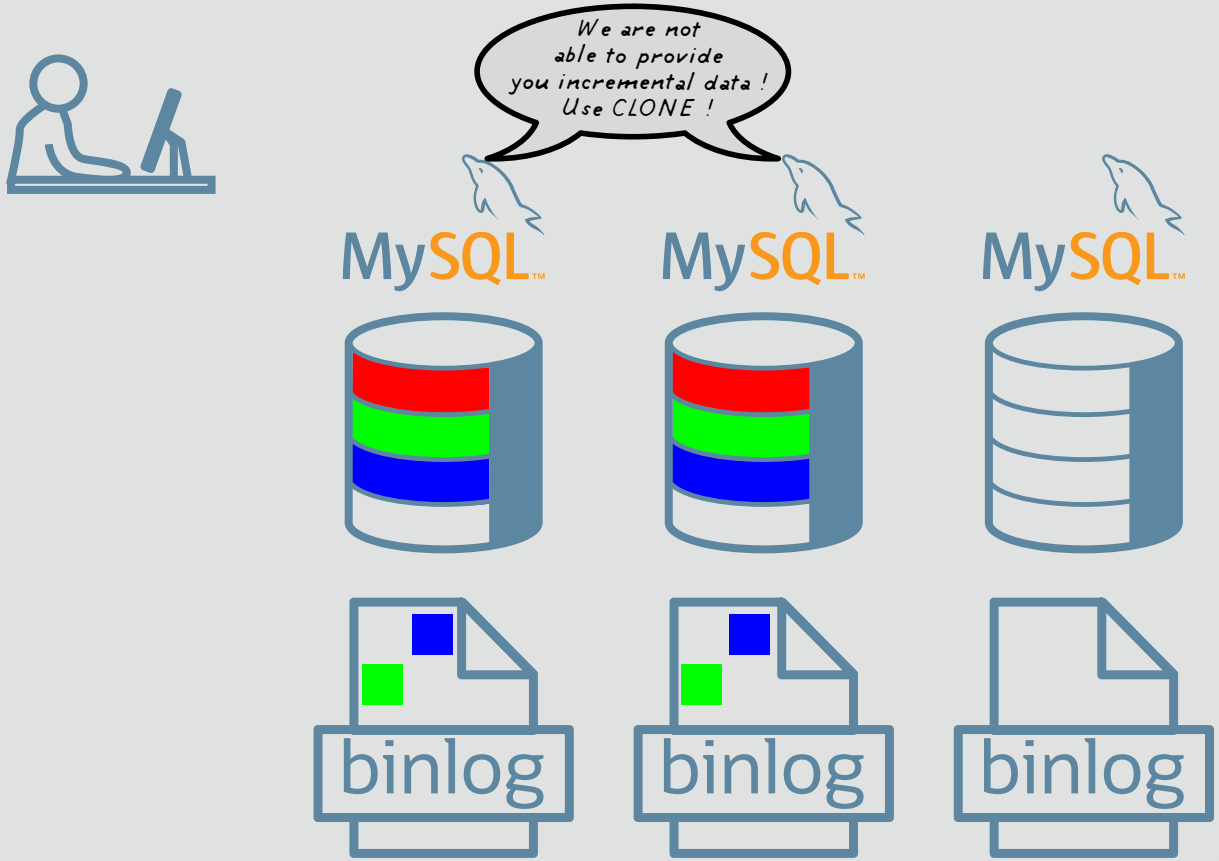
MySQL InnoDB Cluster Provisioning



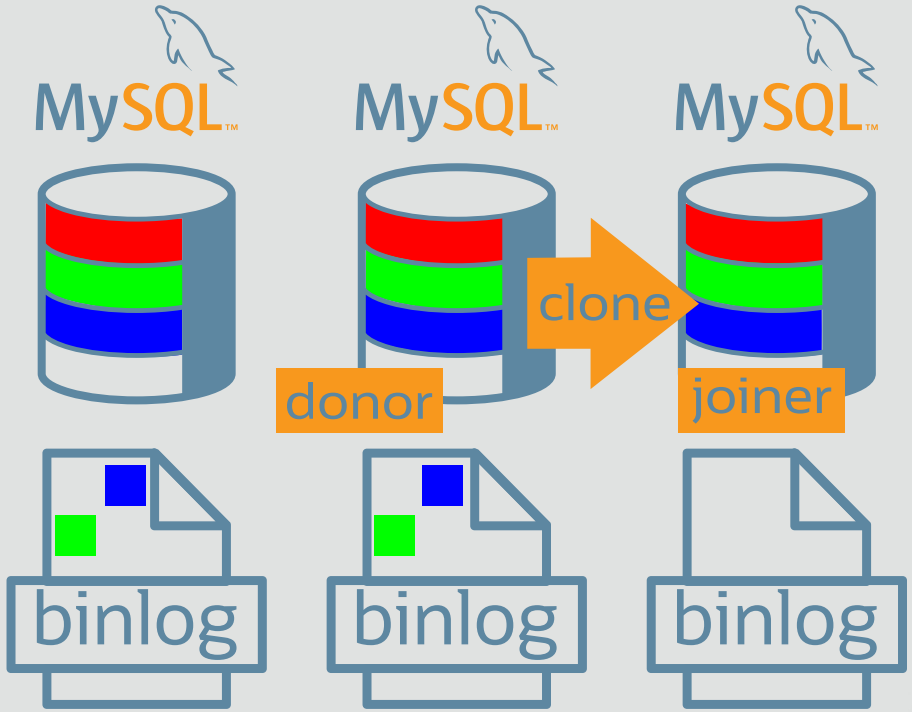
MySQL InnoDB Cluster Provisioning



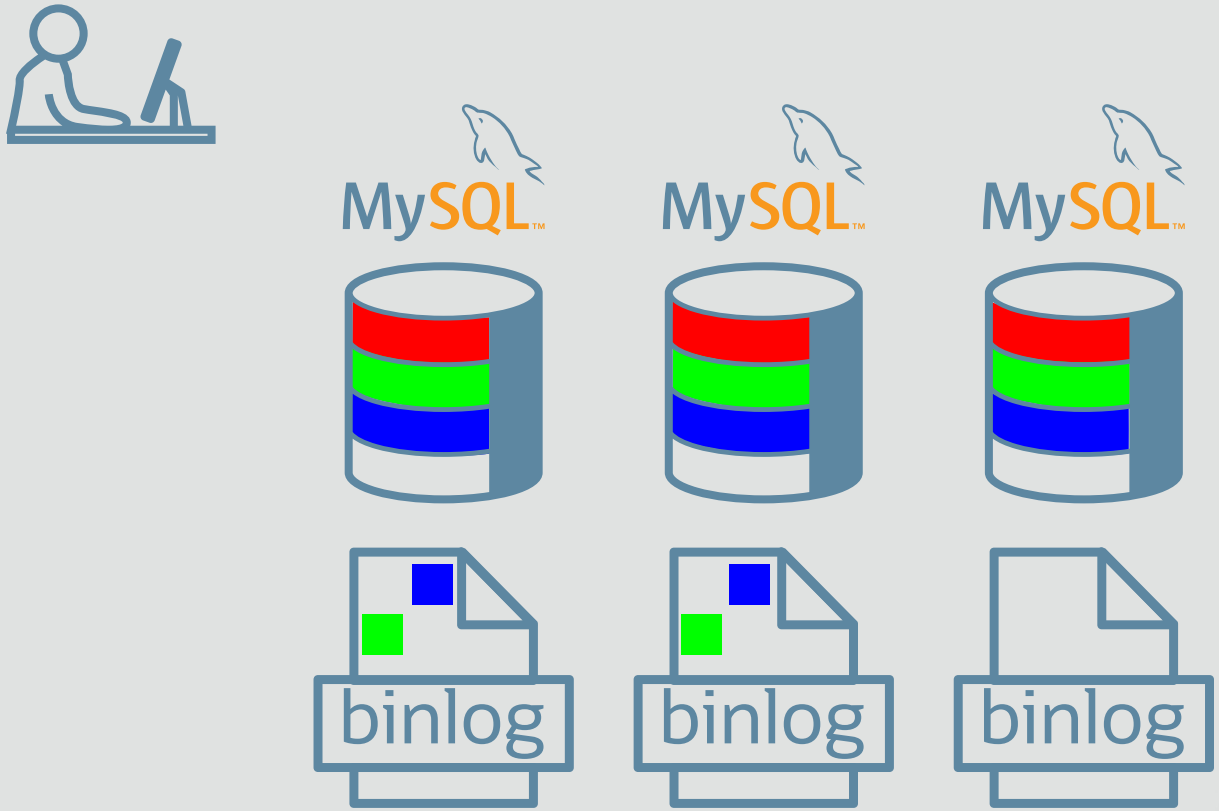
MySQL InnoDB Cluster Provisioning



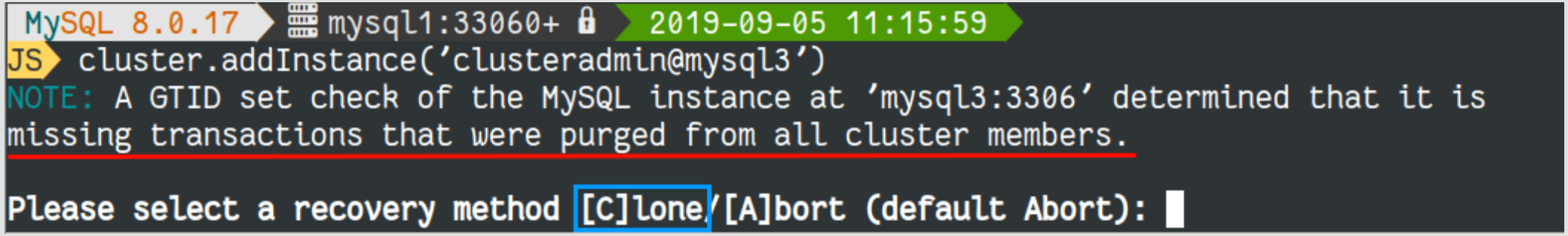
MySQL InnoDB Cluster Provisioning



MySQL InnoDB Cluster Provisioning

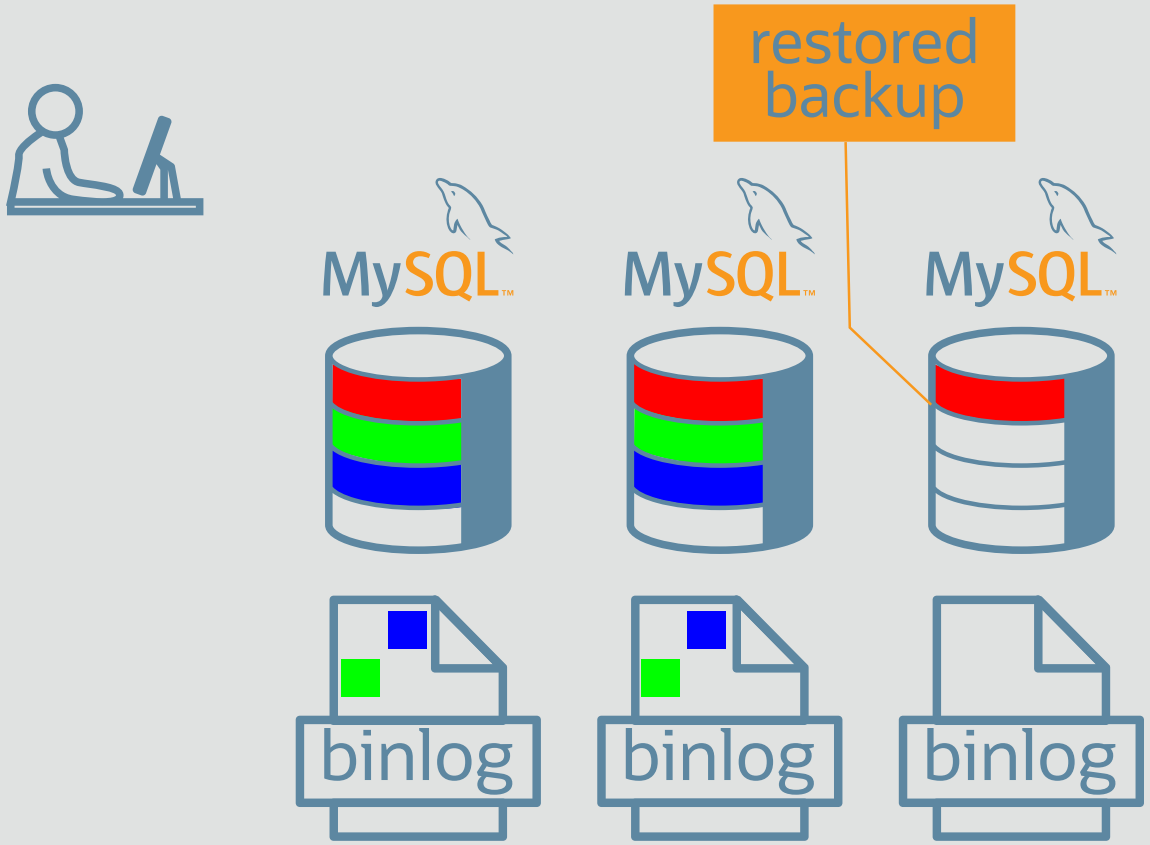


MySQL InnoDB Cluster Provisioning



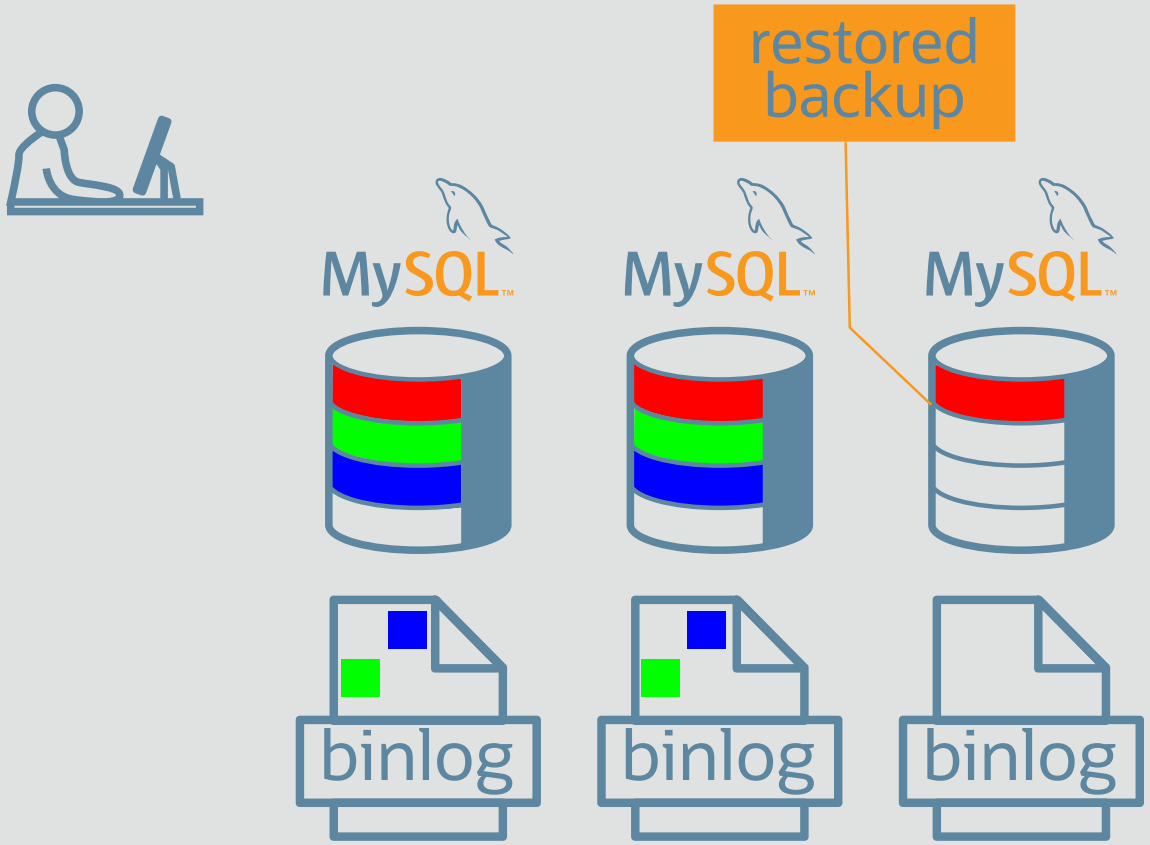
MySQL InnoDB Cluster Provisioning

What kind of provisioning will happen ?



MySQL InnoDB Cluster Provisioning

What kind of provisioning will happen ?



Incremental Recovery !

MySQL InnoDB Cluster Provisioning

```
MySQL 8.0.17  mysql2:33060+  2019-09-04 20:55:49
JS cluster.addInstance('clusteradmin@mysql3')
The safest and most convenient way to provision a new instance is through
automatic clone provisioning, which will completely overwrite the state of
'mysql3:3306' with a physical snapshot from an existing cluster member. To use
this method by default, set the 'recoveryMethod' option to 'clone'.

The incremental distributed state recovery may be safely used if you are sure
all updates ever executed in the cluster were done with GTIDs enabled, there
are no purged transactions and the new instance contains the same GTID set as
the cluster or a subset of it. To use this method by default, set the
'recoveryMethod' option to 'incremental'.

Incremental distributed state recovery was selected because it seems to be safely usable.
Validating instance at mysql3:3306...

This instance reports its own address as mysql3:3306

Instance configuration is suitable.
A new instance will be added to the InnoDB cluster. Depending on the amount of
data on the cluster this might take from a few seconds to several hours.

Adding instance to the cluster...

Monitoring recovery process of the new cluster member. Press ^C to stop monitoring and let it continue in background.
State recovery already finished for 'mysql3:3306'

The instance 'mysql3' was successfully added to the cluster.
```

MySQL InnoDB Cluster Provisioning

As a DBA, I want to force the use of Clone when adding an instance even if Incremental Recovery is possible.

MySQL InnoDB Cluster Provisioning

As a DBA, I want to force the use of Clone when adding an instance even if Incremental Recovery is possible.

Just specify it in the MySQL Shell:

```
MySQL 8.0.17  mysql1:33060+  2019-09-07 08:03:46
JS cluster.addInstance('mysql3:3306',{recoveryMethod: 'clone'})

Clone based recovery selected through the recoveryMethod option
Validating instance at mysql3:3306...

This instance reports its own address as mysql3:3306
```

Recovery Process

The recovery process happens when a previous member joins back a Cluster/Group.

By default and if possible (based on GTID set), the *Incremental Recovery* method is used.

However, sometimes applying a large amount of binary log events might be slower than performing a **Clone**. You have the possibility to configure the treshold: `group_replication_clone_threshold` (default is 9223372036854775807 !):

```
MySQL 8.0.17  mysql1:33060+  2019-09-07 08:01:37
JS \sql set global group_replication_clone_threshold=10000
Query OK, 0 rows affected (0.0197 sec)
```

Recovery Process (2)

As a DBA, I want to force the use of Clone when rejoining an instance even if Incremental Recovery is enough to recover.

Recovery Process (2)

As a DBA, I want to force the use of Clone when rejoining an instance even if Incremental Recovery is enough to recover.

It's possible ! You need to connect to the instance you want to rejoin and set `group_replication_clone_threshold` to **1**.

As **Clone** will be called and the change is not persisted, no need to set it back to its original value. **Never persist it to 1 or to a really low value on a live Group, never!**

Recovery Process (3)

```
 MySQL 8.0.17  mysql1:33060+  2019-09-07 08:11:29
JS \c clusteradmin@mysql3
Creating a session to 'clusteradmin@mysql3'
Fetching schema names for autocompletion... Press ^C to stop.
Closing old connection...
Your MySQL connection id is 165 (X protocol)
Server version: 8.0.17 MySQL Community Server - GPL
No default schema selected; type \use <schema> to set one.
 MySQL 8.0.17  mysql3:33060+  2019-09-07 08:13:09
JS \sql set global group_replication_clone_threshold=1
Query OK, 0 rows affected (0.0008 sec)
 MySQL 8.0.17  mysql3:33060+  2019-09-07 08:13:12
JS cluster.rejoinInstance('clusteradmin@mysql3:3306')
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 'mysql3:3306' was successfully rejoined on the cluster.
```

Force a Donor during Clone

As a DBA, I want to add a node (`mysql3`*) but it should only clone from server* `mysql2`

Force a Donor during Clone

As a DBA, I want to add a node (`mysql3`*) but it should only clone from server* `mysql2`

This is again possible, you need to connect on the other members (in this case `mysql1` and unload the clone plugin).

Please note, that to be able to unload the plugin you need to be connected using the classic protocol (3306).

