dinfratechsource

Welcome to dinfratechsource. One Stop Solution for IT Infrastructure

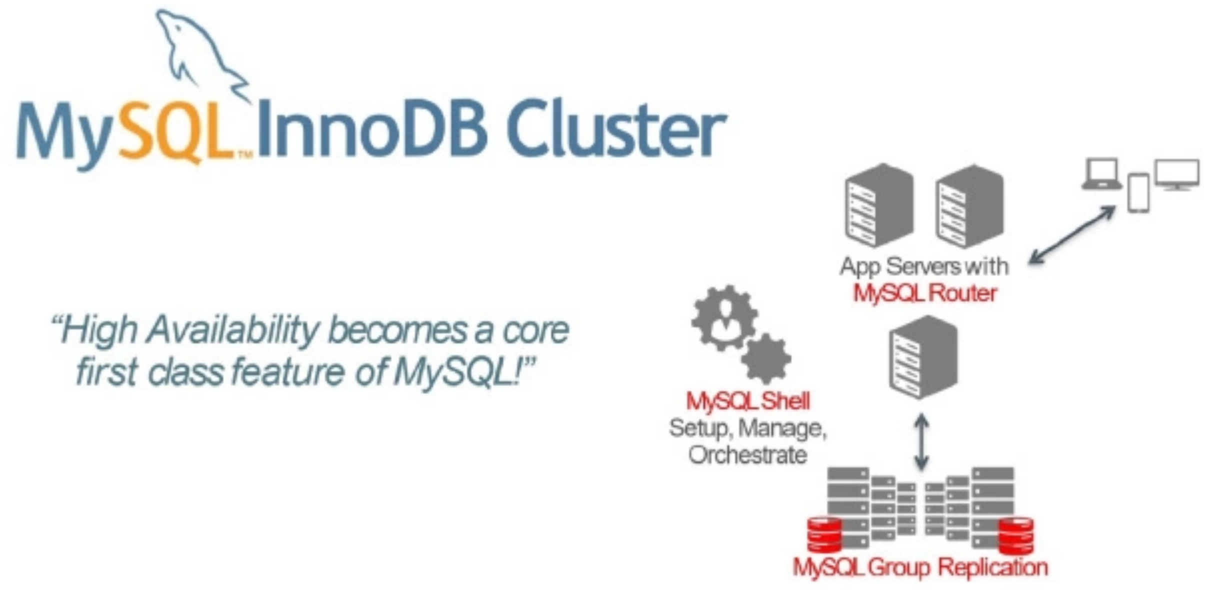
MySQL InnoDB Cluster – A complete High Availability solution for MySQL

MySQL InnoDB cluster provides a complete high availability solution for MySQL.Each MySQL server instance runs MySQL Group Replication, which provides the mechanism to replicate data within InnoDB clusters, with built-in failover.

In the default single-primary mode, an InnoDB cluster has a single read-write server instance – the primary. Multiple secondary server instances are replicas of the primary. If the primary fails, a secondary is automatically promoted to the role of primary.

InnoDB Cluster uses theGroup Replication plugin to allow for virtually synchronous replication, while also providing a MySQL Router that is aware of the cluster state.

InnoDB Cluster also provides a new MySQL Shell to interact with the cluster commands.


MySQL InnoDB Cluster
"High Availability becomes a core first class feature of MySQL!"
App Servers with MySQL Router
MySQL Shell
Setup, Manage, Orchestrate
MySQL Group Replication

<u>InnoDB Cluster Requirements</u>

InnoDB cluster uses Group Replication and therefore your server instances must meet the same requirements.

- All the tables must have **InnoDB storage engine**.

- **Primary Keys** : Every table that is to be replicated by the group must have a defined primary key, or primary key equivalent where the equivalent is a non-null unique key.
- **IPv4 Network** : The group communication engine used by MySQL Group Replication only supports IPv4.
- **Network Performance** : Group Replication is designed to be deployed in a cluster environment where server instances are very close to each other, and is impacted by both network latency as well as network bandwidth.
- Set Binary Log Active (Set –log-bin[=log_file_name])
- **Slave Updates Logged** : (Set –log-slave-updates). Servers need to log binary logs that are applied through the replication applier. Servers in the group need to log all transactions that they receive and apply from the group. This is required because recovery is conducted by relying on binary logs form participants in the group.
- **Set Binary Log format to ROW** : Set –binlog-format=row. Group Replication relies on row-based replication format to propagate changes consistently among the servers in the group.
- **Global Transaction Identifiers On** : (Set –gtid-mode=ON). Group Replication uses global transaction identifiers to track exactly which transactions have been committed on every server instance and thus be able to infer which servers have executed transactions that could conflict with already committed transactions elsewhere.
- **Replication Information Repositories** : (Set –master-info-repository=TABLE) and –relay-log-info-repository=TABLE. The replication

applier needs to have the master information and relay log metadata written to the mysql.slave_master_info and mysql.slave_relay_log_info system tables.
- **Transaction Write Set Extraction** : Set –transaction-write-set-extraction=XXHASH64 so that while collecting rows to log them to the binary log, the server collects the write set as well. The write set is based on the primary keys of each row and is a simplified and compact view of a tag that uniquely identifies the row that was changed.
- **Multi-threaded Appliers** : Group Replication members can be configured as multi-threaded appliers, enabling transactions to be applied in parallel. Set –slave-parallel-workers=N (where N is the number of parallel applier threads), –slave-preserve-commit-order=1, and –slave-parallel-type=LOGICAL_CLOCK. Setting –slave-parallel-workers=N enables the multi-threaded applier on the member. Group Replication relies on consistency mechanisms built around the guarantee that all participating members receive and apply committed transaction in the same order, so you must also set –slave-preserve-commit-order=1 to ensure that the final commit of parallel transactions is in the same order as the original transactions. Finally, in order to determine which transactions can be executed in parallel, the relay log must contain transaction parent information generated with –slave-parallel-type=LOGICAL_CLOCK.
- **Install Python** : The provisioning scripts that MySQL Shell uses to configure servers for use in InnoDB cluster require access to Python version 2.7. For a sandbox deployment Python is required on the single machine used for the deployment, production deployments require Python on each server instance.

Server Requirements :

To start using InnoDB cluster you need to install the following:

- MySQL Server 5.7.17 or higher
- MySQL Shell 1.0.8 or higher
- MySQL Router 2.1.2 or higher
- Python 2.7.x

Setting up Server :

Ubuntu :

sudo

To properly configure the hosts for InnoDB cluster we need sudo to execute commands with super-user privileges.

To install it run the following commands

```
$ su
$ apt-get install sudo
```

To configure it, open the sudoers file and add your user and set the required permissions:

```
$ sudo nano /etc/sudoers
$ sudo apt-get install python
```

To configure the host mapping, edit the hosts file:

```
$ sudo nano /etc/hosts
The file should have the following entrances:

192.168.1.145 prodserver11
192.168.1.146 prodserver22
192.168.1.147 prodserver44
192.168.1.148 prodserver55
```

Add the IP(s) of your host(s) and the name(s). Press ctrl+o and then enter to save the file. Press ctrl+x to close the file.

Note: Ubuntu will configure a loopback interface (127.0.1.1) for the hostname by default. Make sure to remove the loopback interface entry as it can't be used to connect from other hosts.

Install the MySQL APT repository

Open a terminal and use wget to download the official APT repository and then install the package:

```
$ sudo wget http://dev.mysql.com/get/mysql-apt-
config_0.8.4-1_all.deb
$ sudo dpkg -i ./mysql-apt-config_0.8.4-1_all.deb
```

Install MySQL Server and MySQL Shell

```
$ sudo apt-get install mysql-server mysql-shell
```

Configure the local instance calling the following function, and type the password for the user when prompted:

```
mysql-js> dba.configureLocalInstance();
```

MySQL Shell will find the default configuration file and ask you if it is ok to modify it, type “Y”. Since root cannot do remote logins, you have three options to continue with the configuration: enable the remote connections for root, create a new user or not enable remote connections for root neither create a new user.

In this tutorial, we choose to create a new user.

You will see a report with the changes made by MySQL Shell and a message saying that you need to restart the MySQL service to apply them.

```
$ sudo systemctl restart mysql.service
```

Redhat 7/CentOS 7:

Perform the host mapping as below :

```
vi /etc/hosts

192.168.1.145 prodserver11
192.168.1.146 prodserver22
192.168.1.147 prodserver44
192.168.1.148 prodserver55
```

Configure each machine to map the IP of each other machine to a hostname.

```
$ sudo yum -y install python

$ sudo yum install -y mysql-shell
```

Configure the report_host variable in the MySQL configuration of each instance.