Force a Donor during Clone (2)

We connect using the classic protocol and we uninstall the plugin:

```
MySQL 8.0.17  mysql1:33060+  2019-09-07 10:42:32
JS \c clusteradmin@mysql1:3306
Creating a session to 'clusteradmin@mysql1:3306'
Fetching schema names for autocompletion... Press ^C to stop.
Closing old connection...
Your MySQL connection id is 406936
Server version: 8.0.17 MySQL Community Server - GPL
No default schema selected; type \use <schema> to set one.
MySQL 8.0.17  mysql1:3306  2019-09-07 10:42:38
JS \sql uninstall plugin clone
Query OK, 0 rows affected (1.3264 sec)
```

Force a Donor during Clone (2)

We connect using the classic protocol and we uninstall the plugin:

```
MySQL 8.0.17  mysql1:33060+  2019-09-07 10:42:32
JS \c clusteradmin@mysql1:3306
Creating a session to 'clusteradmin@mysql1:3306'
Fetching schema names for autocompletion... Press ^C to stop.
Closing old connection...
Your MySQL connection id is 406936
Server version: 8.0.17 MySQL Community Server - GPL
No default schema selected; type \use <schema> to set one.
MySQL 8.0.17  mysql1:3306  2019-09-07 10:42:38
JS \sql uninstall plugin clone
Query OK, 0 rows affected (1.3264 sec)
```

Then we add the instance forcing **Clone**:

```
MySQL 8.0.17  mysql1:3306  2019-09-07 10:42:44
JS \c clusteradmin@mysql1
Creating a session to 'clusteradmin@mysql1'
Fetching schema names for autocompletion... Press ^C to stop.
Closing old connection...
Your MySQL connection id is 406974 (X protocol)
Server version: 8.0.17 MySQL Community Server - GPL
No default schema selected; type \use <schema> to set one.
MySQL 8.0.17  mysql1:33060+  2019-09-07 10:43:00
JS cluster.addInstance('mysql3:3306',{recoveryMethod: 'clone'})

Clone based recovery selected through the recoveryMethod option
Validating instance at mysql3:3306...

This instance reports its own address as mysql3:3306
```

Force a Donor during Clone (3)

```
NOTE: A server restart is expected to happen as part of the clone process. If the
server does not support the RESTART command or does not come back after a
while, you may need to manually start it back.

* Waiting for clone to finish...
NOTE: mysql3:3306 is being cloned from mysql2:3306
** Stage DROP DATA: Completed
** Clone Transfer
    FILE COPY  ##################################################################  100%  Completed
    PAGE COPY  ##################################################################  100%  Completed
    REDO COPY  ##################################################################  100%  Completed
** Stage RECOVERY: \
NOTE: mysql3:3306 is shutting down...
```

Force a Donor during Clone (4)

We can also verify this on the new joiner `Performance_Schema.clone_status` table:

```
MySQL 8.0.17  mysql3:33060+  2019-09-07 10:47:06
JS \sql  select * from performance_schema.clone_status\G
*************************** 1. row ***************************
            ID: 1
           PID: 0
         STATE: Completed
    BEGIN_TIME: 2019-09-07 10:43:50.362
      END_TIME: 2019-09-07 10:44:09.883
        SOURCE: mysql2:3306
   DESTINATION: LOCAL INSTANCE
      ERROR_NO: 0
 ERROR_MESSAGE:
   BINLOG_FILE: binlog.000003
BINLOG_POSITION: 159186
  GTID_EXECUTED: 26c11a7e-cf5b-11e9-a38b-08002718d305:1-1421
1 row in set (0.0010 sec)
```

Force a Donor during Clone (5)

Don't forget to install the **Clone** plugin again:

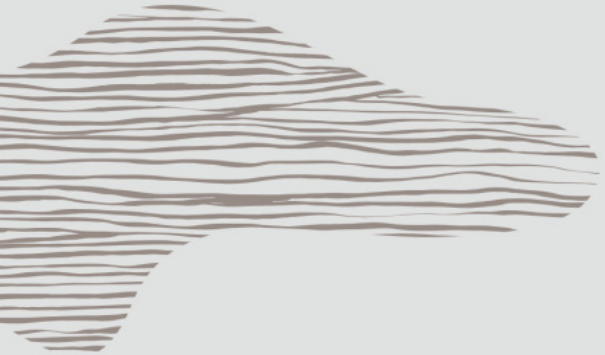
```
MySQL 8.0.17  mysql3:33060+  2019-09-07 10:47:10
JS \c clusteradmin@mysql1:3306
Creating a session to 'clusteradmin@mysql1:3306'
Fetching schema names for autocompletion... Press ^C to stop.
Closing old connection...
Your MySQL connection id is 407472
Server version: 8.0.17 MySQL Community Server - GPL
No default schema selected; type \use <schema> to set one.
MySQL 8.0.17  mysql1:3306  2019-09-07 10:47:40
JS \sql install plugin clone soname 'mysql_clone.so';
Query OK, 0 rows affected (0.3520 sec)
```

Force a Donor during Clone (5)

Don't forget to install the **Clone** plugin again:

```
MySQL 8.0.17  mysql3:33060+  2019-09-07 10:47:10
JS \c clusteradmin@mysql1:3306
Creating a session to 'clusteradmin@mysql1:3306'
Fetching schema names for autocompletion... Press ^C to stop.
Closing old connection...
Your MySQL connection id is 407472
Server version: 8.0.17 MySQL Community Server - GPL
No default schema selected; type \use <schema> to set one.
MySQL 8.0.17  mysql1:3306  2019-09-07 10:47:40
JS \sql install plugin clone soname 'mysql_clone.so';
Query OK, 0 rows affected (0.3520 sec)
```

```
MySQL 8.0.17  mysql1:33060+  2019-10-01 11:58:33
JS cluster.setOption('disableClone', false)
Setting the value of 'disableClone' to 'false' in the Cluster ...

Successfully set the value of 'disableClone' to 'false' in the Cluster:
```

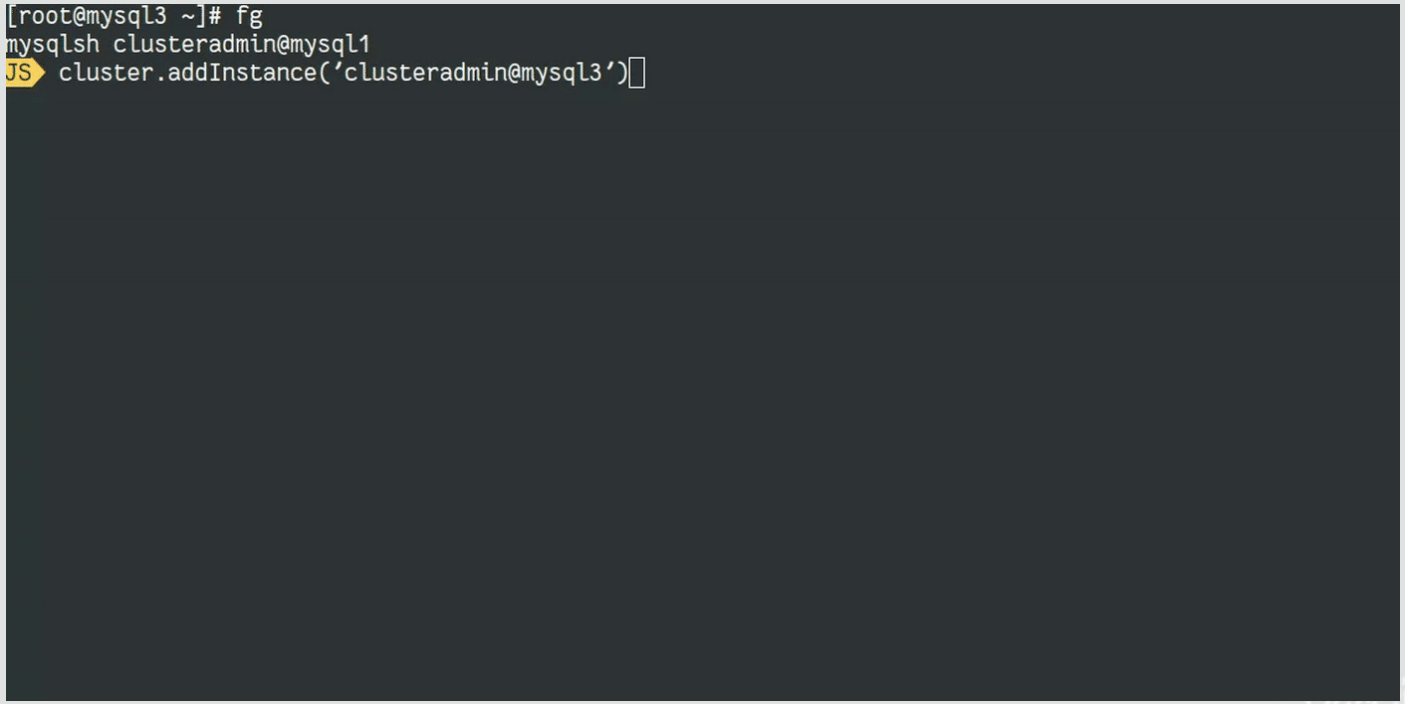


MySQL InnoDB Cluster

Monitoring

Monitoring Provisioning & Recovery process

Currently, when the CLONE is triggered within MySQL Shell, you can follow its progress:



Monitoring Provisioning & Recovery process

And we already saw that the status and the progress is also available in `performance_schema` tables on the joiner:

```
MySQL 8.0.17  mysql3:33060+  performance_schema  2019-09-10 11:54:11
SQL> select * from clone_progress;
+----+-----------+-----------+----------------------------+----------------------------+---------+----------+----------+----------+
| ID | STAGE     | STATE     | BEGIN_TIME                 | END_TIME                   | THREADS | ESTIMATE | DATA     | NETWORK  |
+----+-----------+-----------+----------------------------+----------------------------+---------+----------+----------+----------+
|  1 | DROP DATA | Completed | 2019-09-10 11:24:51.515504 | 2019-09-10 11:24:52.405911 |       1 |        0 |        0 |        0 |
|  1 | FILE COPY | Completed | 2019-09-10 11:24:52.406392 | 2019-09-10 11:24:53.289564 |       1 | 63066235 | 63066235 | 63078288 |
|  1 | PAGE COPY | Completed | 2019-09-10 11:24:53.290006 | 2019-09-10 11:24:53.414982 |       1 |        0 |        0 |       99 |
|  1 | REDO COPY | Completed | 2019-09-10 11:24:53.415236 | 2019-09-10 11:24:53.518025 |       1 |     4096 |     4096 |     4493 |
|  1 | FILE SYNC | Completed | 2019-09-10 11:24:53.518243 | 2019-09-10 11:24:53.82337  |       1 |        0 |        0 |        0 |
|  1 | RESTART   | Completed | 2019-09-10 11:24:53.82337  | 2019-09-10 11:25:05.4309   |       0 |        0 |        0 |        0 |
|  1 | RECOVERY  | Completed | 2019-09-10 11:25:05.4309   | 2019-09-10 11:25:06.841659 |       0 |        0 |        0 |        0 |
+----+-----------+-----------+----------------------------+----------------------------+---------+----------+----------+----------+
```

Monitoring Provisioning & Recovery process

```
MySQL 8.0.17 mysql3:33060+ performance_schema 2019-09-10 11:54:45
SQL select * from clone_status\G
*************************** 1. row ***************************
           ID: 1
          PID: 0
        STATE: Completed
   BEGIN_TIME: 2019-09-10 11:24:51.515
     END_TIME: 2019-09-10 11:25:06.842
       SOURCE: mysql2:3306
  DESTINATION: LOCAL INSTANCE
     ERROR_NO: 0
ERROR_MESSAGE:
  BINLOG_FILE: binlog.000006
BINLOG_POSITION: 1805
 GTID_EXECUTED: 1610e773-d323-11e9-a38b-08002718d305:1-3102:1000032-1000165,
26c11a7e-cf5b-11e9-a38b-08002718d305:1-1588:1001542-1001548
```

Incremental Recovery process

It's also possible to evaluate the amount of transactions a member needs to process before being in sync with the Group when joining. You can find a User-Defined Report to see that amount :

```
MySQL 8.0.17  mysql3:33060+  2019-09-10 12:13:01
JS \show gr_recovery_progress
+-----------------+
| trx_to_recover  |
+-----------------+
| 599             |
+-----------------+
```

More Info: https://lefred.be/content/mysql-innodb-cluster-recovery-process-monitoring-with-the-mysql-shell-reporting-framework/

MySQL InnoDB Cluster - status()

```
MySQL 8.0.17  mysql1:33060+  2019-09-08 20:25:14
JS cluster.status()
{
    "clusterName": "PerconaLiveEU",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "mysql1:3306",
        "ssl": "REQUIRED",
        "status": "OK",
        "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
        "topology": {
            "mysql1:3306": {
                "address": "mysql1:3306",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.17"
            },
            "mysql2:3306": {
                "address": "mysql2:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.17"
            },
            "mysql3:3306": {
                "address": "mysql3:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.16"
            }
        },
        "topologyMode": "Single-Primary"
    },
    "groupInformationSourceMember": "mysql1:3306"
}
```

By default the `status()` method disables the extended mode.

It's possible to have much more information enabling it.

MySQL InnoDB Cluster - status({extended})

The `extended` option allows 4 values (or boolean):

- 0: disables the command verbosity (default)
- 1: includes information about the Group Protocol Version, Group name, cluster member UUIDs, cluster member roles and states as reported by Group Replication and the list of fenced system variables
- 2: includes information about transactions processed by connection and applier
- 3: includes more detailed stats about the replication machinery of each cluster member

MySQL InnoDB Cluster - status({extended:1})

```
MySQL 8.0.17  mysql1:33060+  2019-09-08 20:38:41
JS cluster.status({extended:1})
{
    "clusterName": "PerconaLiveEU",
    "defaultReplicaSet": {
        "GRProtocolVersion": "8.0.16",
        "groupName": "26c11a7e-cf5b-11e9-a38b-08002718d305",
        "name": "default",
        "primary": "mysql1:3306",
        "ssl": "REQUIRED",
        "status": "OK",
        "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
        "topology": {
            "mysql1:3306": {
                "address": "mysql1:3306",
                "fenceSysVars": [],
                "memberId": "c88e9182-cf5a-11e9-8d12-08002718d305",
                "memberRole": "PRIMARY",
                "memberState": "ONLINE",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.17"
            },
            "mysql2:3306": {
```

MySQL InnoDB Cluster - status({extended:2})


```
"status": "OK",
"statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
"topology": {
    "mysql1:3306": {
        "address": "mysql1:3306",
        "fenceSysVars": [],
        "memberId": "c88e9182-cf5a-11e9-8d12-08002718d305",
        "memberRole": "PRIMARY",
        "memberState": "ONLINE",
        "mode": "R/W",
        "readReplicas": {},
        "role": "HA",
        "status": "ONLINE",
        "transactions": {
            "appliedCount": 73,
            "checkedCount": 1486,
            "committedAllMembers": "26c11a7e-cf5b-11e9-a38b-08002718d305:1-1546:1001542-1001548",
            "conflictsDetectedCount": 0,
            "inApplierQueueCount": 0,
            "inQueueCount": 0,
            "lastConflictFree": "26c11a7e-cf5b-11e9-a38b-08002718d305:1001548",
            "proposedCount": 1479,
            "rollbackCount": 0
        },
        "version": "8.0.17"
```

MySQL InnoDB Cluster - status({extended:3})


```
                "role": "HA",
                "status": "ONLINE",
                "transactions": {
                    "appliedCount": 2,
                    "checkedCount": 28,
                    "committedAllMembers": "1610e773-d323-11e9-a38b-08002718d305:1-31,
26c11a7e-cf5b-11e9-a38b-08002718d305:1-1588:1001542-1001548",
                    "conflictsDetectedCount": 0,
                    "connection": {
                        "lastHeartbeatTimestamp": "",
                        "lastQueued": {
                            "endTimestamp": "2019-09-09 21:38:00.880788",
                            "immediateCommitTimestamp": "",
                            "immediateCommitToEndTime": null,
                            "originalCommitTimestamp": "",
                            "originalCommitToEndTime": null,
                            "queueTime": 0.000157,
                            "startTimestamp": "2019-09-09 21:38:00.880631",
                            "transaction": "1610e773-d323-11e9-a38b-08002718d305:27"
                        },
                        "receivedHeartbeats": 0,
                        "receivedTransactionSet": "1610e773-d323-11e9-a38b-08002718d305:1-31,
26c11a7e-cf5b-11e9-a38b-08002718d305:1-1588:1001542-1001548",
                        "threadId": null
                    },
                    "inApplierQueueCount": 0,
```

Monitoring MySQL InnoDB Cluster - Performance_Schema

Other than the `status()` in the MySQL Shell, it's possible to get information from `Performance_Schema` tables:

```
MySQL 8.0.17  mysql3:33060+  performance_schema  2019-09-10 12:43:11
SQL  select * from replication_group_members;
+---------------------------+--------------------------------------+-------------+-------------+--------------+-------------+----------------+
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE | MEMBER_ROLE | MEMBER_VERSION |
+---------------------------+--------------------------------------+-------------+-------------+--------------+-------------+----------------+
| group_replication_applier | b016dfa7-d31b-11e9-8e21-08002718d305 | mysql3      |        3306 | ONLINE       | SECONDARY   | 8.0.17         |
| group_replication_applier | c88e9182-cf5a-11e9-8d12-08002718d305 | mysql1      |        3306 | ONLINE       | SECONDARY   | 8.0.17         |
| group_replication_applier | cf720f38-cf5a-11e9-8b70-08002718d305 | mysql2      |        3306 | ONLINE       | PRIMARY     | 8.0.17         |
+---------------------------+--------------------------------------+-------------+-------------+--------------+-------------+----------------+
3 rows in set (0.0312 sec)
```

Monitoring MySQL InnoDB Cluster - Performance_Schema (2)

```
MySQL 8.0.17  mysql3:33060+  performance_schema  2019-09-10 12:44:13
SQL > select * from replication_group_member_stats\G
*************************** 1. row ***************************
                            CHANNEL_NAME: group_replication_applier
                                 VIEW_ID: 15680483292627132:56
                               MEMBER_ID: b016dfa7-d31b-11e9-8e21-08002718d305
             COUNT_TRANSACTIONS_IN_QUEUE: 0
              COUNT_TRANSACTIONS_CHECKED: 1205
                COUNT_CONFLICTS_DETECTED: 0
       COUNT_TRANSACTIONS_ROWS_VALIDATING: 1
       TRANSACTIONS_COMMITTED_ALL_MEMBERS: 1610e773-d323-11e9-a38b-08002718d305:1-11123:1000032-1000165,
26c11a7e-cf5b-11e9-a38b-08002718d305:1-1588:1001542-1001548
           LAST_CONFLICT_FREE_TRANSACTION: 1610e773-d323-11e9-a38b-08002718d305:11123
COUNT_TRANSACTIONS_REMOTE_IN_APPLIER_QUEUE: 0
         COUNT_TRANSACTIONS_REMOTE_APPLIED: 1207
         COUNT_TRANSACTIONS_LOCAL_PROPOSED: 0
         COUNT_TRANSACTIONS_LOCAL_ROLLBACK: 0
*************************** 2. row ***************************
                            CHANNEL_NAME: group_replication_applier
                                 VIEW_ID: 15680483292627132:56
                               MEMBER_ID: c88c9182-cf5a-11e9-8d12-08002718d305
```

Monitoring MySQL InnoDB Cluster - Performance_Schema (3)

```
SQL > select * from replication_connection_status\G
*************************** 1. row ***************************
                             CHANNEL_NAME: group_replication_applier
                               GROUP_NAME: 1610e773-d323-11e9-a38b-08002718d305
                              SOURCE_UUID: 1610e773-d323-11e9-a38b-08002718d305
                                THREAD_ID: NULL
                           SERVICE_STATE: ON
                       COUNT_RECEIVED_HEARTBEATS: 0
                        LAST_HEARTBEAT_TIMESTAMP: 0000-00-00 00:00:00
                        RECEIVED_TRANSACTION_SET: 1610e773-d323-11e9-a38b-08002718d305:1-11123:1000032-1000165,
26c11a7e-cf5b-11e9-a38b-08002718d305:1-1588:1001542-1001548
                                LAST_ERROR_NUMBER: 0
                               LAST_ERROR_MESSAGE:
                             LAST_ERROR_TIMESTAMP: 0000-00-00 00:00:00
                           LAST_QUEUED_TRANSACTION: 1610e773-d323-11e9-a38b-08002718d305:11123
 LAST_QUEUED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP: 2019-09-10 12:13:47.051566
LAST_QUEUED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP: 0000-00-00 00:00:00
     LAST_QUEUED_TRANSACTION_START_QUEUE_TIMESTAMP: 2019-09-10 12:13:27.003558
       LAST_QUEUED_TRANSACTION_END_QUEUE_TIMESTAMP: 2019-09-10 12:13:27.003578
                             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
```

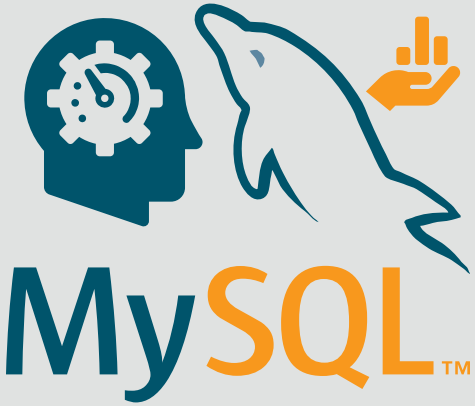
Monitoring MySQL InnoDB Cluster - Performance_Schema (4)

```
*************************** 2. row ***************************
                                CHANNEL_NAME: group_replication_recovery
                                  GROUP_NAME:
                                   SOURCE_UUID:
                                     THREAD_ID: NULL
                                SERVICE_STATE: OFF
                    COUNT_RECEIVED_HEARTBEATS: 0
                     LAST_HEARTBEAT_TIMESTAMP: 0000-00-00 00:00:00
                     RECEIVED_TRANSACTION_SET:
                            LAST_ERROR_NUMBER: 0
                           LAST_ERROR_MESSAGE:
                         LAST_ERROR_TIMESTAMP: 0000-00-00 00:00:00
                      LAST_QUEUED_TRANSACTION:
 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: 0000-00-00 00:00:00
       LAST_QUEUED_TRANSACTION_END_QUEUE_TIMESTAMP: 0000-00-00 00:00:00
                      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
```

As a DBA, I would like to know which of my Secondary-Master is slower ?

As a DBA, I would like to know which of my Secondary-Master is slower ?

`Performance_Schema` provides to us all what we need to discover which Secondary-Slave takes more time to "fully replicate" events.



Let's run this **bash** code on all our Secondary-Masters (**mysql1** and **mysql2**):

```
for i in $(seq 1 10)
do
  if [[ $i -eq 1 ]]; then echo "repl delay      transport time  time to RL      apply time"
fi
mysql -BN -e "
SELECT LAST_APPLIED_TRANSACTION_END_APPLY_TIMESTAMP - LAST_APPLIED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP
  'rep delay (sec)',
LAST_QUEUED_TRANSACTION_START_QUEUE_TIMESTAMP - LAST_QUEUED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP
   'transport time',
LAST_QUEUED_TRANSACTION_END_QUEUE_TIMESTAMP - LAST_QUEUED_TRANSACTION_START_QUEUE_TIMESTAMP 'time RL',
LAST_APPLIED_TRANSACTION_END_APPLY_TIMESTAMP - LAST_APPLIED_TRANSACTION_START_APPLY_TIMESTAMP 'apply time'
FROM performance_schema.replication_applier_status_by_worker t1
JOIN performance_schema.replication_connection_status t2
ON t2.channel_name=t1.channel_name
WHERE t1.channel_name='group_replication_applier'";
sleep 2
done
```

Let's run this **bash** code on all our Secondary-Masters (**mysql1** and **mysql2**):

```
for i in $(seq 1 10)
do
  if [[ $i -eq 1 ]]; then echo "repl delay      transport time  time to RL      apply time"
fi
mysql -BN -e "
SELECT LAST_APPLIED_TRANSACTION_END_APPLY_TIMESTAMP - LAST_APPLIED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP
  'rep delay (sec)',
LAST_QUEUED_TRANSACTION_START_QUEUE_TIMESTAMP - LAST_QUEUED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP
   'transport time',
LAST_QUEUED_TRANSACTION_END_QUEUE_TIMESTAMP - LAST_QUEUED_TRANSACTION_START_QUEUE_TIMESTAMP 'time RL',
LAST_APPLIED_TRANSACTION_END_APPLY_TIMESTAMP - LAST_APPLIED_TRANSACTION_START_APPLY_TIMESTAMP 'apply time'
FROM performance_schema.replication_applier_status_by_worker t1
JOIN performance_schema.replication_connection_status t2
ON t2.channel_name=t1.channel_name
WHERE t1.channel_name='group_replication_applier'";
sleep 2
done
```

This script will query some **performance_schema** tables 10 times every 2 seconds (we write on our cluster every 2 secs), and all members have their time in sync: **ntpd**.

Monitoring MySQL InnoDB Cluster - Performance_Schema (5)

repl delay	transport time	time to RL	apply time
0.027762	0.025275	0.000015	0.002452
0.026801	0.025262	0.000015	0.001499
0.028265	0.025791	0.000014	0.002442
0.028620	0.027163	0.000014	0.001425
0.030794	0.028434	0.000021	0.002316
0.031147	0.029264	0.000015	0.001849
0.031738	0.030298	0.000013	0.001408
0.034962	0.031270	0.000014	0.003659
0.034176	0.032283	0.000014	0.001859
0.041329	0.034135	0.000023	0.007144

[root@mysql1 ~]#

repl delay	transport time	time to RL	apply time
0.005319	0.002559	0.000014	0.002726
0.003053	0.001704	0.000015	0.001314
0.002600	0.001165	0.000013	0.001404
0.003315	0.001557	0.000013	0.001727
0.005705	0.002726	0.000018	0.002937
0.005432	0.001925	0.000013	0.003475
0.003407	0.001915	0.000032	0.001441
0.003228	0.001504	0.000014	0.001692
0.006125	0.002070	0.000014	0.004022
0.008609	0.001600	0.000014	0.006967

[root@mysql2 ~]#

Monitoring MySQL InnoDB Cluster - Performance_Schema (5)

repl delay	transport time	time to RL	apply time
0.027762	0.025275	0.000015	0.002452
0.026801	0.025262	0.000015	0.001499
0.028265	0.025791	0.000014	0.002442
0.028620	0.027163	0.000014	0.001425
0.030794	0.028434	0.000021	0.002316
0.031147	0.029264	0.000015	0.001849
0.031738	0.030298	0.000013	0.001408
0.034962	0.031270	0.000014	0.003659
0.034176	0.032283	0.000014	0.001859
0.041329	0.034135	0.000023	0.007144

[root@mysql1 ~]#

repl delay	transport time	time to RL	apply time
0.005319	0.002559	0.000014	0.002726
0.003053	0.001704	0.000015	0.001314
0.002600	0.001165	0.000013	0.001404
0.003315	0.001557	0.000013	0.001727
0.005705	0.002726	0.000018	0.002937
0.005432	0.001925	0.000013	0.003475
0.003407	0.001915	0.000032	0.001441
0.003228	0.001504	0.000014	0.001692
0.006125	0.002070	0.000014	0.004022
0.008609	0.001600	0.000014	0.006967

[root@mysql2 ~]#

As a DBA, I would like to estimate how many transactions per second could my cluster process with the current configuration.

As a DBA, I would like to estimate how many transactions per second could my cluster process with the current configuration.

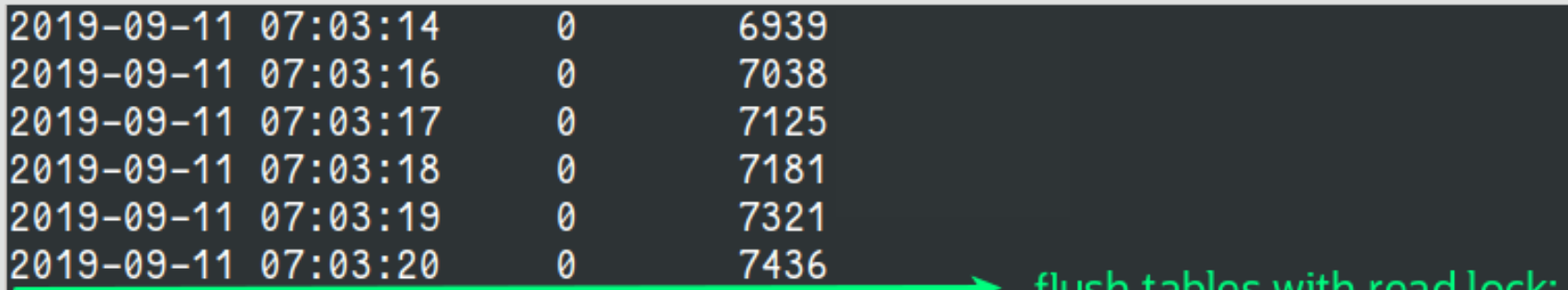
To perform this **estimation**, we will block all writes to a member of our cluster. As a result, the apply queue will grow on that server. When the queue is large enough, we will enable writes again on the node and see how many transactions and how much time the node needed to process everything and be back in sync with the cluster.

Currently, the application is performing 100 tx/sec.

On the node that we will lock, let's run this small bash command:

```
while true
do
  mysql -BNe "select now(), transactions_behind, count_transactions_remote_applied
         from sys.gr_member_routing_candidate_status, performance_schema.replication_group_member_stats
         where member_id=@@server_uuid;"
  sleep 1
done
```

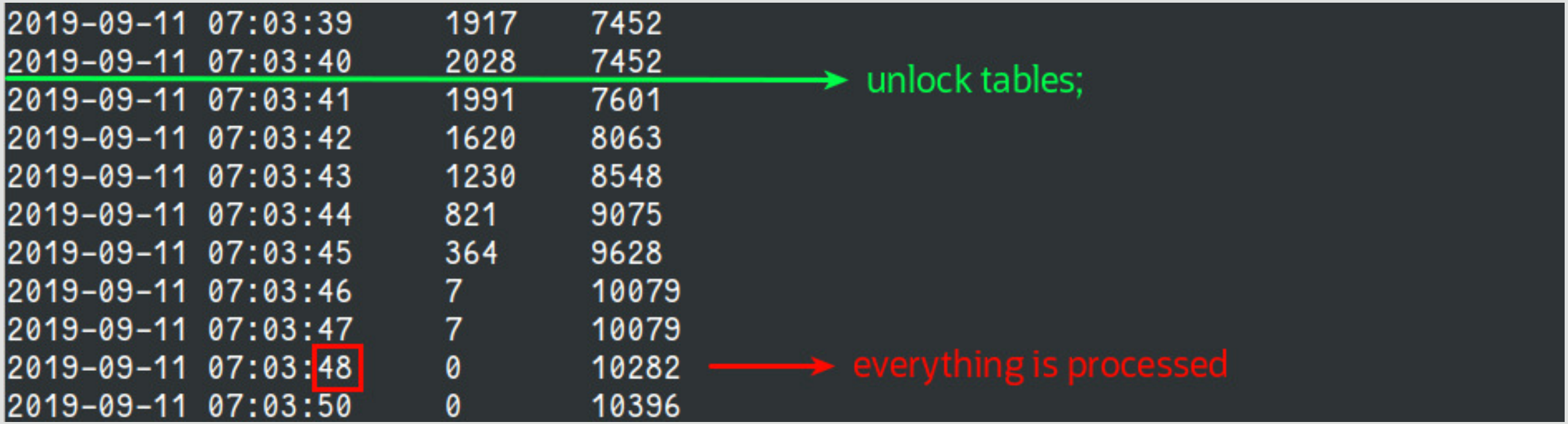
```
2019-09-11 07:03:14     0       6939
2019-09-11 07:03:16     0       7038
2019-09-11 07:03:17     0       7125
2019-09-11 07:03:18     0       7181
2019-09-11 07:03:19     0       7321
2019-09-11 07:03:20     0       7436
2019-09-11 07:03:21     99      7452
2019-09-11 07:03:22     240     7452
2019-09-11 07:03:23     348     7452
2019-09-11 07:03:24     456     7452
2019-09-11 07:03:25     554     7452
```

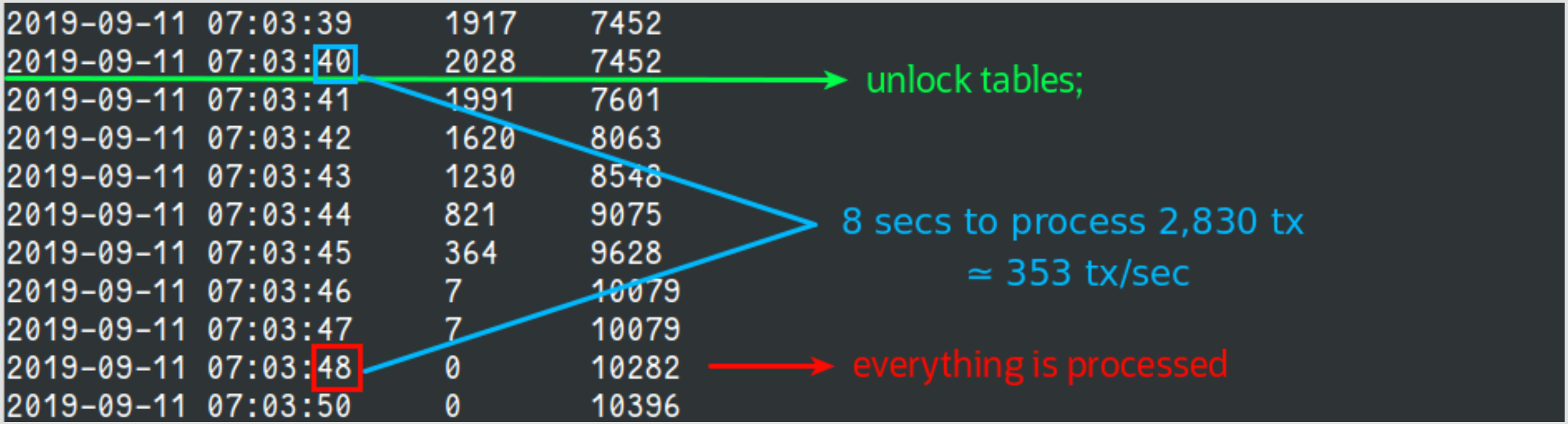


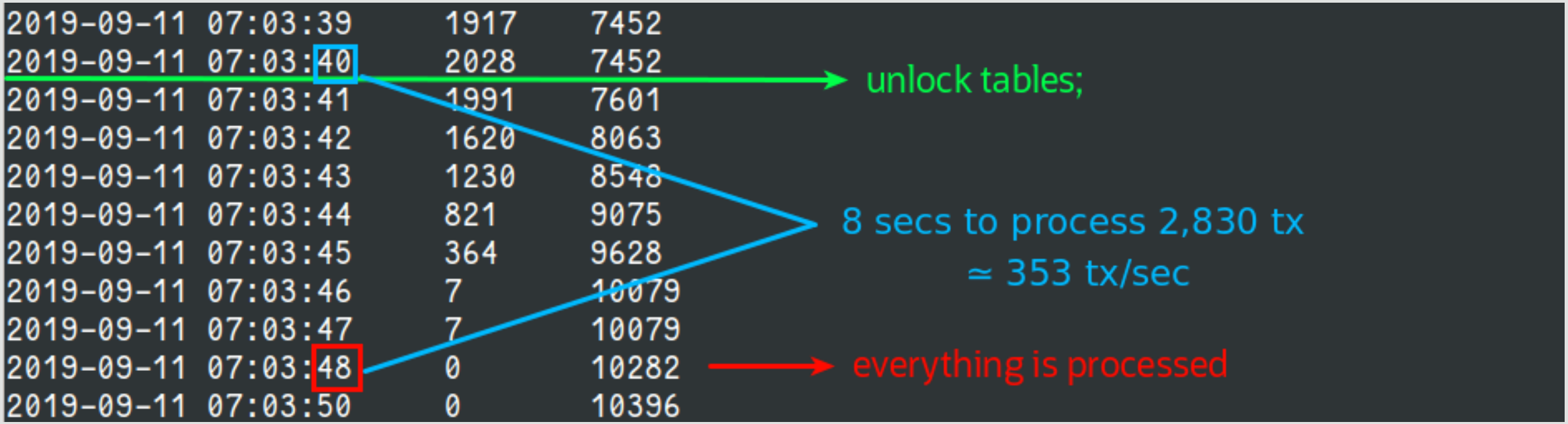
On the node that we will lock, let's run this small bash command:

```
while true
do
  mysql -BNe "select now(), transactions_behind, count_transactions_remote_applied
         from sys.gr_member_routing_candidate_status, performance_schema.replication_group_member_stats
         where member_id=@@server_uuid;"
  sleep 1
done
```

```
2019-09-11 07:03:14      0        6939   } 99 tx
2019-09-11 07:03:16      0        7038   } 87 tx
2019-09-11 07:03:17      0        7125   } 56 tx
2019-09-11 07:03:18      0        7181   } 140 tx
2019-09-11 07:03:19      0        7321
2019-09-11 07:03:20      0        7436
2019-09-11 07:03:21      99       7452   <-- flush tables with read lock;
2019-09-11 07:03:22      240      7452
2019-09-11 07:03:23      348      7452
2019-09-11 07:03:24      456      7452
2019-09-11 07:03:25      554      7452
```







So we can estimate that our MySQL InnoDB Cluster is able to process approximately 350 tx/sec.

So what will happen if we increase our workload to 360 tx/sec ?

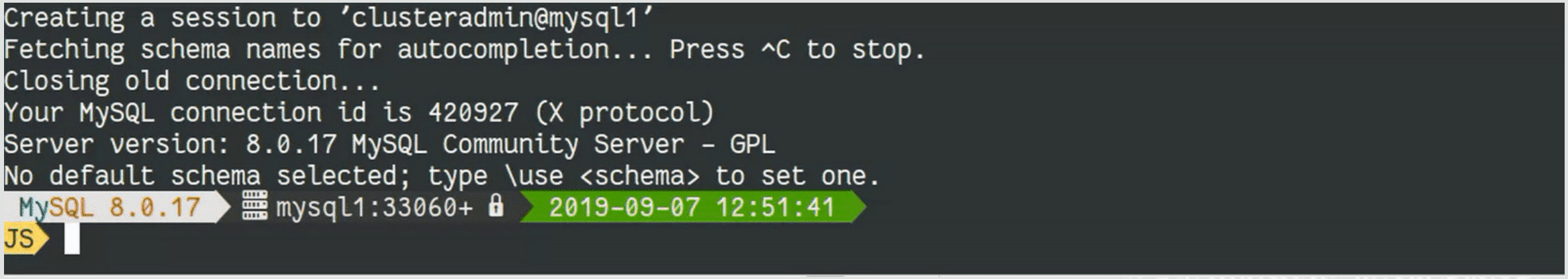
```
2019-09-11 07:08:28      0        10605
2019-09-11 07:08:29      0        10605
2019-09-11 07:08:30      0        10605
2019-09-11 07:08:31      0        10605
2019-09-11 07:08:32      3        10796
2019-09-11 07:08:33      0        11061
2019-09-11 07:08:34      1        11335
2019-09-11 07:08:35      4        11602
2019-09-11 07:08:36      2        11867
2019-09-11 07:08:37      2        12127
2019-09-11 07:08:38      1        12397
2019-09-11 07:08:39      1        12671
2019-09-11 07:08:40      1        12939
2019-09-11 07:08:41      1        13197
2019-09-11 07:08:42      4        13461
2019-09-11 07:08:43      0        13729
2019-09-11 07:08:44      0        13993
2019-09-11 07:08:45      1        14268
2019-09-11 07:08:46      3        14528
2019-09-11 07:08:47      10       14802
2019-09-11 07:08:48      7        15085
2019-09-11 07:08:49      5        15300
2019-09-11 07:08:50      68       15427
2019-09-11 07:08:51      17       15688
```

2019-09-11 07:08:28	0	10605
2019-09-11 07:08:29	0	10605
2019-09-11 07:08:30	0	10605
2019-09-11 07:08:31	0	10605
2019-09-11 07:08:32	3	10796
2019-09-11 07:08:33	0	11061
2019-09-11 07:08:34	1	11335
2019-09-11 07:08:35	4	11602
2019-09-11 07:08:36	2	11867
2019-09-11 07:08:37	2	12127
2019-09-11 07:08:38	1	12397
2019-09-11 07:08:39	1	12671
2019-09-11 07:08:40	1	12939
2019-09-11 07:08:41	1	13197
2019-09-11 07:08:42	4	13461
2019-09-11 07:08:43	0	13729
2019-09-11 07:08:44	0	13993
2019-09-11 07:08:45	1	14268
2019-09-11 07:08:46	3	14528
2019-09-11 07:08:47	10	14802
2019-09-11 07:08:48	7	15085
2019-09-11 07:08:49	5	15300
2019-09-11 07:08:50	68	15427
2019-09-11 07:08:51	17	15688

The cluster will start slowing down and this might have a major impact depending of the *consistency level*.

Monitoring MySQL InnoDB Cluster - MySQL Shell UDR

It's also possible to constantly monitor the status of your MySQL InnoDB Cluster using **U**ser-**D**efined **R**eports in MySQL Shell:



More Info: https://lefred.be/content/using-the-new-mysql-shell-reporting-framework-to-monitor-innodb-cluster/

Monitoring MySQL Router

Since MySQL 8.0.17 it's possible to query the MySQL Router using its **REST API** !

You need to enable it in the Router's config file:

```
[http_server]
port=8080

[rest_api]

[rest_router]
require_realm=somerealm

[rest_routing]
require_realm=somerealm
```

```
[rest_metadata_cache]
require_realm=somerealm

[http_auth_realm:somerealm]
backend=somebackend
method=basic
name=Some Realm

[http_auth_backend:somebackend]
backend=file
filename=/etc/mysqlrouter/mysqlrouter.pwd
```

Monitoring MySQL Router (2)

You can then query the REST API using `curl`:

```
$ curl -s -u fred:fred \
  http://192.168.91.2:8080/api/20190715/routes/myCluster_default_rw/destinations
{"items":[{"address":"mysql2","port":3306}]}
```

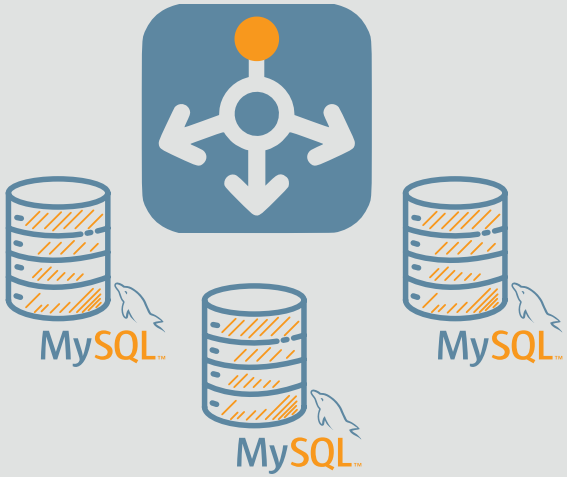
Or use/create a plugin for the MySQL Shell.

More Info:

- https://lefred.be/content/mysqlrouter-8-0-17-and-the-rest-api/
- https://lefred.be/content/mysql-router-8-0-17s-rest-api-mysql-shell-extensions/

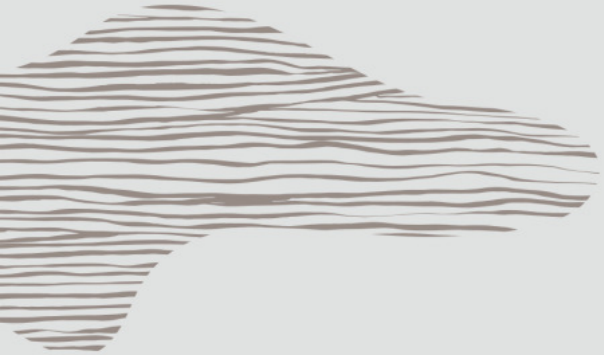
Monitoring MySQL Router (3)

```
MySQL  2019-07-15 23:18:16
JS myrouter.status()
+-------------------------+
| Cluster name: myCluster |
+-------------------------+
     Refresh Succeeded: 193
        Refresh Failed: 0
 Last Refresh Hostname: mysql1:3306
   +--------+
   | routes |
   +--------+
    * myCluster_default_ro (alive) :
        Total Connections: 2    Active Connections: 2   Blocked Hosts: 0
        ---> mysql2 : 3306
        ---> mysql3 : 3306
    * myCluster_default_rw (alive) :
        Total Connections: 1    Active Connections: 1   Blocked Hosts: 0
        ---> mysql1 : 3306
    * myCluster_default_x_ro (alive) :
        Total Connections: 0    Active Connections: 0   Blocked Hosts: 0
        ---> mysql2 : 33060
        ---> mysql3 : 33060
    * myCluster_default_x_rw (alive) :
        Total Connections: 0    Active Connections: 0   Blocked Hosts: 0
        ---> mysql1 : 33060
```



Monitoring MySQL Router (4)

```
MySQL  2019-07-15 23:38:15
JS myrouter.connections()
+-----------------------+--------------------+-------------+----------------------+----------------------+-----------------------------+
| Route                 | Source             | Destination |          From Server |            To Server | Connection Started          |
+-----------------------+--------------------+-------------+----------------------+----------------------+-----------------------------+
| myCluster_default_ro  | 192.168.91.1:45452 | mysql3:3306 |               275 kb |                3 kb | 2019-07-15T21:19:23.365529Z |
|                       | 192.168.91.1:45450 | mysql2:3306 |                58 kb |                1 kb | 2019-07-15T21:19:00.254364Z |
+-----------------------+--------------------+-------------+----------------------+----------------------+-----------------------------+
| myCluster_default_rw  | 192.168.91.1:34970 | mysql1:3306 |               179 kb |                2 kb | 2019-07-15T21:18:34.386295Z |
+-----------------------+--------------------+-------------+----------------------+----------------------+-----------------------------+
| myCluster_default_x_ro |                   |             |                      |                      |                             |
+-----------------------+--------------------+-------------+----------------------+----------------------+-----------------------------+
| myCluster_default_x_rw |                   |             |                      |                      |                             |
+-----------------------+--------------------+-------------+----------------------+----------------------+-----------------------------+
MySQL  2019-07-15 23:38:18
JS myrouter.connections('_rw')
+-----------------------+--------------------+-------------+----------------------+----------------------+-----------------------------+
| Route                 | Source             | Destination |          From Server |            To Server | Connection Started          |
+-----------------------+--------------------+-------------+----------------------+----------------------+-----------------------------+
| myCluster_default_rw  | 192.168.91.1:34970 | mysql1:3306 |               179 kb |                2 kb | 2019-07-15T21:18:34.386295Z |
+-----------------------+--------------------+-------------+----------------------+----------------------+-----------------------------+
| myCluster_default_x_rw |                   |             |                      |                      |                             |
+-----------------------+--------------------+-------------+----------------------+----------------------+-----------------------------+
MySQL  2019-07-15 23:38:21
JS myrouter.connections('default_rw')
+-----------------------+--------------------+-------------+----------------------+----------------------+-----------------------------+
| Route                 | Source             | Destination |          From Server |            To Server | Connection Started          |
+-----------------------+--------------------+-------------+----------------------+----------------------+-----------------------------+
| myCluster_default_rw  | 192.168.91.1:34970 | mysql1:3306 |               179 kb |                2 kb | 2019-07-15T21:18:34.386295Z |
+-----------------------+--------------------+-------------+----------------------+----------------------+-----------------------------+
```



MySQL InnoDB Cluster

Network Monitoring

Network monitoring - the basics

```
JS cluster.status()
{
    "clusterName": "myCluster",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "mysql1:3306",
        "ssl": "REQUIRED",
        "status": "OK",
        "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
        "topology": {
            "mysql1:3306": {
                "address": "mysql1:3306",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.17"
            },
            "mysql2:3306": {
                "address": "mysql2:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.17"
            },
            "mysql3:3306": {
                "address": "mysql3:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.17"
            }
        },
        "topologyMode": "Single-Primary"
    },
    "groupInformationSourceMember": "mysql1:3306"
}
```

All good ;)

Network monitoring - the basics

```
JS cluster.status()
{
    "clusterName": "myCluster",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "mysql1:3306",
        "ssl": "REQUIRED",
        "status": "OK_NO_TOLERANCE",
        "statusText": "Cluster is NOT tolerant to any failures. 1 member is not active",
        "topology": {
            "mysql1:3306": {
                "address": "mysql1:3306",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.17"
            },
            "mysql2:3306": {
                "address": "mysql2:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.17"
            },
            "mysql3:3306": {
                "address": "mysql3:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "(MISSING)"
            }
        },
        "topologyMode": "Single-Primary"
    },
    "groupInformationSourceMember": "mysql1:3306"
}
```