Deploying Sandbox Instances

The first step is to deploy sandbox MySQL Server instances, so that we can play around and experiment safely, without affecting any existing MySQL databases.

Start MySQL Shell (as your ordinary, non-root OS user):

```
$ mysqlsh

mysql-js> dba.deployLocalInstance(3310)
```

The argument to deployLocalInstance() is the TCP port number where MySQL Server should listen for connections. By default, the sandbox is created in a directory named $HOME/mysql-sandboxes/<port>.

You will be prompted to pick a MySQL root account password for that instance, this is the password that you will use to connect to that instance in the future.

Note: use the same password for all your sandboxes in the same cluster.

Repeat the above command two more times, using different port numbers each time. This will allow us to create an InnoDB cluster that is tolerant to up to one failure.

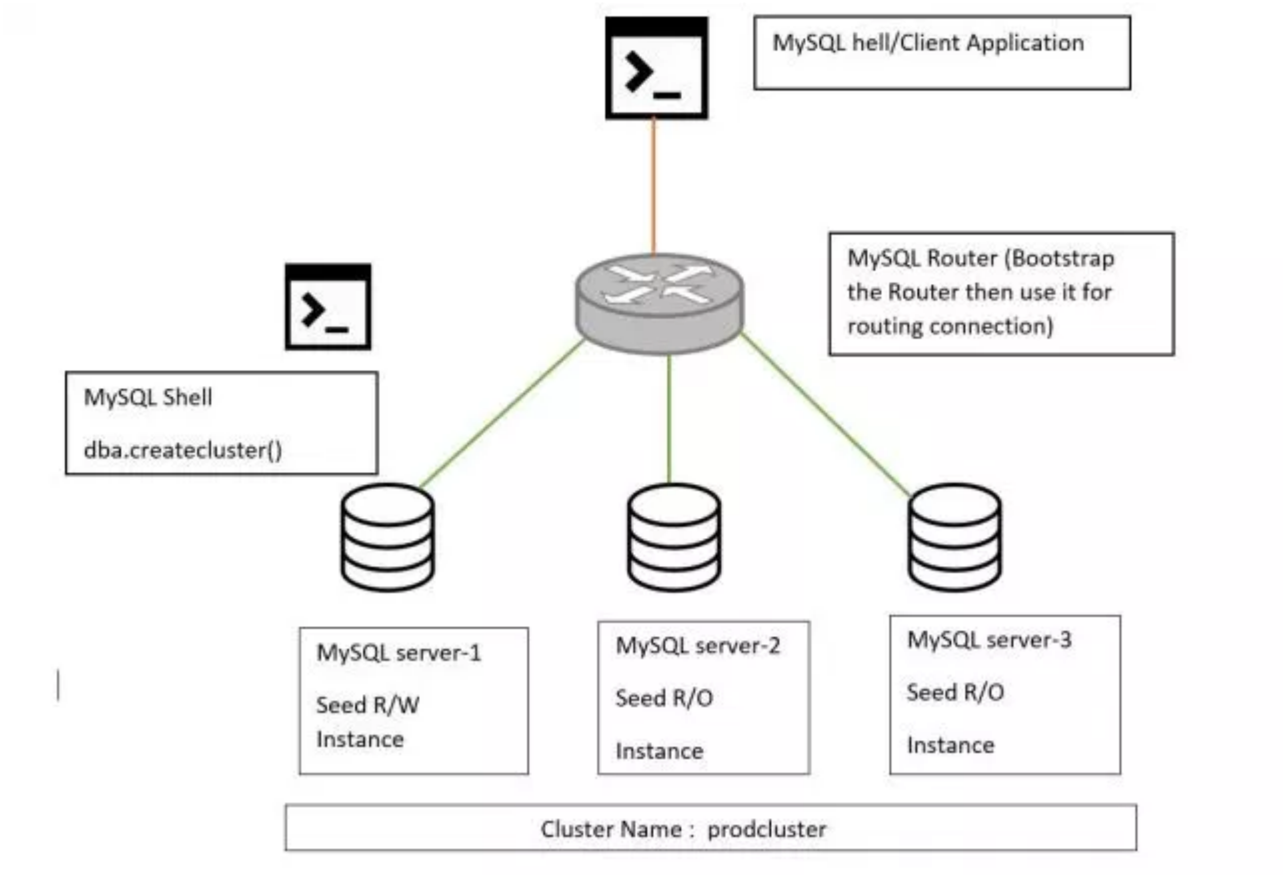
```
$ mysqlsh
Welcome to MySQL Shell 1.0.5-labs Development Preview

Copyright (c) 2016, Oracle and/or its affiliates. All rights
reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type '\help', '\h' or '\?' for help, type '\quit' or '\q' to
exit.

Currently in JavaScript mode. Use \sql to switch to SQL mode and
execute queries.
mysql-js> dba.deployLocalInstance(3310)
A new MySQL sandbox instance will be created on this host in
/home/kojima/mysql-sandboxes/3310

Please enter a MySQL root password for the new instance:
Deploying new MySQL instance...

Instance localhost:3310 successfully deployed and started.
Use '\connect root@localhost:3310' to connect to the instance.

mysql-js> dba.deployLocalInstance(3320)
A new MySQL sandbox instance will be created on this host in
/home/kojima/mysql-sandboxes/3320

Please enter a MySQL root password for the new instance:
Deploying new MySQL instance...

Instance localhost:3320 successfully deployed and started.
Use '\connect root@localhost:3320' to connect to the instance.

mysql-js> dba.deployLocalInstance(3330)
A new MySQL sandbox instance will be created on this host in
/home/kojima/mysql-sandboxes/3330

Please enter a MySQL root password for the new instance:
Deploying new MySQL instance...

Instance localhost:3330 successfully deployed and started.
Use '\connect root@localhost:3330' to connect to the instance.
```