When members crash or are expelled

Network monitoring - the basics

```
JS cluster.status()
{
    "clusterName": "myCluster",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "mysql3:3306",
        "ssl": "REQUIRED",
        "status": 'NO_QUORUM',
        "statusText": "Cluster has no quorum as visible from 'mysql1:3306' and cannot process write transactions.
2 members are not active",
        "topology": {
            "mysql1:3306": {
                "address": "mysql1:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.17"
            },
            "mysql2:3306": {
                "address": "mysql2:3306",
                "mode": "n/a",
                "readReplicas": {},
                "role": "HA",
                "shellConnectError": "MySQL Error 2003 (HY000): Can't connect to MySQL server on 'mysql2' (110)",
                "status": "UNREACHABLE",
                "version": "8.0.17"
            },
            "mysql3:3306": {
                "address": "mysql3:3306",
                "mode": "n/a",
                "readReplicas": {},
                "role": "HA",
                "shellConnectError": "MySQL Error 2003 (HY000): Can't connect to MySQL server on 'mysql3' (110)",
                "status": "UNREACHABLE",
                "version": "8.0.17"
```

When a majority of the members becomes unreachable

Network monitoring - Adapt to faulty networks

To avoid frequent member expels due to faulty networks make use of:

`group_replication_member_expel_timeout`

```
MySQL 8.0.17  mysql1:33060+  2019-09-12 09:39:35
JS cluster.setOption('expelTimeout', 60)
Setting the value of 'expelTimeout' to '60' in all ReplicaSet members ...

Successfully set the value of 'expelTimeout' to '60' in the 'default' ReplicaSet.
```

Network monitoring - Deal with failure scenarios

Configure your rejoin attempts for each member (this method persists the change)

```
MySQL 8.0.17  mysql1:33060+  2019-09-12 09:50:36
JS cluster.setInstanceOption("mysql1:3306", "autoRejoinTries", 3)
WARNING: The member will only proceed according to its exitStateAction if auto-rejoin fails (i.e. all retry
attempts are exhausted).

Setting the value of 'autoRejoinTries' to '3' in the instance: 'mysql1:3306' ...

Successfully set the value of 'autoRejoinTries' to '3' in the 'default' ReplicaSet member: 'mysql1:3306'.
```

Network monitoring - Deal with failure scenarios

Configure your rejoin attempts for each member (this method persists the change)

```
MySQL 8.0.17  mysql1:33060+  2019-09-12 09:50:36
JS cluster.setInstanceOption("mysql1:3306", "autoRejoinTries", 3)
WARNING: The member will only proceed according to its exitStateAction if auto-rejoin fails (i.e. all retry
attempts are exhausted).

Setting the value of 'autoRejoinTries' to '3' in the instance: 'mysql1:3306' ...

Successfully set the value of 'autoRejoinTries' to '3' in the 'default' ReplicaSet member: 'mysql1:3306'.
```

If you prefer to not persist it:

```
MySQL 8.0.17  mysql1:33060+  2019-09-12 09:50:58
JS \sql SET @@GLOBAL.group_replication_autorejoin_tries = 5
Query OK, 0 rows affected (0.0007 sec)
```

Network monitoring - Deal with failure scenarios

Configure your timeout during loss of quorum

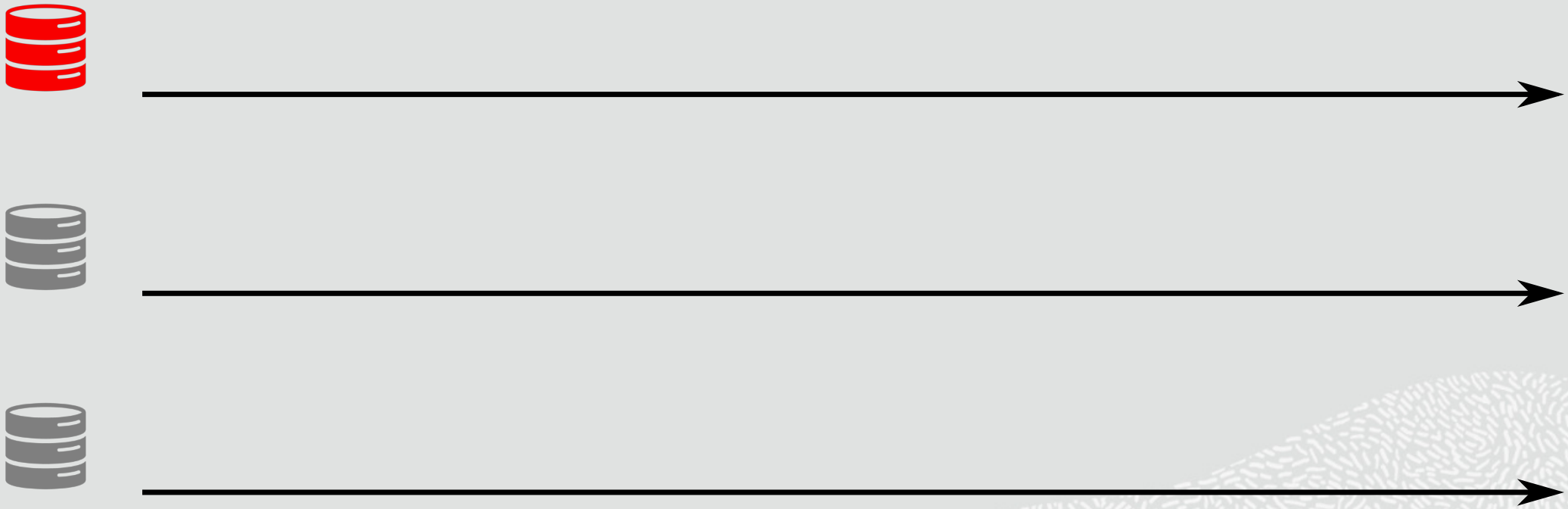
```
MySQL 8.0.17  mysql1:33060+  2019-09-12 09:43:12
JS \sql SET @@GLOBAL.group_replication_unreachable_majority_timeout = 60
Query OK, 0 rows affected (0.0033 sec)
MySQL 8.0.17  mysql1:33060+  2019-09-12 09:43:47
JS \sql SET PERSIST group_replication_unreachable_majority_timeout = 60
Query OK, 0 rows affected (0.0095 sec)
```

Network monitoring - Deal with failure scenarios

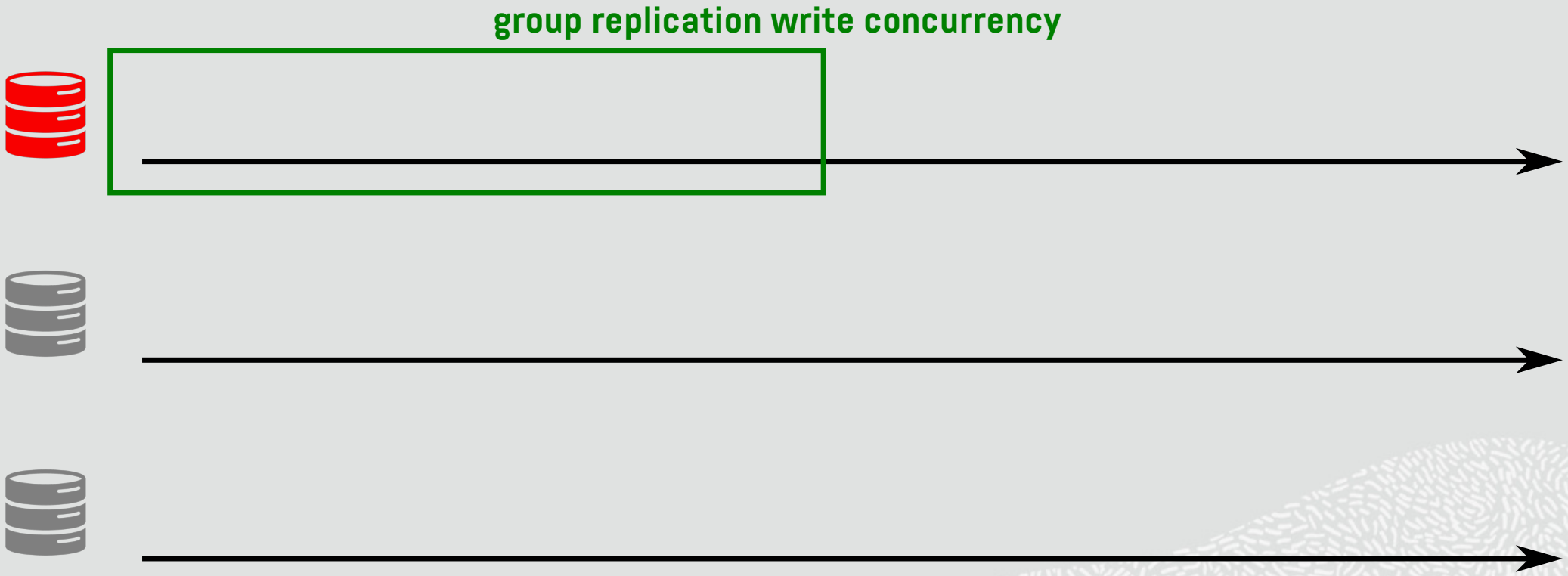
Be prepared: whom shall be the primary when failure strikes

```
 MySQL 8.0.17  mysql1:33060+  2019-09-12 09:44:30
JS cluster.setInstanceOption("mysql1:3306", "memberWeight", 75)
Setting the value of 'memberWeight' to '75' in the instance: 'mysql1:3306' ...

Successfully set the value of 'memberWeight' to '75' in the 'default' ReplicaSet member: 'mysql1:3306'.
```

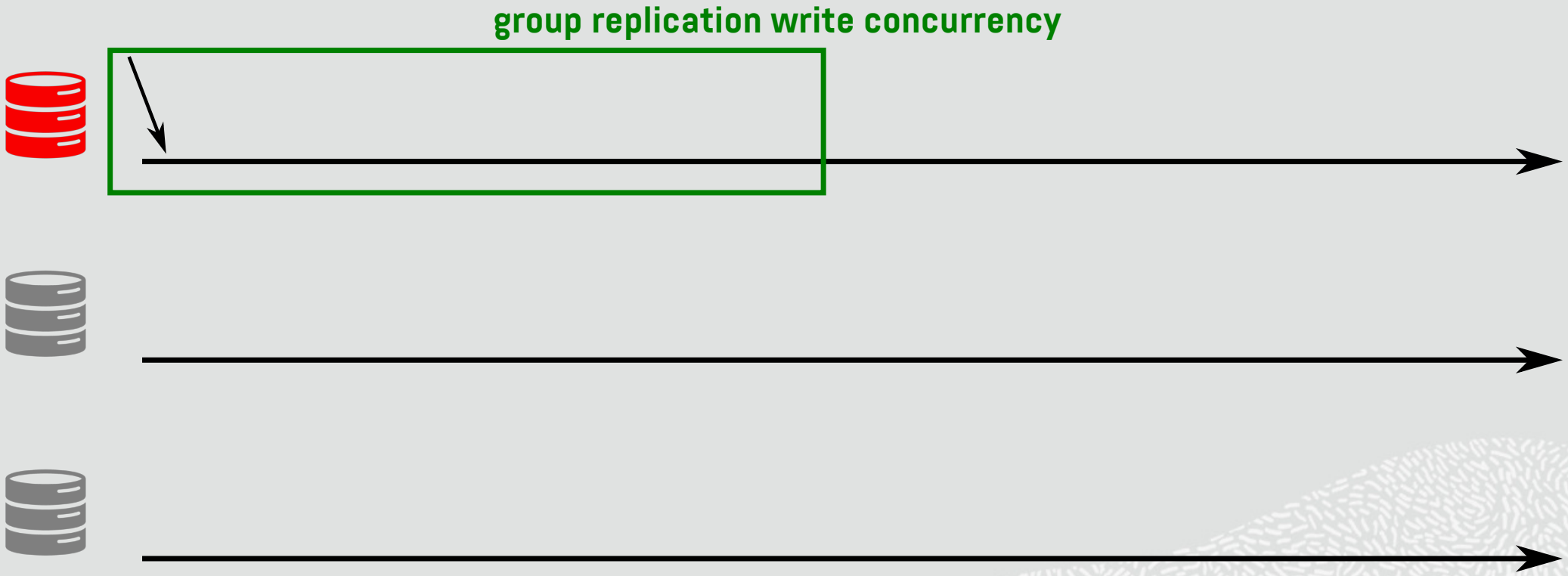
Network monitoring - Adapt to higher latencies



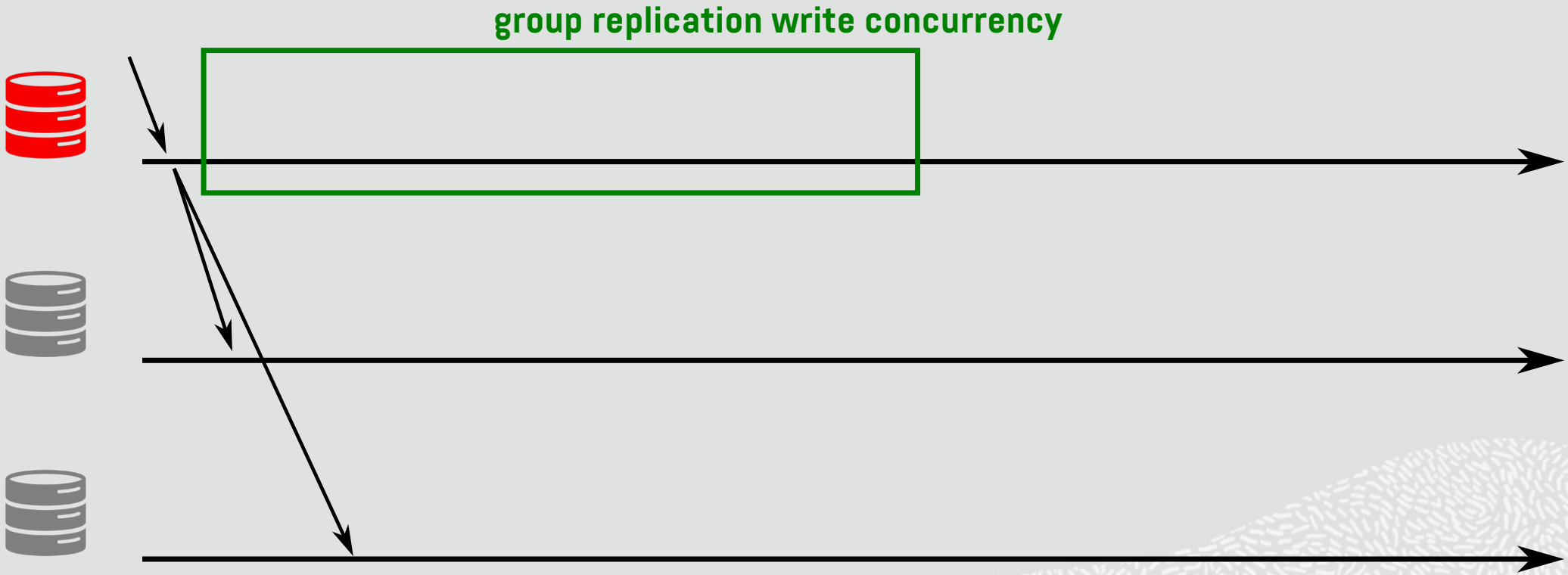
Network monitoring - Adapt to higher latencies



Network monitoring - Adapt to higher latencies

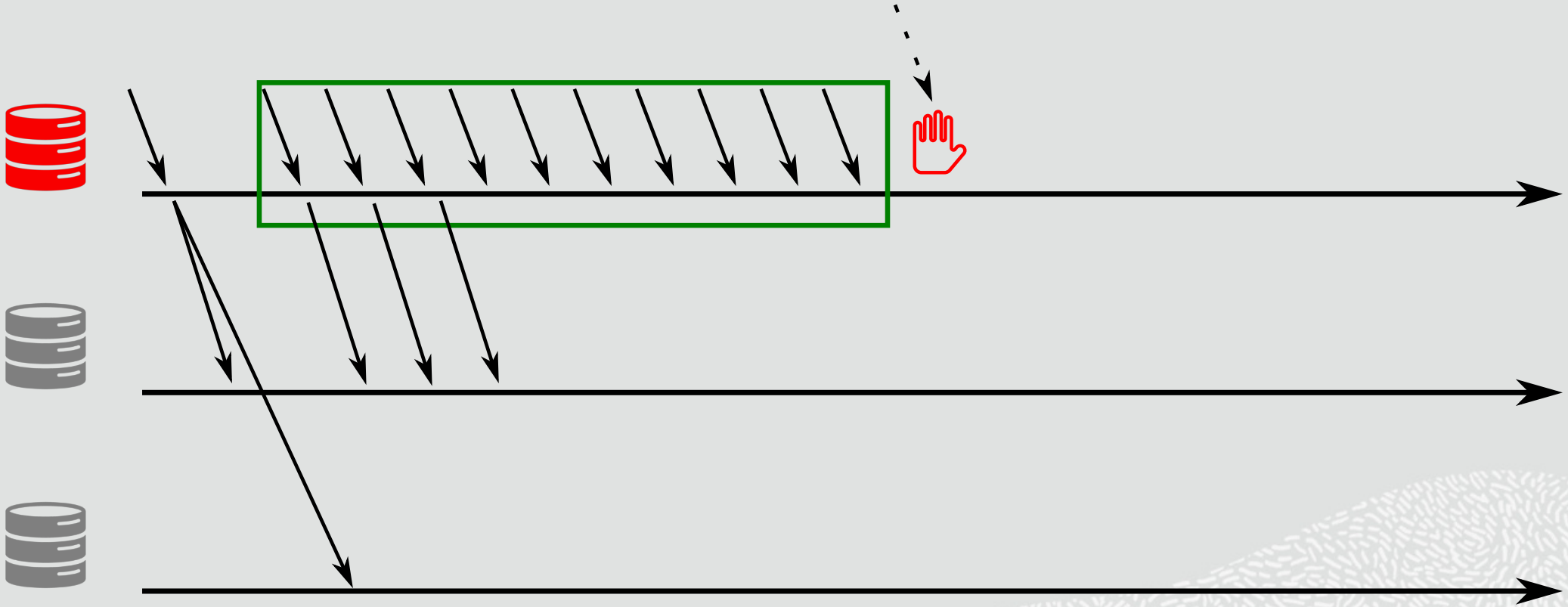


Network monitoring - Adapt to higher latencies

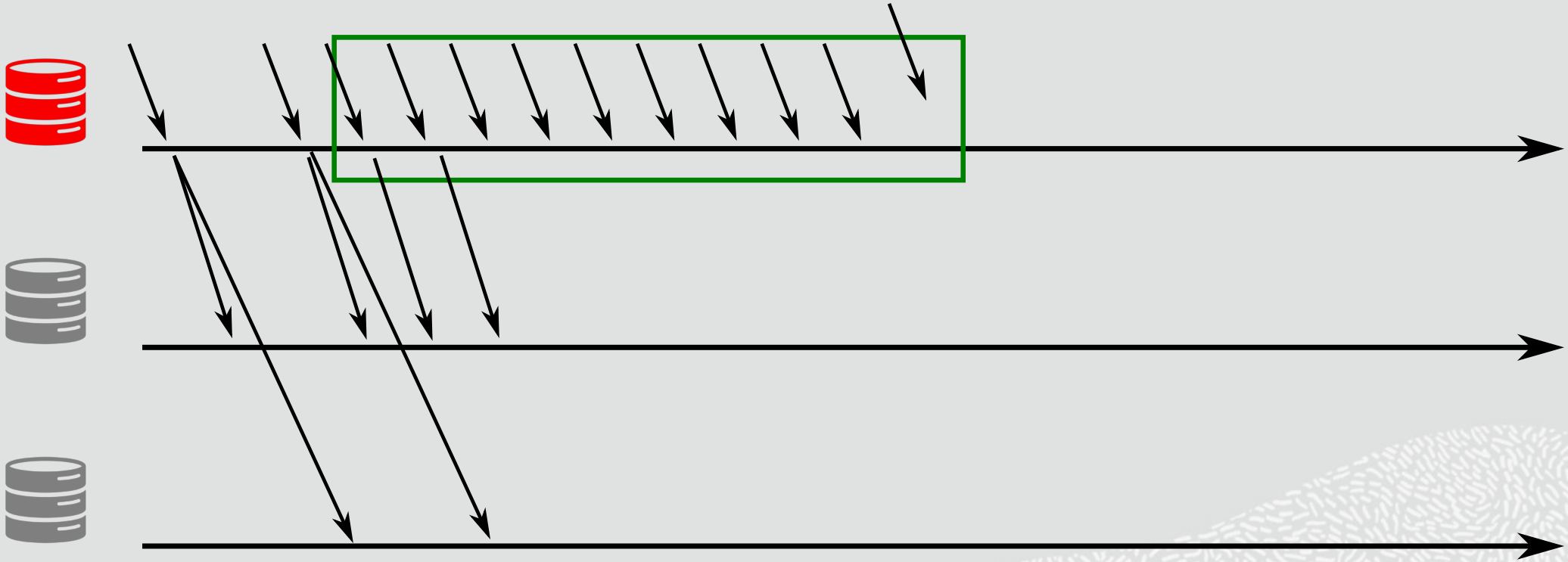


Network monitoring - Adapt to higher latencies

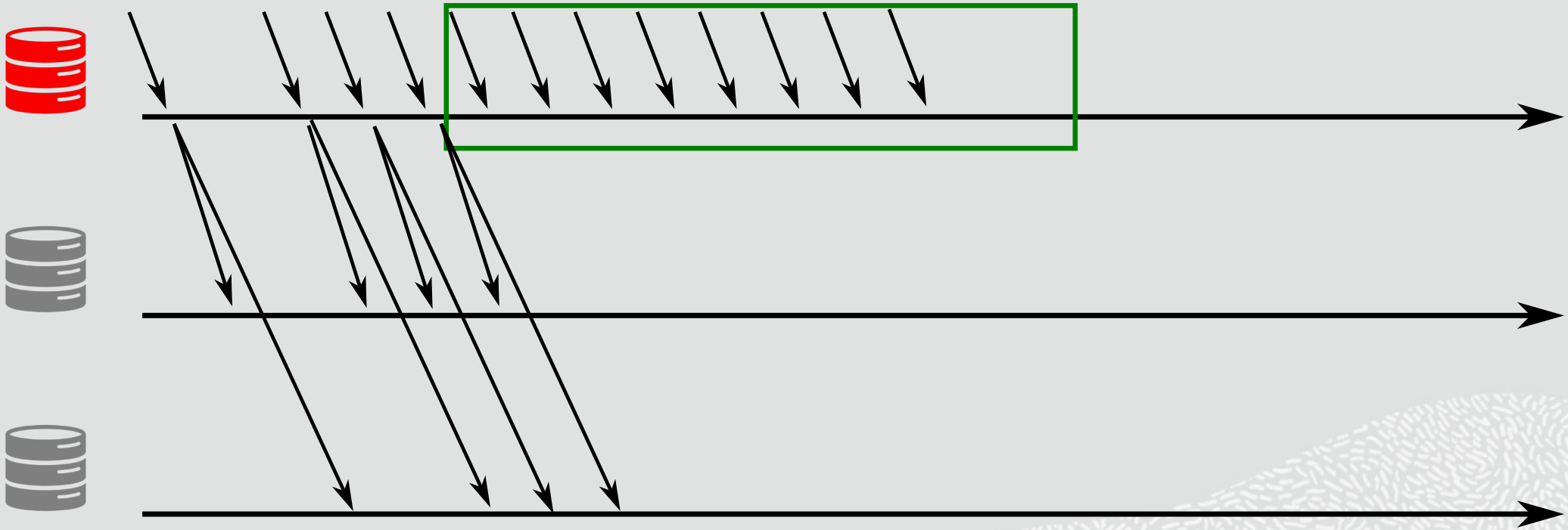
Network monitoring - Adapt to higher latencies