Production Deployment of InnoDB Cluster

In this tutorial we will deploy three production clusters and one router

Setup as below :

```
 prodserver55  (MySQL server 1)
 prodserver11  (MySQL server 2)
 prodserver22  (MySQL server 3)
 prodserver44  (MySQL MySQL Router )
```


MySQL hell/Client Application
MySQL Router (Bootstrap the Router then use it for routing connection)
MySQL Shell
dba.createcluster()
MySQL server-1
Seed R/W Instance
MySQL server-2
Seed R/O Instance
MySQL server-3
Seed R/O Instance
Cluster Name : prodcluster

Prerequisites :

Three Mysql 5.7 instance nodes up and running each on prodserver55, prodserver11 and prodserver22

It is recommended to disable SELinux Policy

Firewall Ports :

Open following firewall Ports:

```
$ firewall-cmd --zone=public --add-port=3306/tcp --permanent
$ firewall-cmd --zone=public --add-port=3306/udp --permanent
$ firewall-cmd --zone=public --add-port=6446/tcp --permanent
$ firewall-cmd --zone=public --add-port=6446/udp --permanent
$ firewall-cmd --zone=public --add-port=6447/tcp --permanent
$ firewall-cmd --zone=public --add-port=6447/udp --permanent
$ firewall-cmd --zone=public --add-port=64460/tcp --permanent
$ firewall-cmd --zone=public --add-port=64460/udp --permanent
$ firewall-cmd --zone=public --add-port=64470/tcp --permanent
$ firewall-cmd --zone=public --add-port=64470/udp --permanent

$ firewall-cmd --reload

$ systemctl restart firewalld

$ firewall-cmd --list-ports
```

Create User :

The user account used to administer an instance does not have to be the root account, however the user needs to be assigned full read and write privileges on the InnoDB cluster metadata tables in addition to full MySQL administrator privileges (SUPER, GRANT OPTION, CREATE, DROP and so on).

```
CREATE USER innob_user@'%' IDENTIFIED BY 'user_Password' ;

GRANT ALL PRIVILEGES ON mysql_innodb_cluster_metadata.* TO
innob_user@'%' WITH GRANT OPTION;
GRANT RELOAD, SHUTDOWN, PROCESS, FILE, SUPER, REPLICATION SLAVE,
REPLICATION CLIENT, \
CREATE USER ON *.* TO innob_user@'%' WITH GRANT OPTION;
GRANT SELECT ON *.* TO innob_user@'%' WITH GRANT OPTION;
```

If only read operations are needed (such as for monitoring purposes), an account with more restricted privileges can be used.

```
GRANT SELECT ON mysql_innodb_cluster_metadata.* TO
innob_user@'%';
GRANT SELECT ON performance_schema.global_status TO
innob_user@'%';
GRANT SELECT ON
performance_schema.replication_applier_configuration TO
innob_user@'%';
GRANT SELECT ON performance_schema.replication_applier_status TO
innob_user@'%';
GRANT SELECT ON
performance_schema.replication_applier_status_by_coordinator TO
innob_user@'%';
GRANT SELECT ON
performance_schema.replication_applier_status_by_worker TO
innob_user@'%';
GRANT SELECT ON
performance_schema.replication_connection_configuration TO
innob_user@'%';
GRANT SELECT ON performance_schema.replication_connection_status
TO innob_user@'%';
GRANT SELECT ON performance_schema.replication_group_member_stats
TO innob_user@'%';
GRANT SELECT ON performance_schema.replication_group_members TO
innob_user@'%';
```

Start MySQL Shell

Verbose Logging :

When working with a production deployment it can be useful to configureverbose logging for MySQL Shell

```
shell> mysqlsh --log-level=DEBUG3
```

In addition to enabling the MySQL Shell log level, you can configure the amount of output Admin API provides in MySQL Shell after each call to the API.

```
mysql-js> dba.verbose=2
```

2 adds debug output to the verbose output providing full information about what each call to Admin API executes.

Checking Instance Configuration

Before creating a production deployment from server instances you need to check that MySQL on each instance is correctly configured by using the *dba.checkInstanceConfiguration()* function.

First, we will check the configuration of one of our MySQL server. Some changes are required, we will perform them using the Shell and we will restart mysqld:

```
# mysqlsh
mysql-js>
dba.checkInstanceConfiguration('innob_user@prodserver55:3306')
```

The restart_required field in the final part of the report tells you whether MySQL on the instance requires a restart to detect any change made to the configuration file.

Configuring the Instance

AdminAPI provides the *dba.configureLocalInstance()* function that finds the MySQL server's option file and modifies it to ensure that the instance is correctly configured for InnoDB cluster. Alternatively make the changes to the instance's option file manually based on the information in the report.

```
mysql-js>
dba.configureLocalInstance('innob_user@prodserver55:3306')
```

Now MySQL has all the required mandatory settings to run Group Replication.We can verify the configuration again in the Shell with *dba.checkInstanceConfiguration()* function.

Restart the mysqld service

```
$ systemctl restart mysqld
```

Repeat this process for each server instance that you added to the cluster. Similarly if you modify the cluster structure, for example changing the number of instances, you need to repeat this process for each server instance to update the InnoDB cluster metadata accordingly for each instance in the cluster.

MySQL InnoDB Cluster Creation

Once you have prepared your instances, use the *dba.createCluster()* function to create the cluster. The machine which you are running MySQL Shell on is used as the seed instance for the cluster. The seed instance is replicated to the other instances which you add to the cluster, making them replicas of the seed instance. Log in to the instance and run MySQL Shell locally.

To persist the InnoDB cluster metadata for all instances, log in to each instance that you added to the cluster and run MySQL Shell locally.

```
# mysqlsh

mysql-js> var i1='innob_user@prodserver55:3306'
mysql-js> var i2='innob_user@prodserver11:3306'
mysql-js> var i3='innob_user@prodserver22:3306'

mysql-js> shell.connect(i1)
```

MySQL Shell must be connected to an instance before you can create a cluster because when you issue dba.createCluster(name) MySQL Shell creates a MySQL protocol session to the server instance connected to the MySQL Shell's current global session.

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

Note :

If you encounter an error related to metadata being inaccessible you might have the loop-back network interface configured. For correct InnoDB cluster usage disable the loop-back interface.

The *createCluster()* command takes one parameter, which is a symbolic name you give to this InnoDB cluster. This will, among other things:

- Deploy the metadata schema in that instance (mysql.mysql_innodb_cluster_metadata)
- Verify that its configuration is correct for Group Replication, making changesif necessary
- Register it as the seed instance of the new cluster
- Create necessary internal administrative accounts
- Start Group Replication

To check the cluster has been created, use the clusterinstance's *status()* function.

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

```
mysql-js> cluster.status()
{
    "clusterName": "prodcluster",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "prodserver55:3306",
        "status": "OK_NO_TOLERANCE",
        "statusText": "Cluster is NOT tolerant to any failures.",
        "topology": {
            "prodserver55:3306": {
                "address": "prodserver55:3306",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            }
        }
    }
}
```

Once server instances belong to a cluster it is important to only administer them using MySQL Shell and AdminAPI. Attempting to manually change the configuration of Group Replication on an instance once it has been added to a cluster is not supported.

Similarly, modifying server variables critical to InnoDB cluster, such as server_uuid after an instance is configured using AdminAPI is not supported.