Network monitoring - Adapt to higher latencies



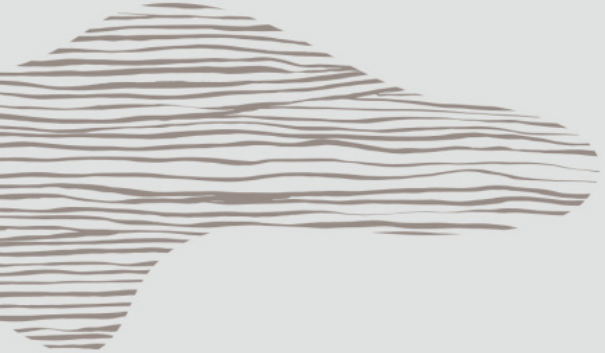
Network monitoring - Adapt to higher latencies



Network monitoring - Adapt to higher latencies

```
 MySQL 8.0.17  mysql1:33060+  2019-09-12 10:00:46
JS \sql SELECT group_replication_get_write_concurrency()
+-----------------------------------------+
| group_replication_get_write_concurrency() |
+-----------------------------------------+
| 10                                      |
+-----------------------------------------+
1 row in set (0.0236 sec)
 MySQL 8.0.17  mysql1:33060+  2019-09-12 10:00:49
JS \sql SELECT group_replication_set_write_concurrency(20)
+-----------------------------------------------------------------------------------+
| group_replication_set_write_concurrency(20)                                       |
+-----------------------------------------------------------------------------------+
| UDF is asynchronous, check log or call group_replication_get_write_concurrency(). |
+-----------------------------------------------------------------------------------+
1 row in set (0.0008 sec)
```

Configure your write concurrency in the group



MySQL InnoDB Cluster

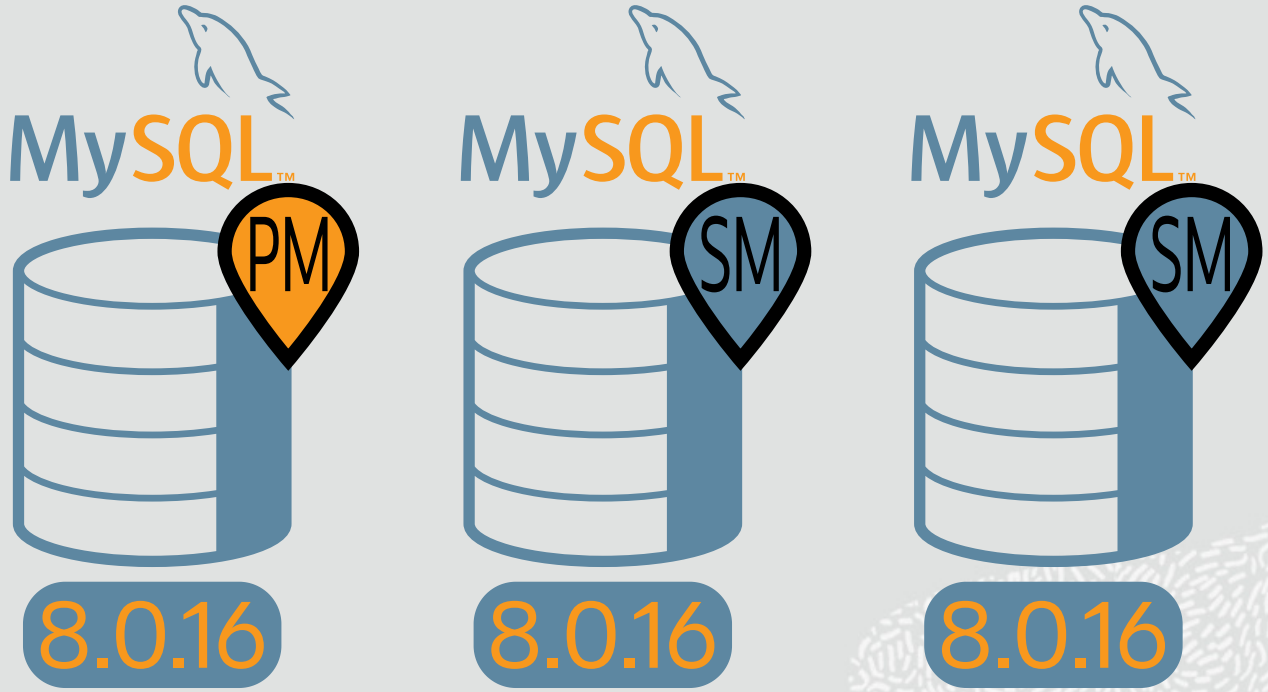
Operations

Upgrading to a newer version

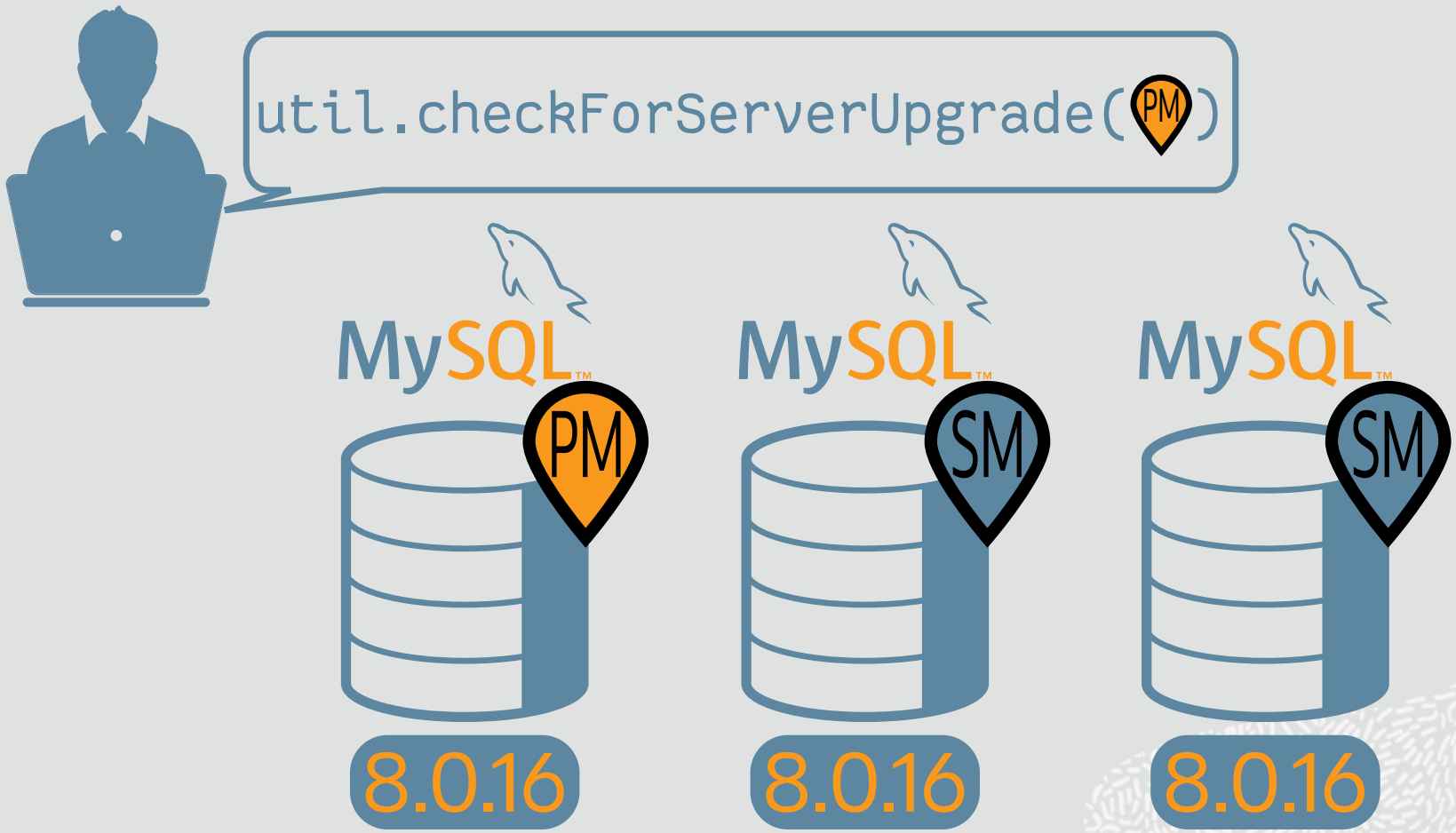
Before upgrading, don't forget to use `util.upgradeChecker()` in MySQL Shell. This operation must be performed on the **Primary Master**:

```
 MySQL  2019-09-07 14:06:04
JS util.checkForServerUpgrade('clusteradmin@mysql1')
Please provide the password for 'clusteradmin@mysql1': ****
Save password for 'clusteradmin@mysql1'? [Y]es/[N]o/Ne[v]er (default No): y
The MySQL server at mysql1, version 8.0.17 - MySQL Community Server - GPL, will
now be checked for compatibility issues for upgrade to MySQL 8.0.18...

1) Issues reported by 'check table x for upgrade' command
  No issues found

Errors:   0
Warnings: 0
Notices:  0

No known compatibility errors or issues were found.
```