We can now validate that the dataset on the other instances is correct (no extra transactions executed). This is done by validating the GTIDs.

Now validate the remaining two cluster nodes

```
mysql-js> cluster.checkInstanceState(i2)
Please provide the password for 'innob_user@prodserver11:3306':
Analyzing the instance replication state...

The instance 'prodserver11:3306' is valid for the cluster.
The instance is new to Group Replication.

{
    "reason": "new",
    "state": "ok"
}



mysql-js> cluster.checkInstanceState(i3)
Please provide the password for 'innob_user@prodserver22:3306':
Analyzing the instance replication state...

The instance 'prodserver22:3306' is valid for the cluster.
The instance is new to Group Replication.

{
    "reason": "new",
    "state": "ok"
}
```

Use the *cluster.addInstance(instance)* function to add more instances to the cluster, where instance is a URI type string to connect to the local instance.

```
mysql-js> cluster.addInstance(i2)
```

To verify the instance has been added, use the cluster instance’s status() function.

```
mysql-js> cluster.status()
{
    "clusterName": "prodcluster",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "prodserver55:3306",
        "status": "OK_NO_TOLERANCE",
        "statusText": "Cluster is NOT tolerant to any failures.",
        "topology": {
            "prodserver55:3306": {
                "address": "prodserver55:3306",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            },
            "prodserver55:3406": {
                "address": "prodserver11:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            }
        }
    }
}
```

Note : If you have SELinux enabled this step may fail, because you may need to run additional commands to add a policy and enabled this.

The following command is an example of the usage of the ipWhiteList and memberSslMode options to add a new instance to a cluster:

```
mysql-js> cluster.addInstance('inno_b@prodserver55:3306', {
memberSslMode: 'REQUIRED',
ipWhitelist:'192.168.1.145/32,192.168.1.145/32'});
```

Remember to set the same memberSslMode value set in the cluster creation ifit's not 'AUTO', when you try to add a new instance to the cluster.

Add the third node to cluster

```
mysql-js> cluster.addInstance(i3)
```

```
mysql-js> cluster.status()
{
    "clusterName": "prodcluster",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "prodserver55:3306",
        "status": "OK",
        "statusText": "Cluster is ONLINE and can tolerate up to
ONE failure.",
        "topology": {
            "prodserver55:3306": {
                "address": "prodserver55:3306",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            },
            "prodserver11:3306": {
                "address": "prodserver11:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            },
            "prodserver22:3306": {
                "address": "prodserver22:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            }
        }
    }
}
```

This command queries the current status of the InnoDB cluster and produces a short report. The status field for each instance should show either ONLINE or RECOVERING. RECOVERING means that the instance is receiving updates from the seed instance and should eventually switch to ONLINE.

Another point to note is that one of the instances (the PRIMARY) is marked R/W (read/writable), while the other two are marked R/O (read only). Only the instance marked R/W can execute transactions that update the database. If that instance becomes unreachable for any reason (like a system crash), one of the remaining two instances automatically takes over its place and becomes the new PRIMARY.

Cluster Node States :

ONLINE: The instance is online and participating in the cluster.

OFFLINE: The instance has lost connection to the other instances.

RECOVERING: The instance is attempting to synchronize with the cluster by retrieving transactions it needs before it can become an ONLINE member.

UNREACHABLE: The instance has lost communication with the cluster.

ERROR: The instance has encountered an error during the recovery phase or while applying a transaction.

MISSING : The state of an instance which is part of the configured cluster, but is currently unavailable.

Read InnoDB Cluster Navigation here

Adopting a Group Replication Deployment

If you have an existing deployment of Group Replication and you want to use it to create a cluster, pass the *adoptFromGR* option to the *dba.createCluster()* function.

Note :

Group Replication members might contain MyISAM tables. Convert all such tables to InnoDB before adopting the group to an InnoDB cluster.

```
mysql-js> var cluster = dba.createCluster('prodCluster',
{adoptFromGR: true});
```

Deploy MySQL Router

In order for applications to handle failover, they need to be aware of the topology of the InnoDB cluster. They also need to know, at any time, which of the instances is the PRIMARY. While it’s possible for applications to implement that logic by themselves, MySQL Router can do that for you, with minimal work and no code changes in applications.

MySQL router provides you the ability to hide your network configuration behind a proxy and map the data requests to the cluster.

MySQL Router can configure itself based on the InnoDB cluster’s metadata using the –bootstrap option.

This configures MySQL Router automatically to route connections to the cluster’s server instances. Client applications connect to the ports MySQL Router provides, without any need to be aware of the InnoDB cluster topology.

Ubuntu :

```
$ sudo apt-get install mysql-router
```

Redhat/CentOS :

```
$ sudo yum install mysql-router
```

The recommended deployment of MySQL Router is on the same host as the application.You need the MASTER key of the InnoDB cluster to auto-configure MySQL Router.

Assuming MySQL Router is already installed, all we need to do is to bootstrap it with the metadata server, calling mysqlrouter with the following command line option from the system’s shell:

MySQL Router uses the included metadata cache plugin to retrieve the InnoDB cluster’s metadata, consisting of a list of server instance addresses which make up the InnoDB cluster and their role in the cluster.

```
$ sudo mysqlrouter --bootstrap innob_user@prodserver55:3306
--directory /mysqlrouter  --user=mysql
```

```
[root@prodserver44 /]# sudo mysqlrouter --bootstrap
innob_user@prodserver55:3306  --directory /mysqlrouter
--user=mysql
Please enter MySQL password for innob_user:

Bootstrapping MySQL Router instance at /mysqlrouter...
MySQL Router  has now been configured for the InnoDB cluster
'prodcluster'.

The following connection information can be used to connect to
the cluster.

Classic MySQL protocol connections to cluster 'prodcluster':
- Read/Write Connections: localhost:6446
- Read/Only Connections: localhost:6447

X protocol connections to cluster 'prodcluster':
- Read/Write Connections: localhost:64460
- Read/Only Connections: localhost:64470
```

MySQL Router connects to the InnoDB cluster, fetches its metadata and configures itself for use. The generated configuration creates 2 TCP ports: one for read-write sessions (which redirect connections to the PRIMARY) and one for read-only sessions (which redirect connections to one of the SECONDARY instances, in a round-robin fashion).

The MySQL Router bootstrap process creates a mysqlrouter.conf file, with the settings based on the cluster metadata retrieved from the address passed to the –bootstrap option.

Based on the InnoDB cluster metadata retrieved, MySQL Router automatically creates a configuration file, including a metadata_cache section with bootstrap_server_addresses containing the addresses for all server instances in the cluster.

```
[root@prodserver44 /]# cd mysqlrouter/
[root@prodserver44 mysqlrouter]# ls
data  log  mysqlrouter.conf  mysqlrouter.key  run  start.sh
stop.sh

[root@prodserver44 mysqlrouter]# cat mysqlrouter.conf
# File automatically generated during MySQL Router bootstrap
[DEFAULT]
user=mysql
logging_folder=/mysqlrouter/log
runtime_folder=/mysqlrouter/run
data_folder=/mysqlrouter/data
keyring_path=/mysqlrouter/data/keyring
master_key_path=/mysqlrouter/mysqlrouter.key

[logger]
level = INFO

[metadata_cache:prodcluster]
router_id=1
bootstrap_server_addresses=mysql://prodserver55:3306,mysql:
//prodserver11:3306,mysql://prodserver22:3306
user=mysql_router1_aebolhe5ougc
metadata_cluster=prodcluster
ttl=300

[routing:prodcluster_default_rw]
bind_address=0.0.0.0
bind_port=6446
destinations=metadata-cache://prodcluster/default?role=PRIMARY
mode=read-write
protocol=classic

[routing:prodcluster_default_ro]
bind_address=0.0.0.0
bind_port=6447
destinations=metadata-cache://prodcluster/default?role=SECONDARY
mode=read-only
protocol=classic

[routing:prodcluster_default_x_rw]
bind_address=0.0.0.0
bind_port=64460
destinations=metadata-cache://prodcluster/default?role=PRIMARY
mode=read-write
protocol=x
```

```
[routing:prodcluster_default_x_ro]
bind_address=0.0.0.0
bind_port=64470
destinations=metadata-cache://prodcluster/default?role=SECONDARY
mode=read-only
protocol=x
```

Note :

When you change the topology of a cluster by adding another server instance after you have bootstrapped MySQL Router, you need to update bootstrap_server_addresses based on the updated metadata. Either restart MySQL Router using the –bootstrap option, or manually edit the bootstrap_server_addresses section of the mysqlrouter.conf file and restart MySQL Router.