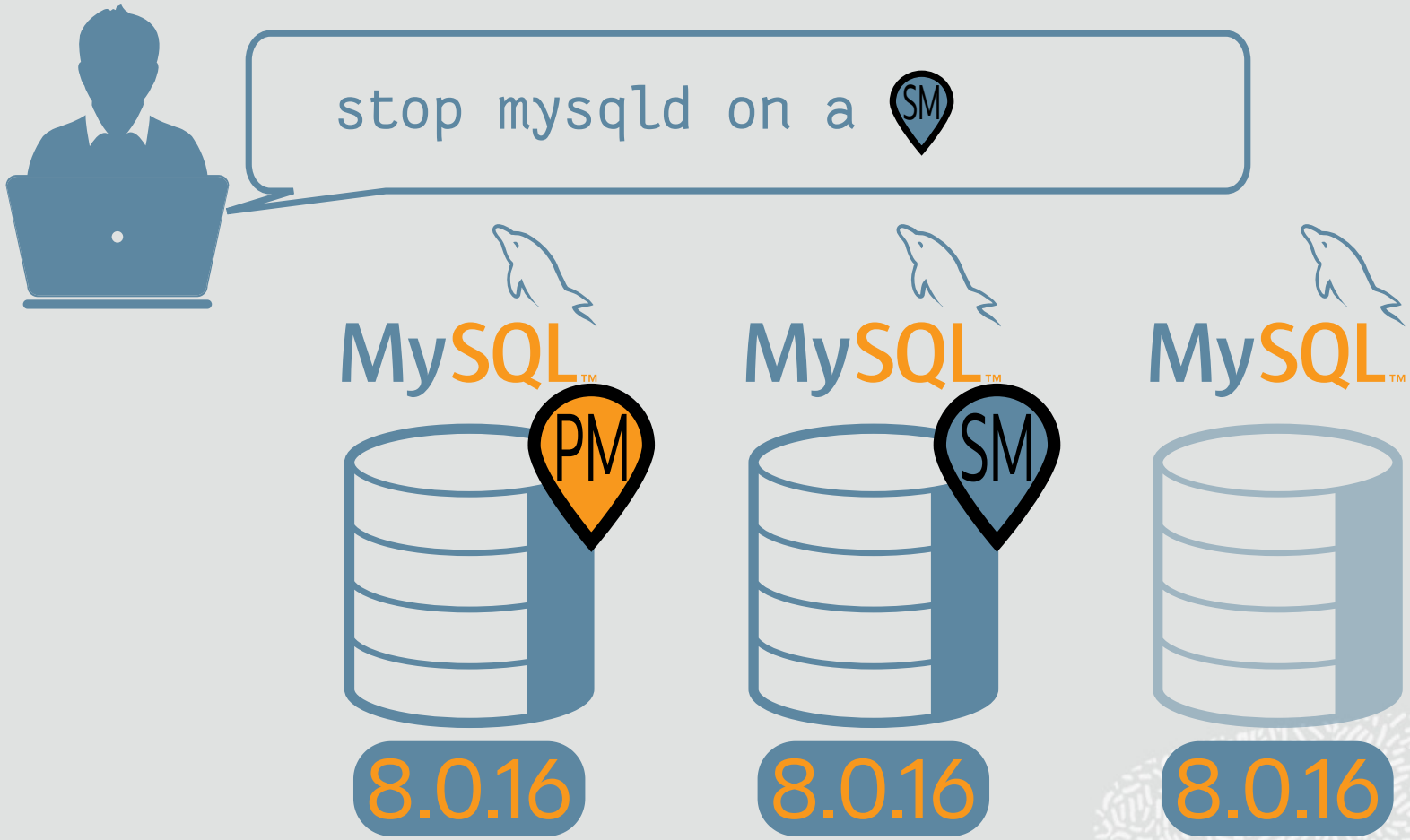
Upgrading Process



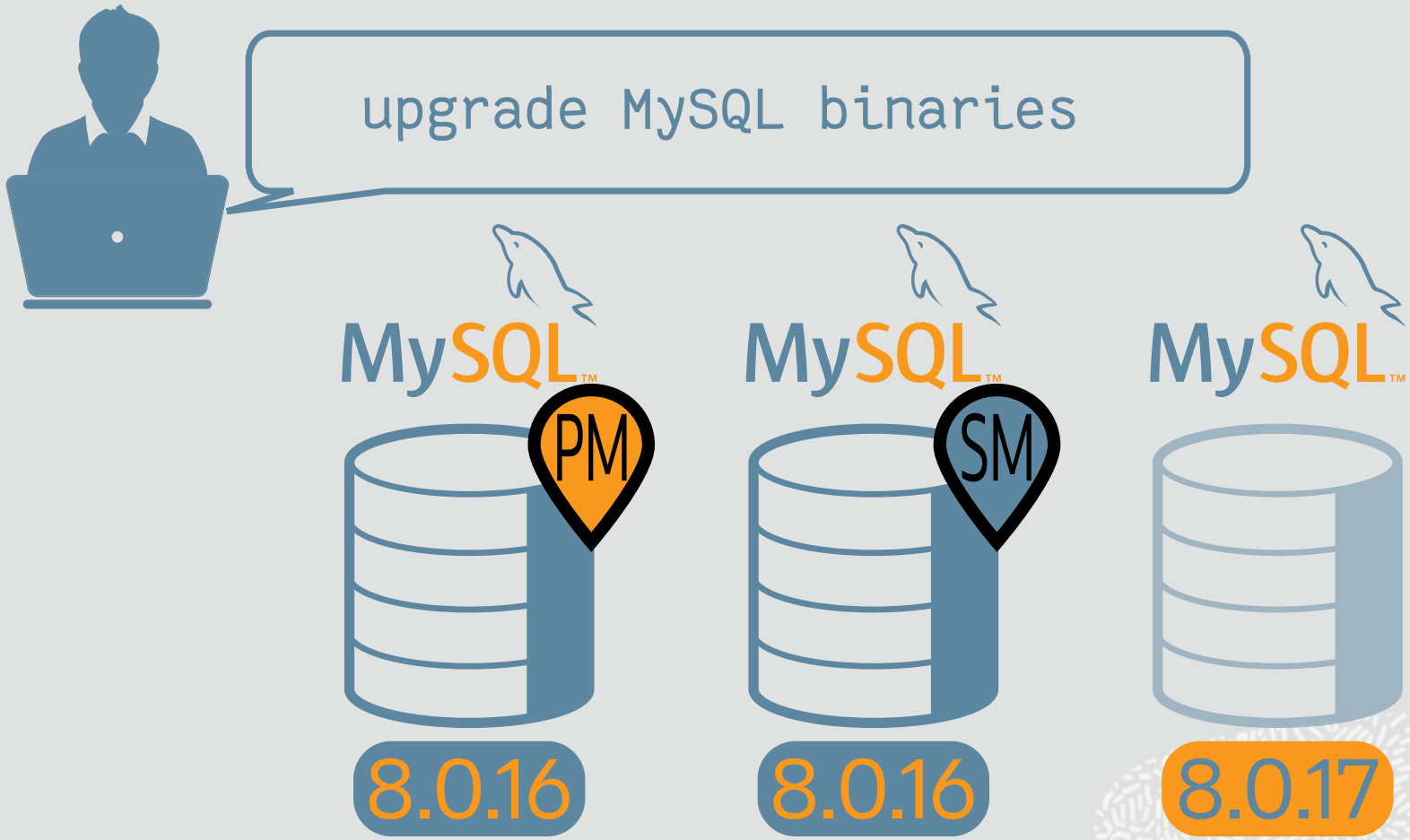
Upgrading Process



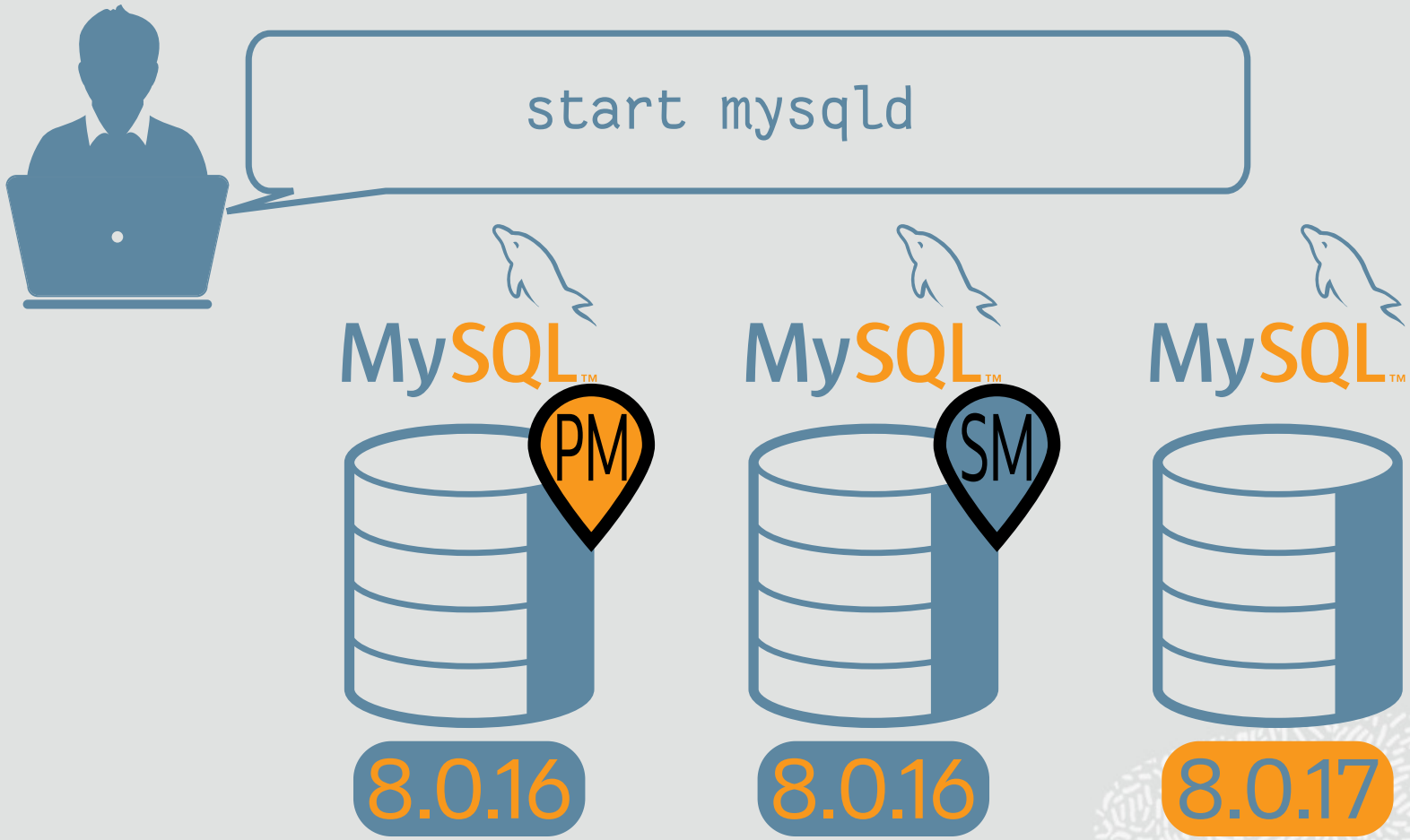
Upgrading Process



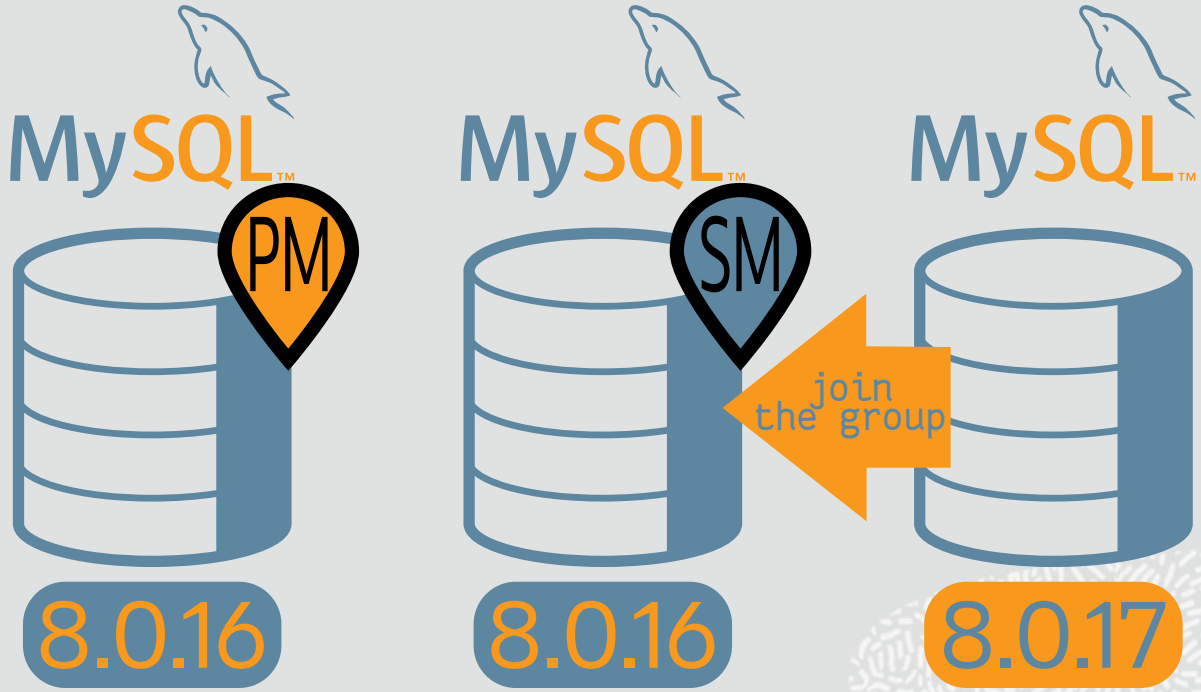
Upgrading Process



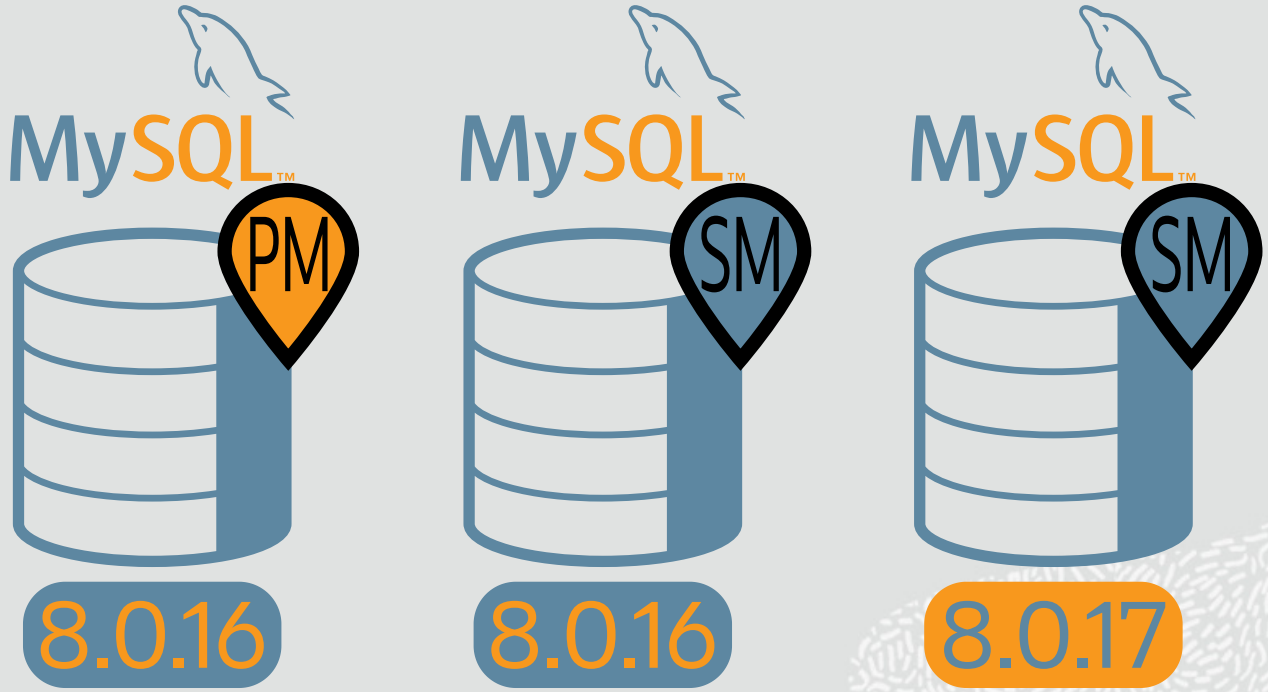
Upgrading Process



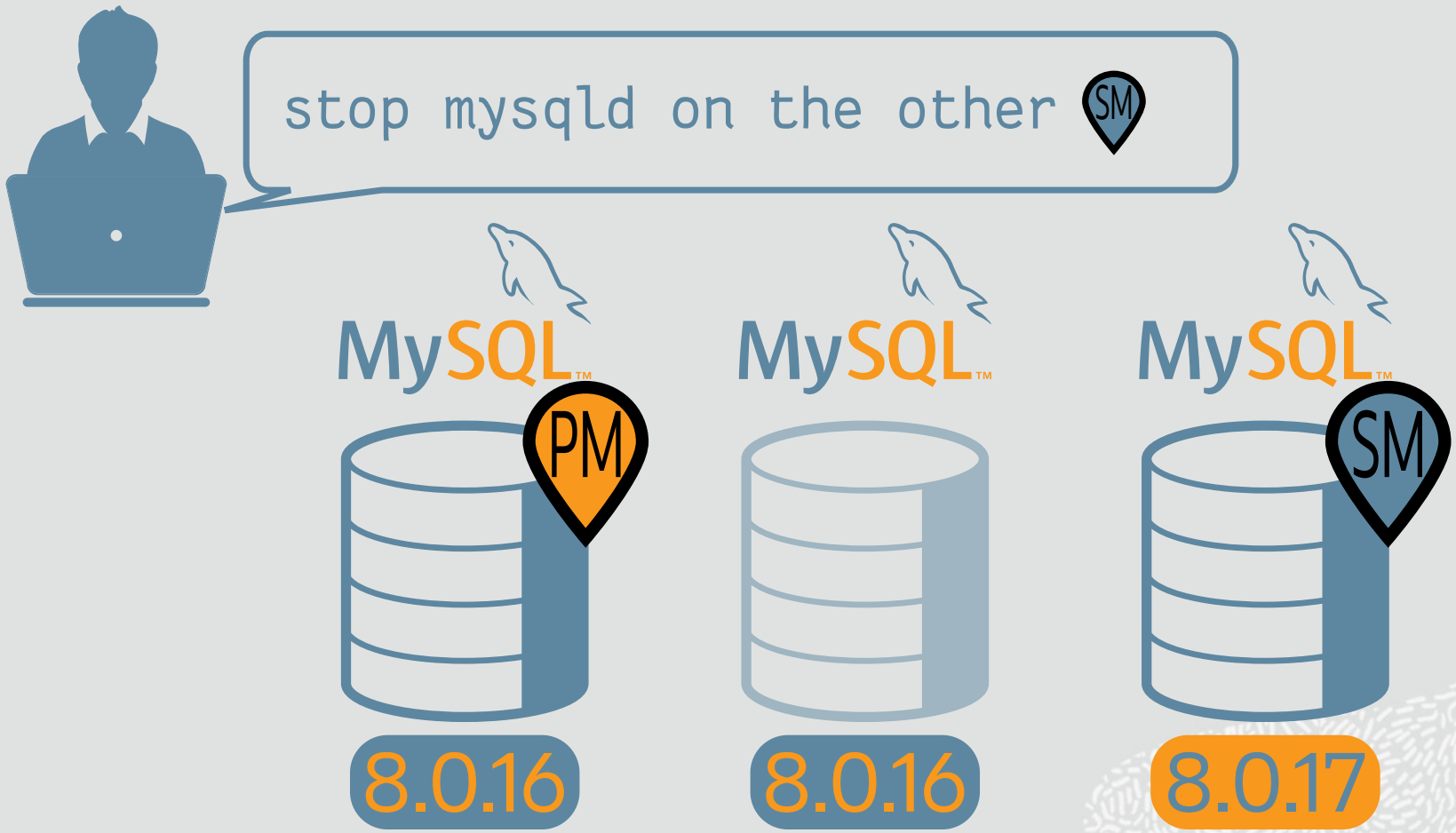
Upgrading Process



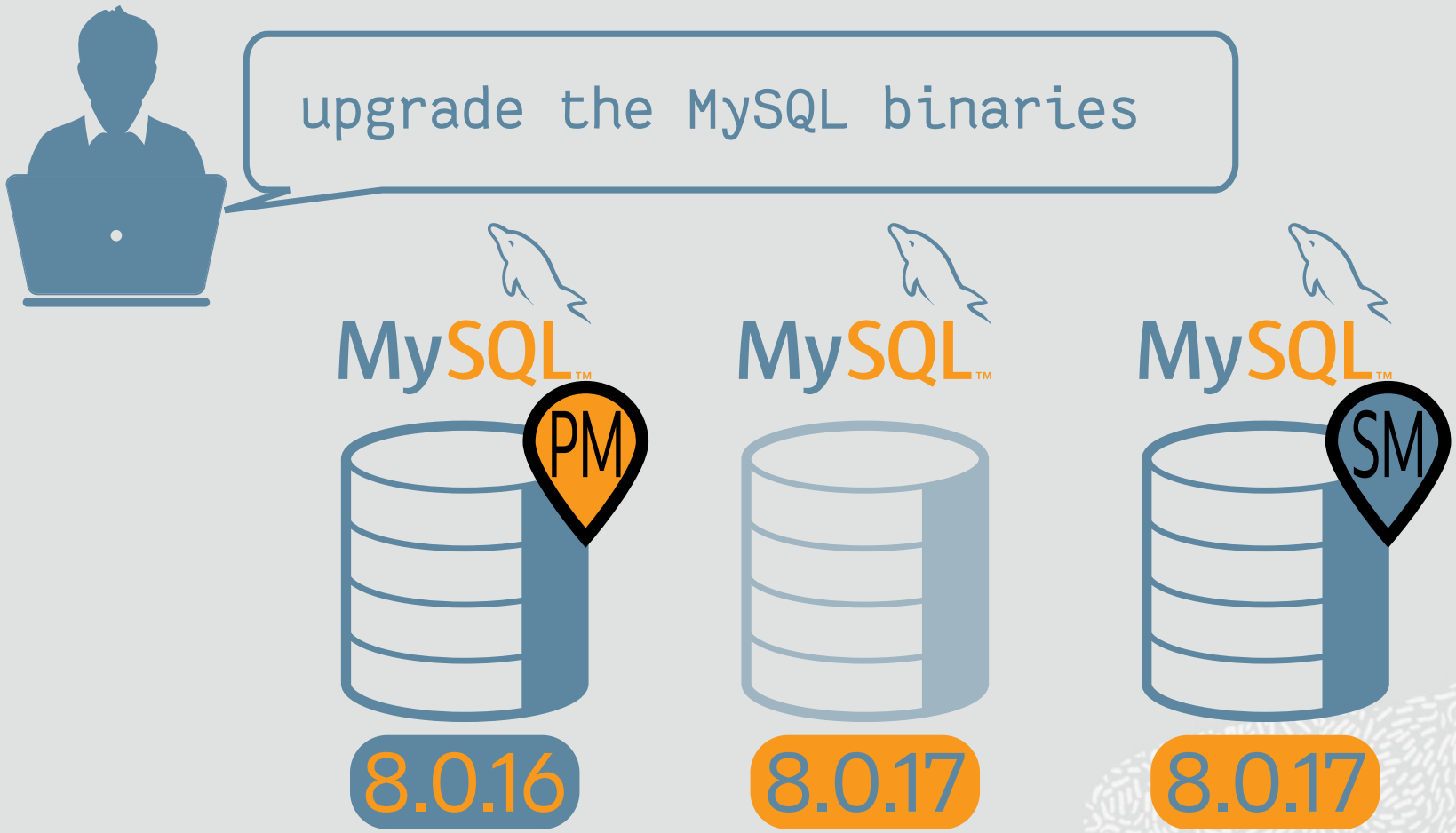
Upgrading Process



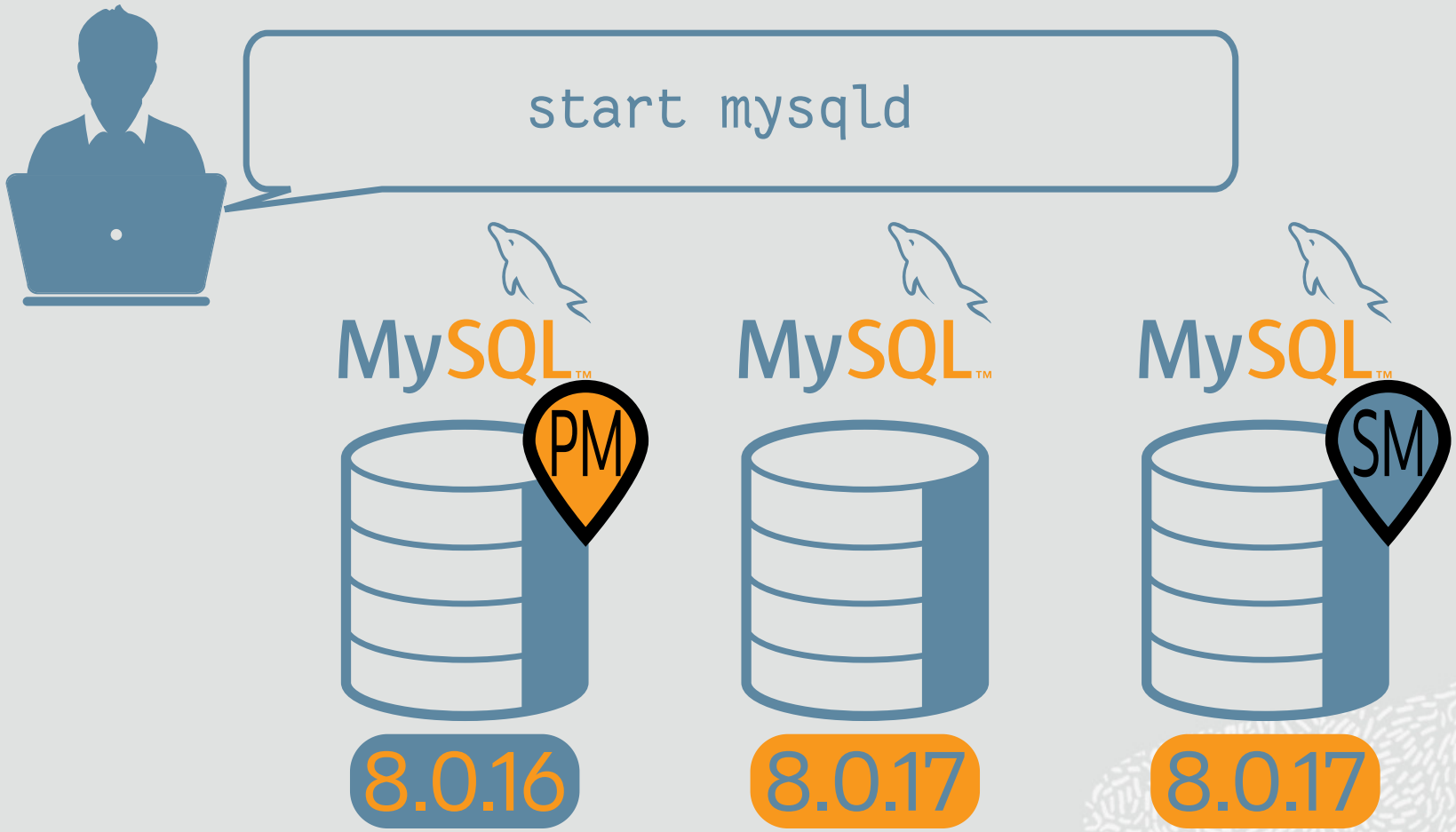
Upgrading Process



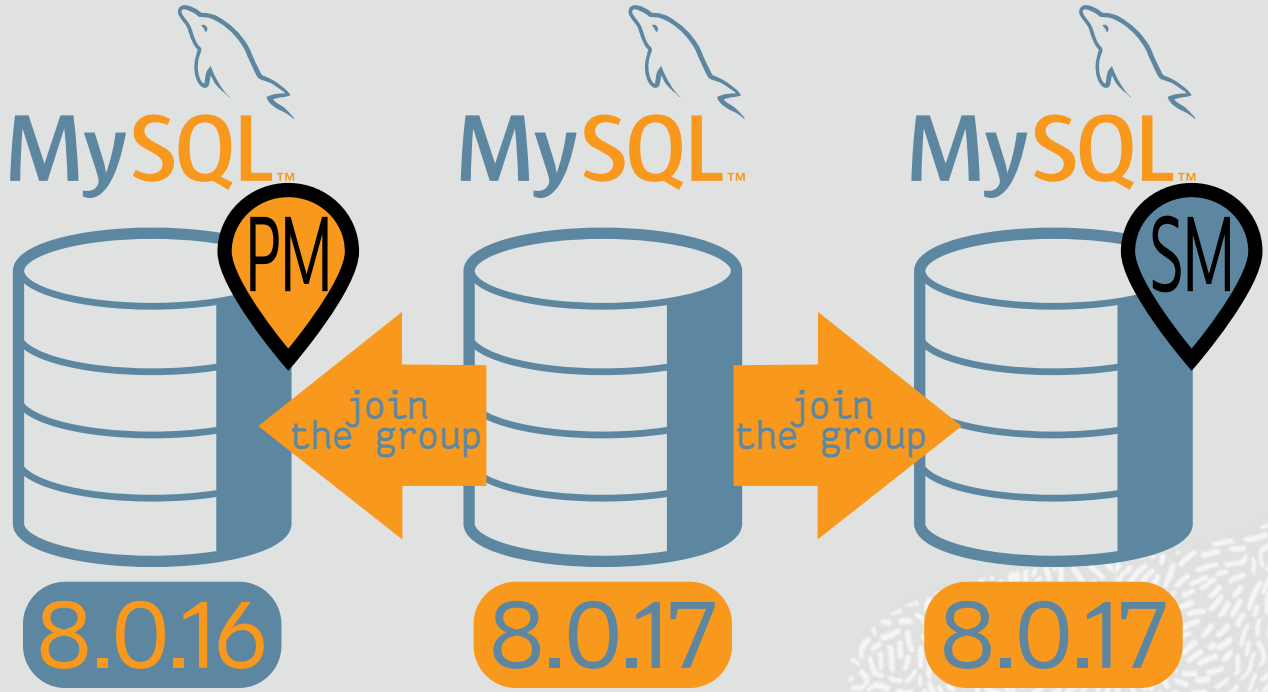
Upgrading Process



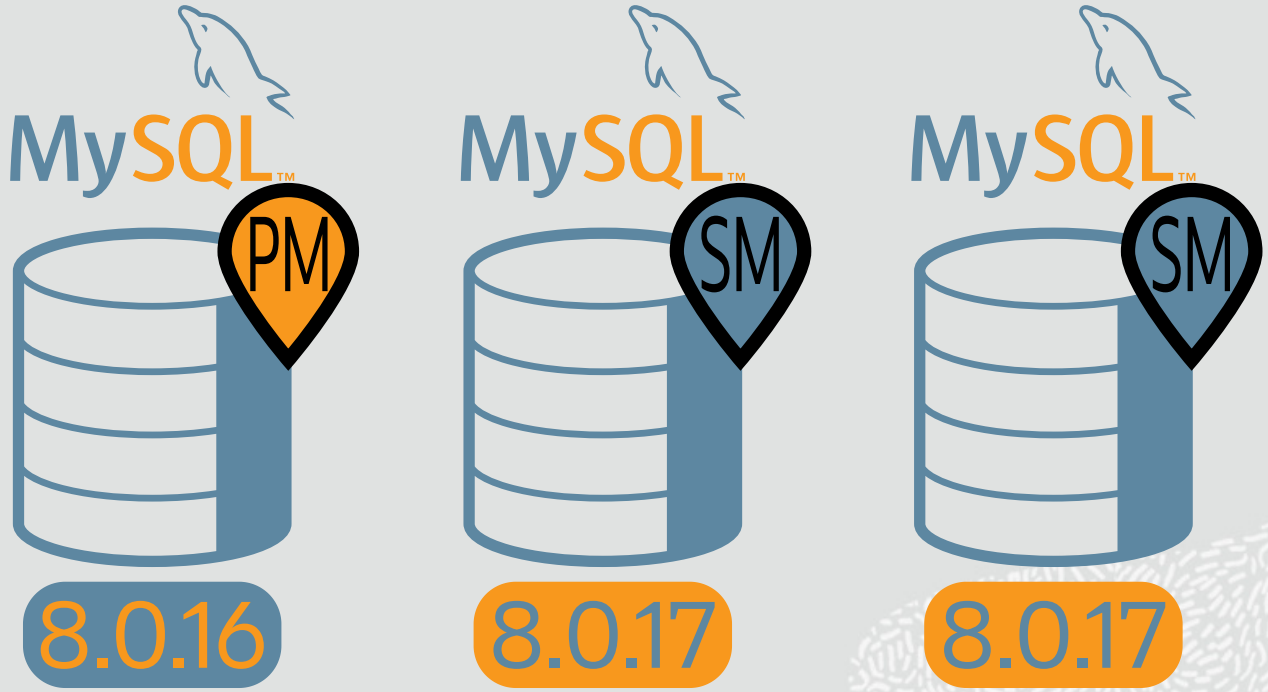
Upgrading Process



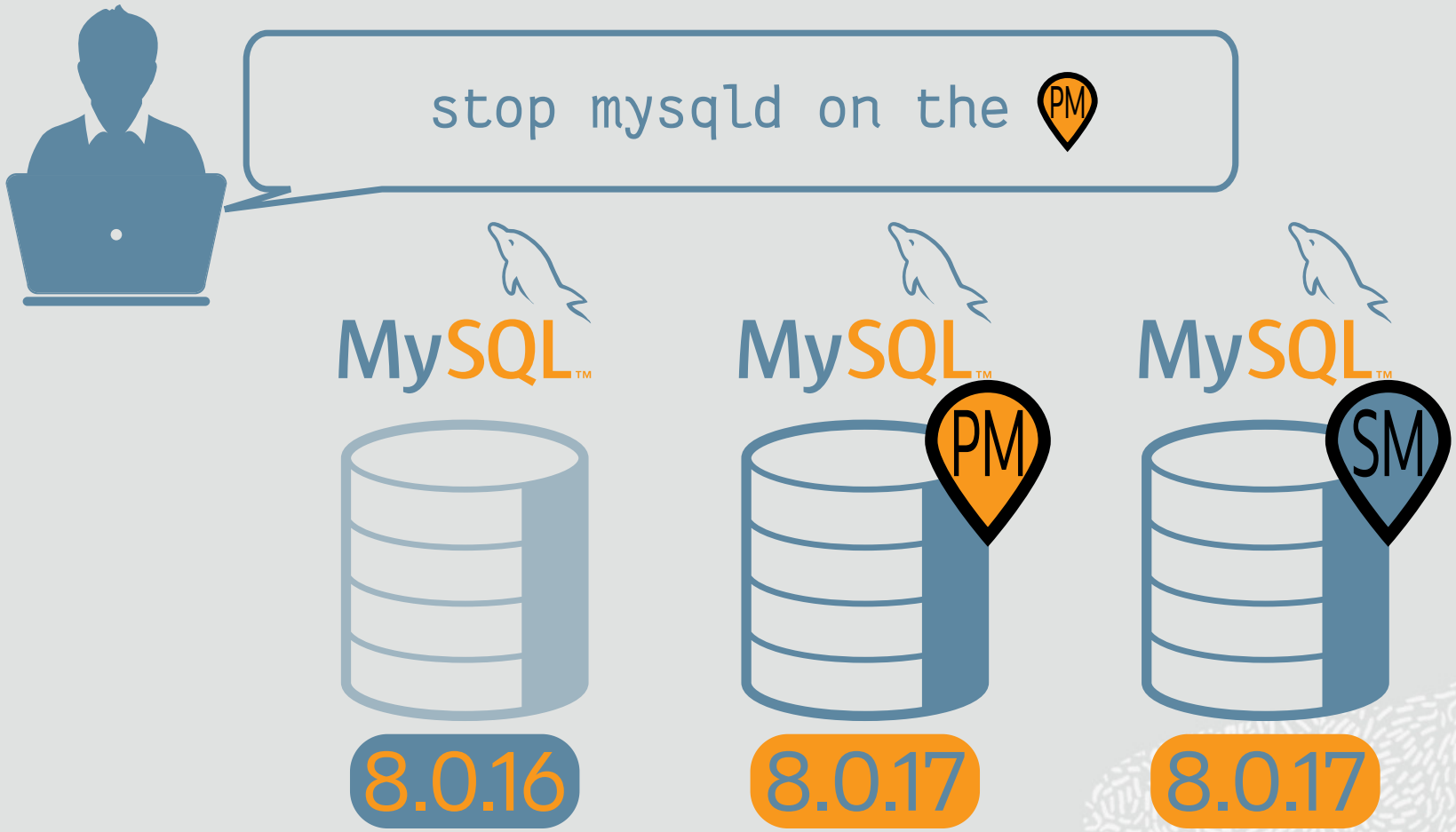
Upgrading Process



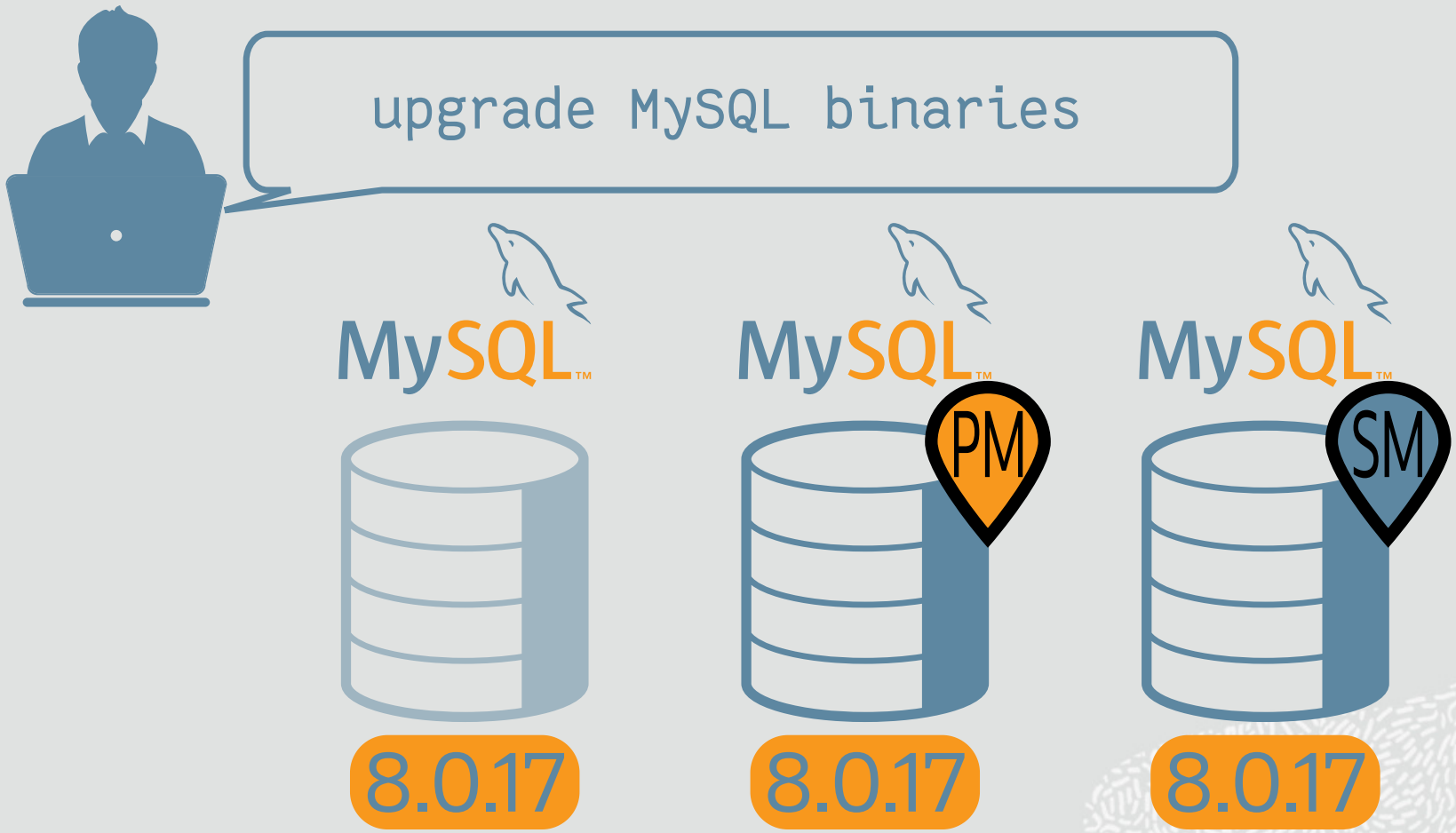
Upgrading Process



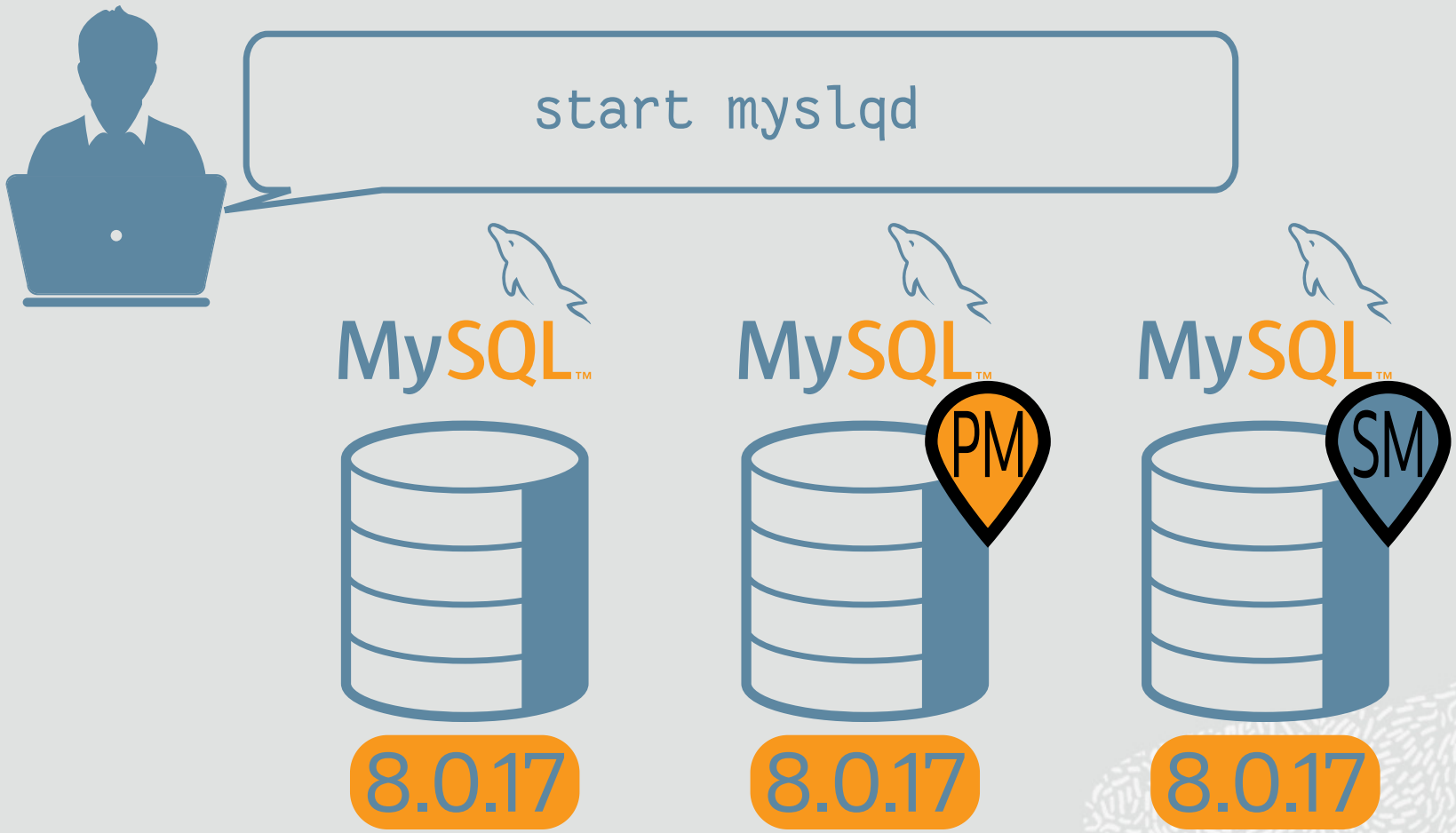
Upgrading Process



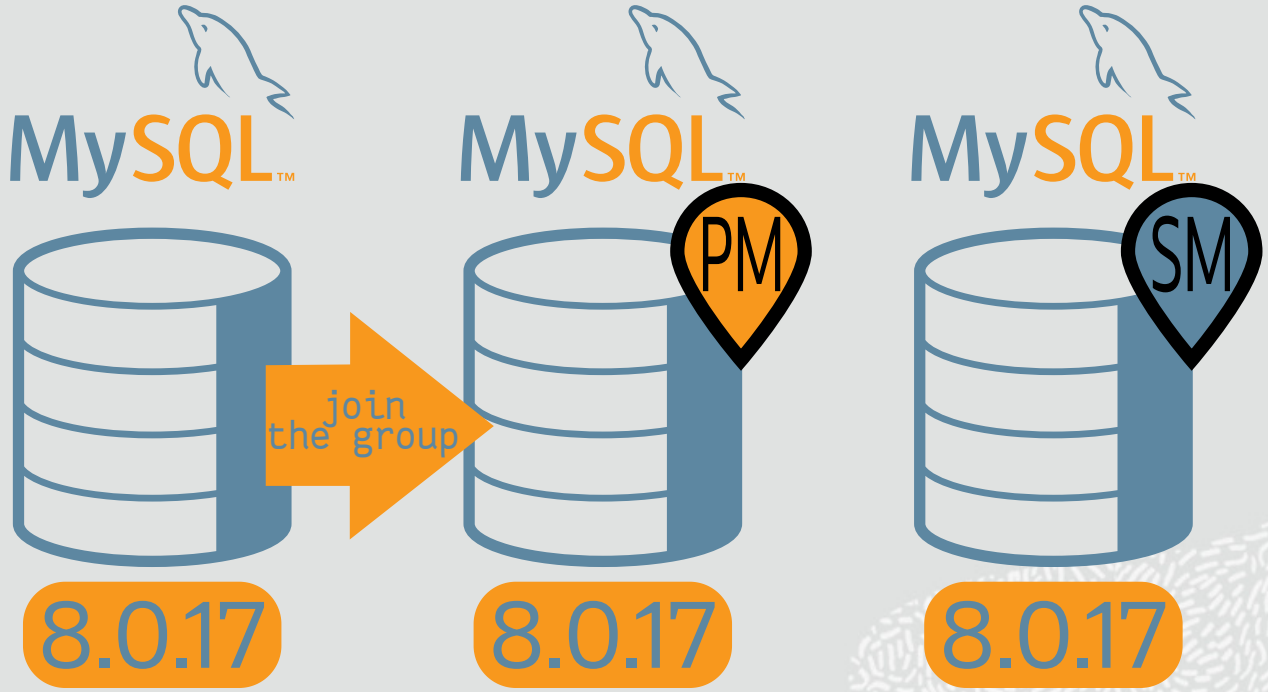
Upgrading Process



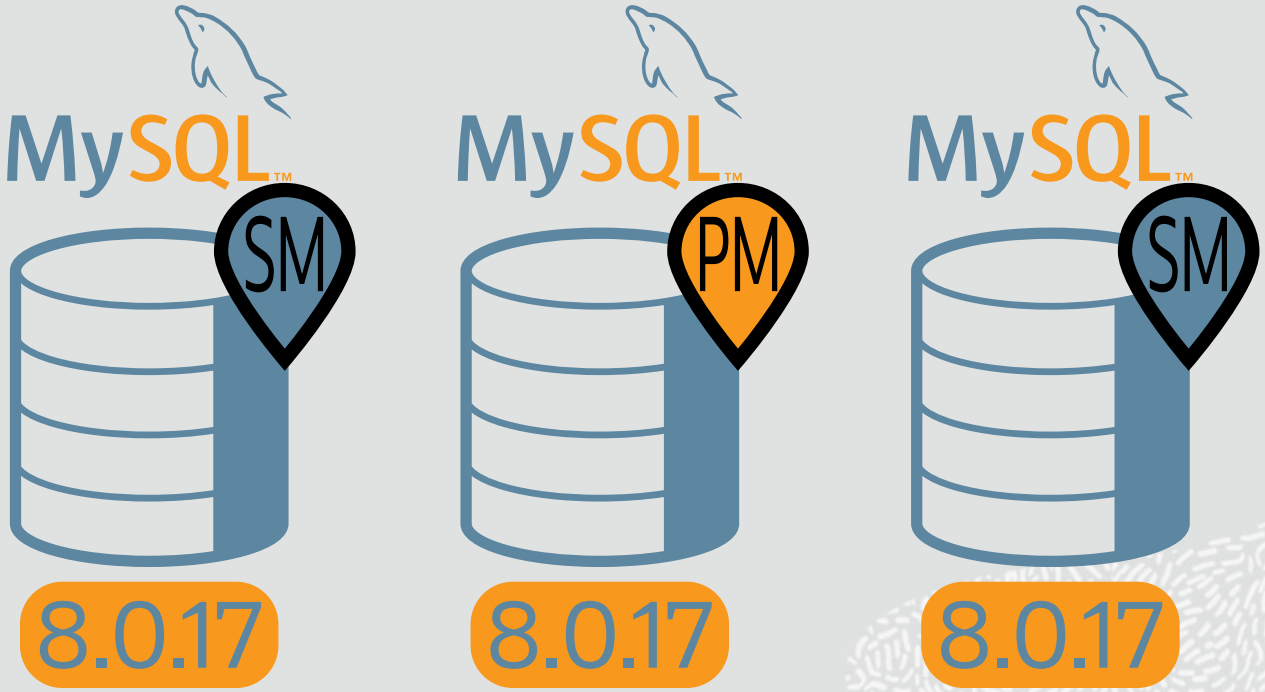
Upgrading Process



Upgrading Process



Upgrading Process



Upgrading Process Info

- The lowest version of a Group must always be the Primary Master
- If ejected from the Group, the lowest version instance won't be able to rejoin the Group again

Downgrading

Downgrading an existing instance won't work as downgrading MySQL Server is not supported:

```
[ERROR] [MY-013171] [InnoDB] Cannot boot server version 80016 on data directory built by version 80017.
                             Downgrade is not supported.
```

However it's possible to downgrade your MySQL InnoDB Cluster without downtime, by using a previous backup.

Before 8.0.17, any 8.0.x could join a MySQL InnoDB Cluster 8.0. Since 8.0.17, the minor version is also checked.

Downgrading (2)

So if you want to have a lower version to join a cluster, you need to set `group_replication_allow_local_lower_version_join` to **`ON`** on the joiner.

But don't forget that this instance running with a lower version should be promoted to be the **Primary-Master** as soon as possible to avoid any possible issues related to protocol and variable differences.

Backup

Taking a backup of your MySQL InnoDB Cluster is trivial. You can use exactly the same method as you are using for a standalone MySQL Server:

- Physical (MySQL Enterperise Backup (MEB), Xtrabackup)
- Logical (mysqldump, mysqlpump, mydumper)
- Snapshot (lvm, MySQL Clone)

Backup (2)

As a DBA using MySQL InnoDB Cluster retains data consistency, I only want to backup one member of the Group and preferably a Secondary Master, no need to waste disk space with multiple backups of the same data.

Backup (2)

As a DBA using MySQL InnoDB Cluster retains data consistency, I only want to backup one member of the Group and preferably a Secondary Master, no need to waste disk space with multiple backups of the same data.

This is possible ! Let's define the best backup solution.

Backup (3)

The perfect backup solution for you MySQL InnoDB Cluster, should then perform the following steps:

- check if the node where the script run is part of the cluster
- check if the node is indeed a secondary master
- eventually check if the node is lagging behind (large apply queue)
- ensure that the backup is not running on another member

Backup (3)

The perfect backup solution for you MySQL InnoDB Cluster, should then perform the following steps:

- check if the node where the script run is part of the cluster
- check if the node is indeed a secondary master
- eventually check if the node is lagging behind (large apply queue)
- ensure that the backup is not running on another member

Here, you can find a working script calling MEB to use as a cron job :
https://lefred.be/content/how-to-backup-your-innodb-cluster/

DDLs

There is nothing special about DDLs. When DDLs are performed, the cluster is not blocked completely.

Of course, concurent DDLs and DMLs are not a good idea when using a cluster in **Multi-Primary** mode.

If you use a **Multi-Primary** Cluster, you must swicth to **Single-Primary** mode the time of the DDL.

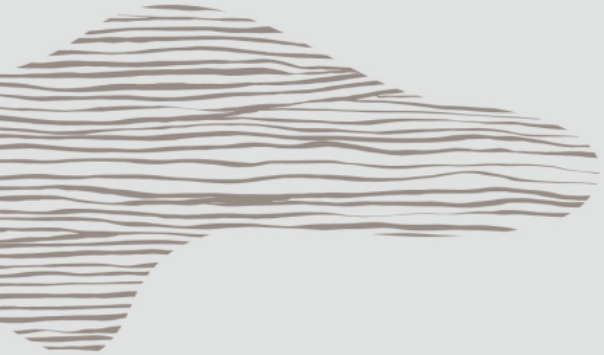
Don't forget that MySQL 8.0 supports also **Instant DDLs** like adding a column.

DDLs on Multi-Primary Clusters

```
 MySQL 8.0.17  mysql1:33060+  2019-09-07 20:53:29
JS \show gr_info
+-------------+---------+---------+--------+-----------+------------+-----------+----------+
| server      | role    | version | quorum | tx behind | tx to cert | remote tx | local tx |
+-------------+---------+---------+--------+-----------+------------+-----------+----------+
| mysql3:3306 | PRIMARY | 8.0.16  | YES    | 0         | 0          | 4         | 3        |
| mysql1:3306 | PRIMARY | 8.0.17  | YES    | 0         | 0          | 69        | 1479     |
| mysql2:3306 | PRIMARY | 8.0.17  | YES    | 0         | 0          | 1529      | 0        |
+-------------+---------+---------+--------+-----------+------------+-----------+----------+
 MySQL 8.0.17  mysql1:33060+  2019-09-07 20:53:31
JS cluster.switchToSinglePrimaryMode()
Switching cluster 'PerconaLiveEU' to Single-Primary mode...

Instance 'mysql3:3306' remains PRIMARY.
Instance 'mysql1:3306' was switched from PRIMARY to SECONDARY.
Instance 'mysql2:3306' was switched from PRIMARY to SECONDARY.

WARNING: Existing connections that expected a R/W connection must be disconnected, i.e. instances that became SECONDARY.

The cluster successfully switched to Single-Primary mode.
 MySQL 8.0.17  mysql1:33060+  2019-09-07 20:53:35
JS \show gr_info
+-------------+-----------+---------+--------+-----------+------------+-----------+----------+
| server      | role      | version | quorum | tx behind | tx to cert | remote tx | local tx |
+-------------+-----------+---------+--------+-----------+------------+-----------+----------+
| mysql3:3306 | PRIMARY   | 8.0.16  | YES    | 0         | 0          | 4         | 4        |
| mysql1:3306 | SECONDARY | 8.0.17  | YES    | 0         | 0          | 70        | 1479     |
| mysql2:3306 | SECONDARY | 8.0.17  | YES    | 0         | 0          | 1530      | 0        |
+-------------+-----------+---------+--------+-----------+------------+-----------+----------+
```

DDLs on Multi-Primary Clusters

```
 MySQL 8.0.16  mysql3:33060+  2019-09-07 20:55:33
JS \sql alter table clusterdemo.demo add column description text
Query OK, 0 rows affected (0.0721 sec)

Records: 0  Duplicates: 0  Warnings: 0
 MySQL 8.0.16  mysql3:33060+  2019-09-07 20:55:36
JS cluster.switchToMultiPrimaryMode()
Switching cluster 'PerconaLiveEU' to Multi-Primary mode...

Instance 'mysql3:3306' remains PRIMARY.
Instance 'mysql1:3306' was switched from SECONDARY to PRIMARY.
Instance 'mysql2:3306' was switched from SECONDARY to PRIMARY.

The cluster successfully switched to Multi-Primary mode.
```



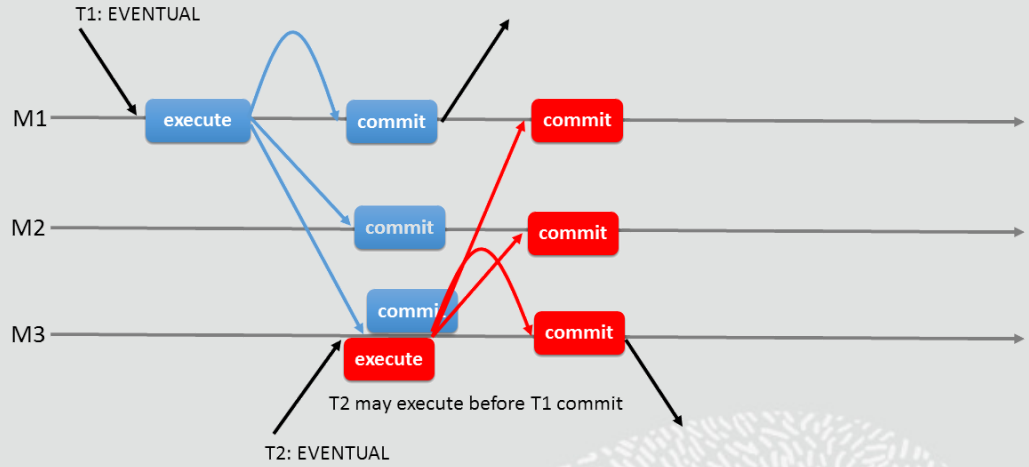
Configurable Consistency Guarantees

Consistency Levels

Consistency: EVENTUAL (default)

By default, there is no synchronization point for the transactions, when you perform a write on a node, if you immediately read the same data on another node, it is eventually there.

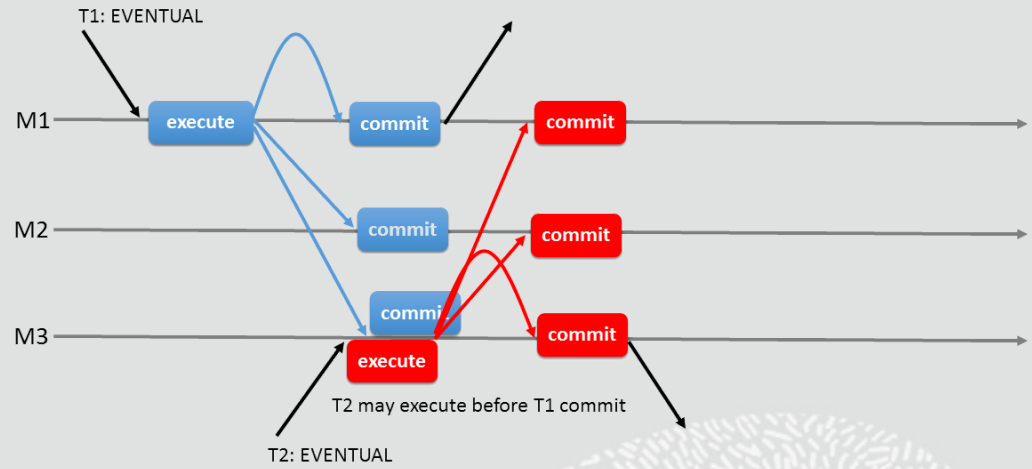
```
mysql> show variables like
       'group_replication_consistency';
+--------------------------------+-----------+
| Variable_name                  | Value     |
+--------------------------------+-----------+
| group_replication_consistency | EVENTUAL |
+--------------------------------+-----------+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



Consistency: EVENTUAL (default)

By default, there is no synchronization point for the transactions, when you perform a **write** on a node, if you immediately read the same data on another node, it is eventually there.

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



Since MySQL 8.0.16, we have the possibility to set the synchronization point at read or at write or both (globally or for a session).

Consistency: BEFORE (READ)

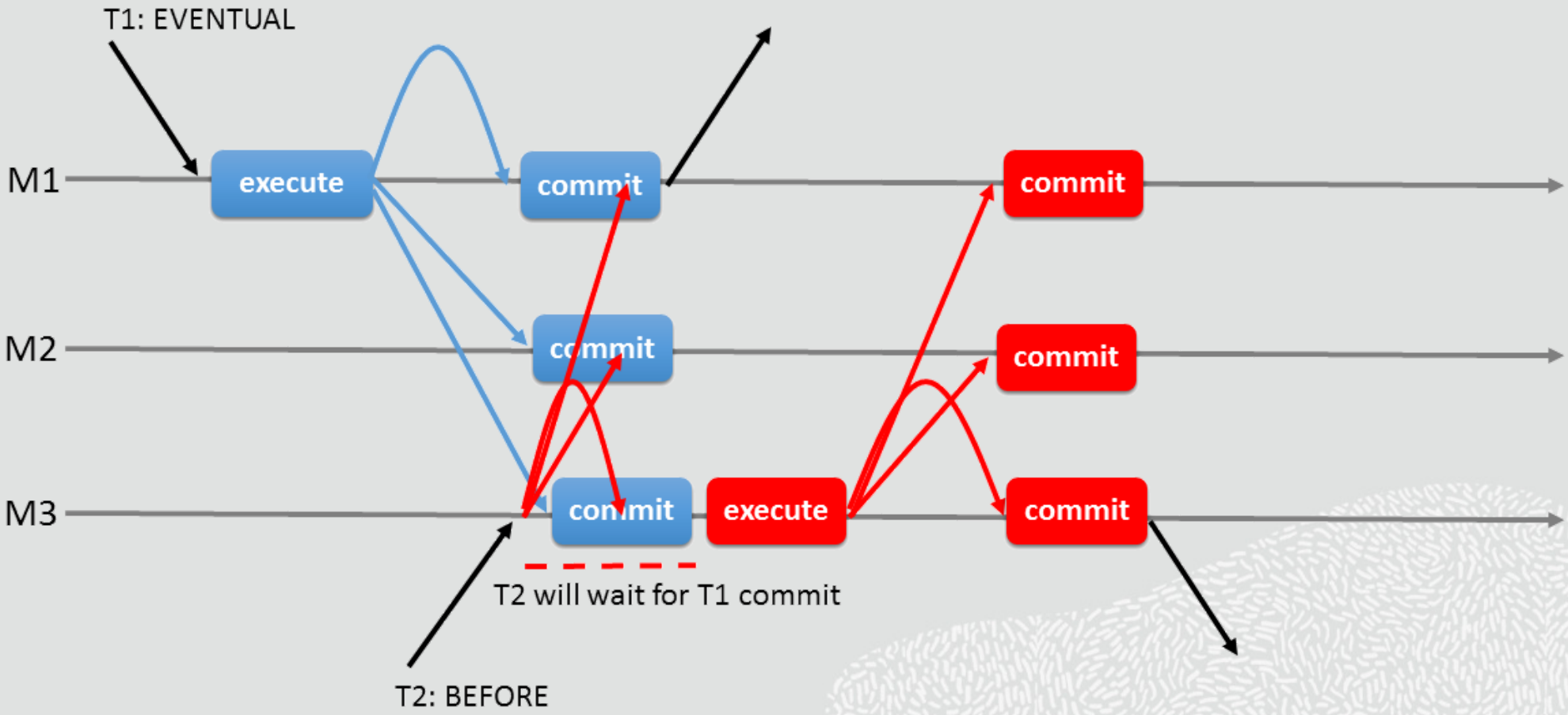
As a DBA, I want to load balance my reads without deploying additional restrictions on which server I read from to avoid reading stale data, my group writes are much more than my group reads.

Consistency: BEFORE (READ)

As a DBA, I want to load balance my reads without deploying additional restrictions on which server I read from to avoid reading stale data, my group writes are much more than my group reads.

As a developer, I want specific transactions in my workload to always read up-to-date data from the group, so that whenever that sensitive data is updated, I will enforce that reads shall read the most up to date value.

Consistency: BEFORE (READ)



Consistency: AFTER (WRITE)

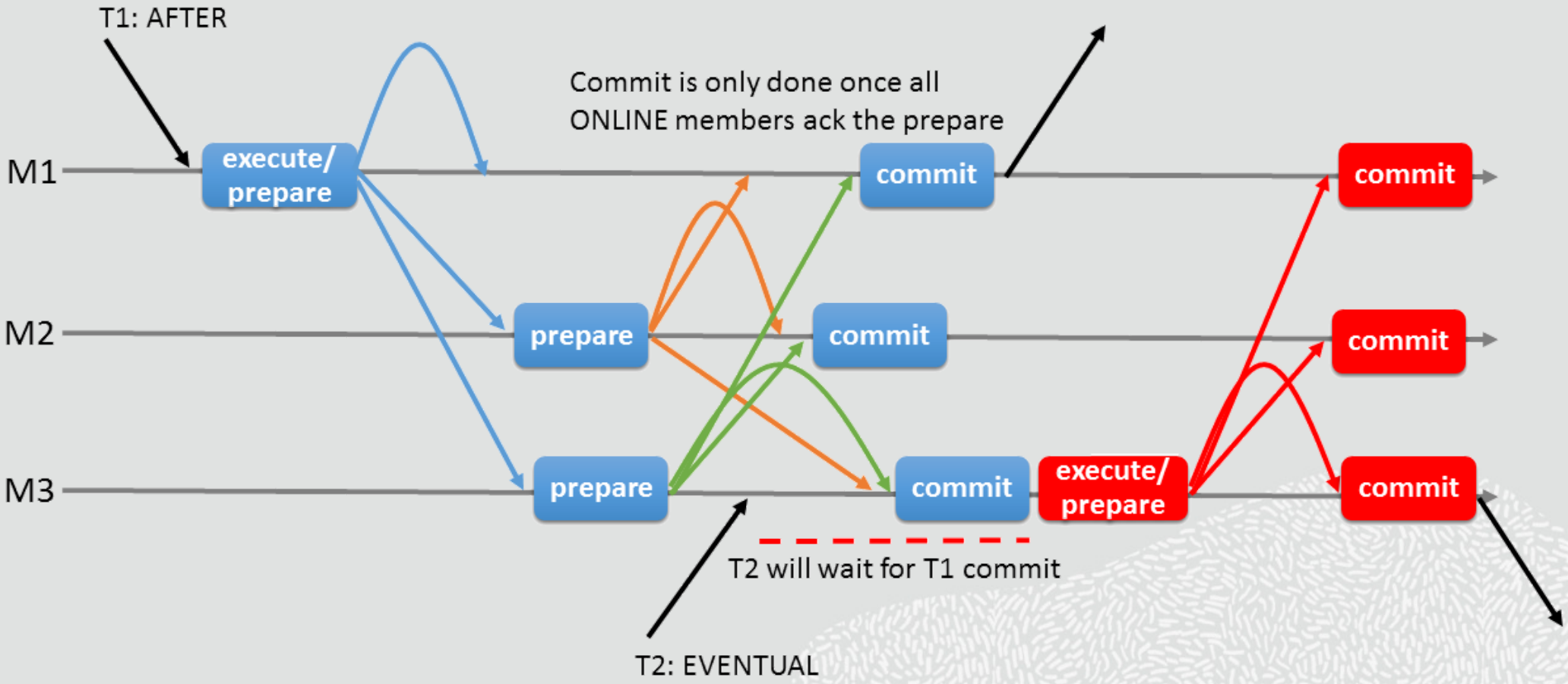
I want to load balance my reads without deploying additional restrictions on which server I read from to avoid reading stale data, my group writes are much less than my group reads.

Consistency: AFTER (WRITE)

I want to load balance my reads without deploying additional restrictions on which server I read from to avoid reading stale data, my group writes are much less than my group reads.

I have a group that mostly does reads-only, I want my read-write transactions to be applied everywhere once they commit, so that subsequent reads are done on up-to-date data that includes my latest write. Without paying at reads.

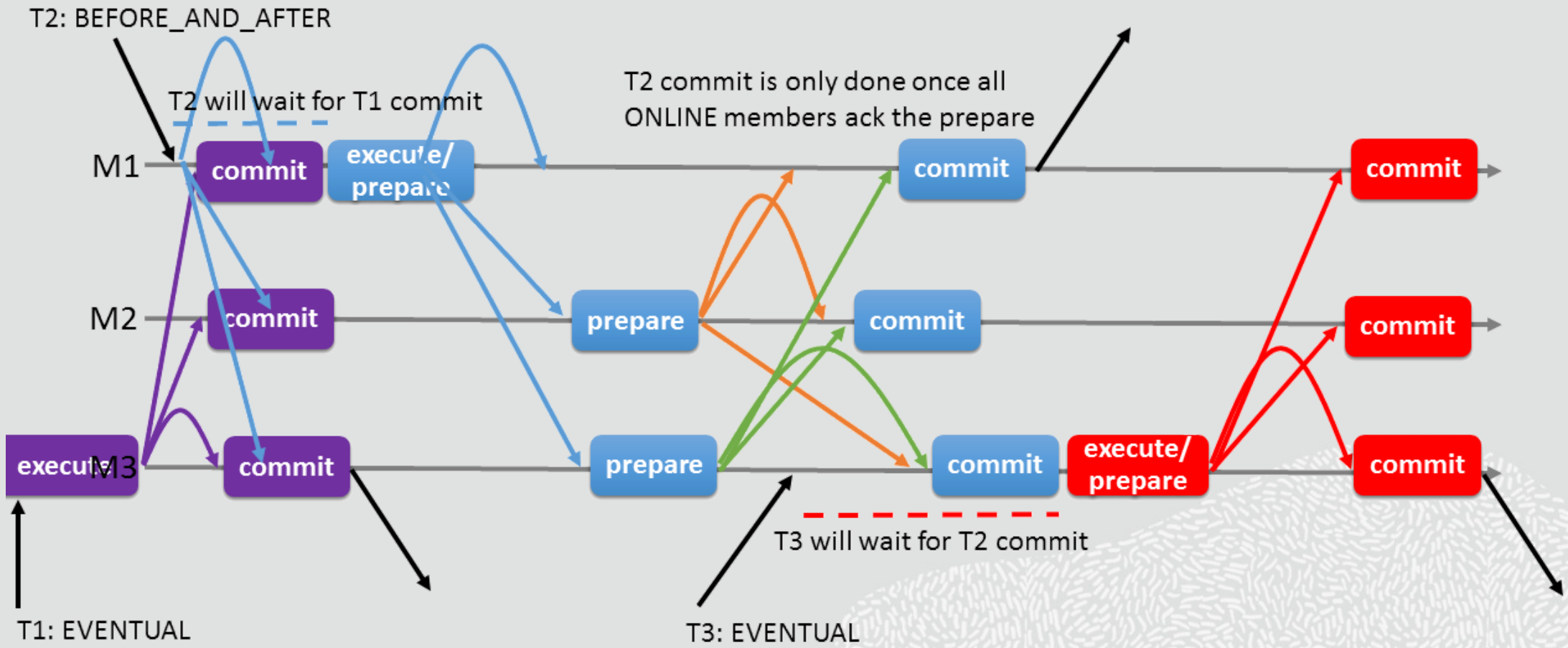
Consistency: AFTER (WRITE)



Consistency: BEFORE_AND_AFTER

I want that my application to replicate data as close as possible to synchronous.
And of course I'm OK to pay the required price !

Consistency: BEFORE_AND_AFTER



Defining Consistency

- The consistency level of the MySQL InnoDB Cluster can be defined during the creation:

```
 MySQL 8.0.17  mysql1:33060+  2019-09-09 16:58:13
JS dba.createCluster("fooBar", {consistency: "AFTER"})
A new InnoDB cluster will be created on instance 'mysql1:3306'.

Disabling super_read_only mode on instance 'mysql1:3306'.
Validating instance at mysql1:3306...

This instance reports its own address as mysql1:3306

Instance configuration is suitable.
Creating InnoDB cluster 'fooBar' on 'mysql1:3306'...

Adding Seed Instance...
Cluster successfully created. Use Cluster.addInstance() to add MySQL instances.
At least 3 instances are needed for the cluster to be able to withstand up to
one server failure.

<Cluster:fooBar>
```

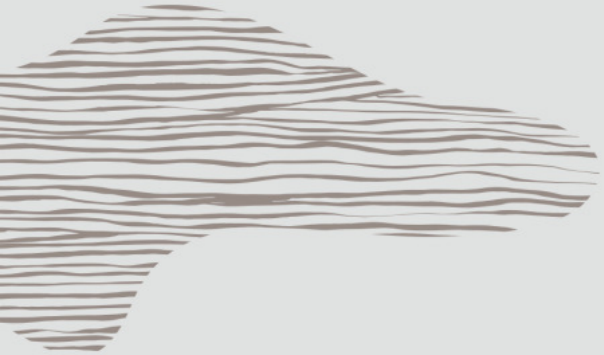
Defining Consistency (2)

- The consistency can be globally changed in MySQL Shell:

```
MySQL 8.0.17 mysql1:33060+ 2019-09-09 16:11:51
JS cluster.setOption("consistency","BEFORE")
Setting the value of 'consistency' to 'BEFORE' in all ReplicaSet members ...

Successfully set the value of 'consistency' to 'BEFORE' in the 'default' ReplicaSet.
```

- It's also possible and recommended for the most restrictive levels to define it at statement session level:

```
MySQL 8.0.17 mysql1:33060+ 2019-09-09 20:57:37
SQL set group_replication_consistency=BEFORE_AND_AFTER;
Query OK, 0 rows affected (0.0010 sec)
MySQL 8.0.17 mysql1:33060+ 2019-09-09 20:58:07
SQL insert into clusterdemo.demo (hostname) values ('lefred');
Query OK, 1 row affected (0.1135 sec)
```



MySQL InnoDB Cluster

MySQL Router Specific Settings

MySQL Router : Metadata Cache

The settings to access MySQL InnoDB Cluster's Metadata Cache is defined in the `[metadata_cache:<name>]` section of MySQL Router's config file.

MySQL Router : Metadata Cache

The settings to access MySQL InnoDB Cluster's Metadata Cache is defined in the `[metadata_cache:<name>]` section of MySQL Router's config file.

The most interesting settings is:

- `ttl`: Time to live (in seconds) of information in the metadata cache. One milissecond is `0.001`. To continously query the the metadata cache in a tight loop, set it to `0` (the default value is `0.5`).

MySQL Router : Metadata Cache (2)

As of MySQL Router 8.0.17, `ttl` is now obsolete as you can enable `use_gr_notifications` to refresh metadata on each of the four notifications that Group Replication sends:

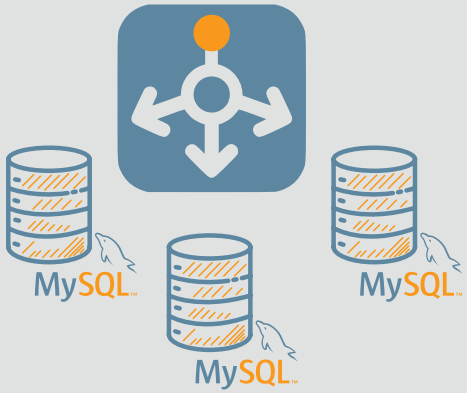
- `group_replication/membership/quorum_loss`
- `group_replication/membership/view`
- `group_replication/status/role_change`
- `group_replication/status/state_change`

MySQL Router

By default, when *bootstrapped*, MySQL Router is configured using **`first-available`** as routing strategy for R/W connections (to the Primary-Master).

For R/O connections (to Secondary-Masters), the strategy is **`round-robin-with-fallback`**.

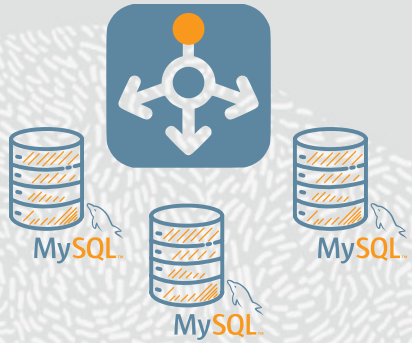
This is actually setup for the MySQL Classic Protocol and MySQL X Protocol.



MySQL Router : Routing Strategy Methods

- `first-available`: the new connection is routed to the first available server from the destinations list. In case of failure, the next available server is used. This cycle continues until all servers are unavailable.

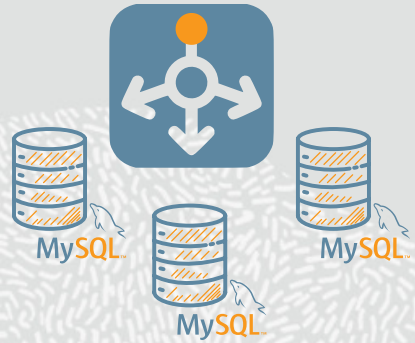
- `round-robin-with-fallback`: for load-balancing, each new connection is made to the next available secondary server in a round-robin fashion. If a secondary server is not available then servers from the primary list are used in round-robin fashion.



MySQL Router : Routing Strategy Methods

- `round-robin`: for load-balancing, each new connection is made to the next available server in a round-robin fashion.

- `next-available`: like first-available, in that the new connection is routed to the first available server from the destinations list. Unlike first-available, if a server is marked as unreachable then it gets discarded and is never used again as a destination



MySQL Router : Routing destinations

Now that we have defined a strategy for the MySQL Router, we also need to define destinations.

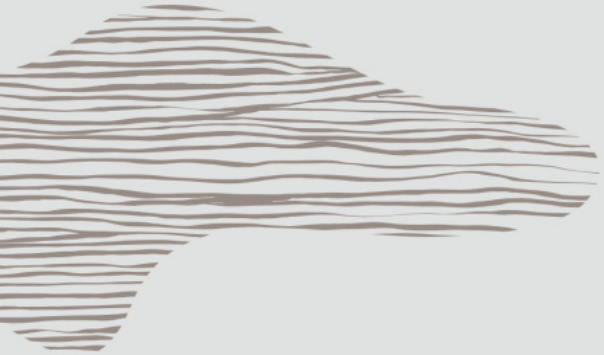
Destinations (`desinations` in `[routing:<name>]` section), can be a comma-separated list of MySQL Servers, or a metadata-cache definition:

MySQL Router : Routing destinations

Now that we have defined a strategy for the MySQL Router, we also need to define destinations.

Destinations (**desinations** in **[routing:<name>]** section), can be a comma-separated list of MySQL Servers, or a metadata-cache definition:

```
[routing:PerconaLiveEU_default_rw]
bind_address=0.0.0.0
bind_port=6446
destinations=metadata-cache://PerconaLiveEU/default?role=PRIMARY
routing_strategy=first-available
protocol=classic
```



MySQL InnoDB Cluster

Configuration & Settings

Instance Specific Settings

Within the MySQL Shell you can define some Group Replication specific settings via the Admin API that will only affect the targeted instance:

- **exitStateAction**: string value indicating the group replication exit state action. (`ABORT_SERVER`, `READ_ONLY` and `OFFLINE_MODE` [8.0.18])
- **memberWeight**: integer value with a percentage weight for automatic primary election on failover.
- **autoRejoinTries**: integer value to define the number of times an instance will attempt to rejoin the cluster after being expelled.

Instance Specific Settings (2)

- **label**: a string identifier of the instance.

```
MySQL 8.0.17  mysql3:33060+  2019-09-12 10:09:49
JS cluster.setInstanceOption('mysql3:3306','label','node3')
Setting the value of 'label' to 'node3' in the instance: 'mysql3:3306' ...

Successfully set the value of 'label' to 'node3' in the 'default' ReplicaSet member: 'mysql3:3306'.
```

Group Specific Settings

Within the MySQL Shell you have the possibility to define some Group specific settings via the Admin API:

- **clusterName**: string value to define the cluster name.
- **exitStateAction**: string value indicating the group replication exit state action.
- **memberWeight**: integer value with a percentage weight for automatic primary election on failover.
- **consistency**: string value indicating the consistency guarantees that the cluster provides.

Group Specific Settings (2)

- **expelTimeout**: integer value to define the time period in seconds that cluster members should wait for a non-responding member before evicting it from the cluster.
- **autoRejoinTries**: integer value to define the number of times an instance will attempt to rejoin the cluster after being expelled.
- **disableClone**: boolean value used to disable the clone usage on the cluster.

```
JS cluster.setOption("autoRejoinTries",5)
WARNING: Each cluster member will only proceed according to its exitStateAction if auto-rejoin fails (i.e. all retry attempts are exhausted).

Setting the value of 'autoRejoinTries' to '5' in all ReplicaSet members ...

Successfully set the value of 'autoRejoinTries' to '5' in the 'default' ReplicaSet.
```

Upgrade to MySQL 8.0

It's time to upgrade to MySQL 8.0, the fastest MySQL adoption release ever !

Meet the MySQL Team at the Conference


Sunny Bains


Frédéric Descamps


Pedro Gomes


Luis Soares


Dimitri Kravtchuk

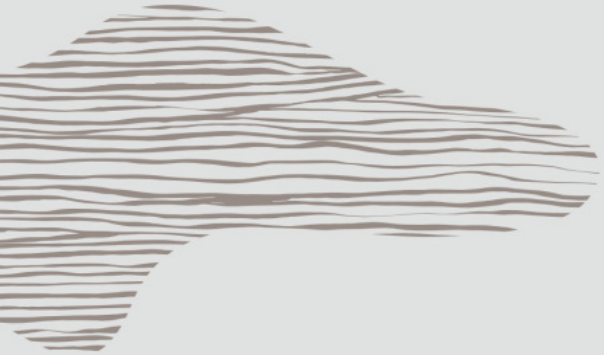

Georgi Kodinov


Kenny Gryp


Ståle Deraas


Norvald Ryeng


Geir Høydalsvik



Thank you !