The generated MySQL Router configuration creates TCP ports which you use to connect to the cluster. Ports for communicating with the cluster using both Classic MySQL protocol and X Protocol are created. To use X Protocol the server instances must have X Plugin installed and configured. For a sandbox deployment, instances have X Plugin set up automatically.

For a production deployment, if you want to use X Protocol you need to install and configure X Plugin on each instance, see Section 19.3, “Setting Up MySQL as a Document Store”.

The default available ports are :

Ports	**Description**
6446	For Classic MySQL protocol read-write sessions, which MySQL Router redirects incoming connections to primary server instances.
6447	For Classic MySQL protocol read-only sessions, which MySQL Router redirects incoming connections to one of the secondary server instances.
64460	For X Protocol read-write sessions, which MySQL Router redirects incoming connections to primary server instances.
64470	For X Protocol read-only sessions, which MySQL Router redirects incoming connections to one of the secondary server instances.

When using a single-primary cluster, read-write sessions are redirected to the single primary, with a multi-primary cluster read-write sessions are redirected to one of the primary instances. For incoming read-only connections MySQL Router redirects connections to one of the secondary instances in a round-robin fashion. Once bootstrapped and configured, start MySQL Router:

```
/mysqlrouter/start.sh


default :

shell> mysqlrouter &


# /mysqlrouter/start.sh
[root@prodserver44 mysqlrouter]# PID 2869 written to /mysqlrouter
/mysqlrouter.pid


ps -ef | grep mysql

mysql     2869  2868  0 18:58 pts/0    00:00:00 /bin/mysqlrouter
-c /mysqlrouter/mysqlrouter.conf --user=mysql
root      2881  2263  0 18:58 pts/0    00:00:00 grep --color=auto
mysql
```

Tail the log

```
[root@prodserver44 log]# tail -n50 mysqlrouter.log
2018-02-12 18:58:45 INFO    [7f7e7e75c700]
[routing:prodcluster_default_x_ro] started: listening on
0.0.0.0:64470; read-only
2018-02-12 18:58:45 INFO    [7f7e7df5b700]
[routing:prodcluster_default_x_rw] started: listening on
0.0.0.0:64460; read-write
2018-02-12 18:58:45 INFO    [7f7e7ff5f700] Starting Metadata
Cache
2018-02-12 18:58:45 INFO    [7f7e7ff5f700] Connections using
ssl_mode 'PREFERRED'
2018-02-12 18:58:45 INFO    [7f7e7f75e700]
[routing:prodcluster_default_ro] started: listening on
0.0.0.0:6447; read-only
2018-02-12 18:58:45 INFO    [7f7e7ef5d700]
[routing:prodcluster_default_rw] started: listening on
0.0.0.0:6446; read-write
2018-02-12 18:58:45 INFO    [7f7e7ff5f700] Connected with
metadata server running on prodserver55:3306
2018-02-12 18:58:45 INFO    [7f7e7ff5f700] Changes detected in
cluster 'prodcluster' after metadata refresh
2018-02-12 18:58:45 INFO    [7f7e7ff5f700] Metadata for cluster
'prodcluster' has 1 replicasets:
2018-02-12 18:58:45 INFO    [7f7e7ff5f700] 'default' (3 members,
single-master)
2018-02-12 18:58:45 INFO    [7f7e7ff5f700]     prodserver55:3306
/ 33060 - role=HA mode=RW
2018-02-12 18:58:45 INFO    [7f7e7ff5f700]     prodserver11:3306
/ 34060 - role=HA mode=RO
2018-02-12 18:58:45 INFO    [7f7e7ff5f700]     prodserver22:3306
/ 35060 - role=HA mode=RO
2018-02-12 18:58:45 INFO    [7f7e6effd700] Connected with
metadata server running on prodserver55:3306
2018-02-12 19:03:46 INFO    [7f7e6effd700] Connected with
metadata server running on prodserver55:3306
```

To stop MySQL Router, in a terminal run the stop script generated:

```
$ /mysqlrouter/stop.sh
```

You can now connect a MySQL client, such as MySQL Shell to one of the incoming MySQL Router ports as described above and see how the client gets transparently connected to one of the InnoDB cluster instances.

```
[root@prodserver44 /]# mysqlsh  innob_user@prodserver55:6446
Creating a Session to 'innob_user@prodserver55:6446'
Enter password:
Your MySQL connection id is 57
Server version: 5.7.21-log MySQL Community Server (GPL)
No default schema selected; type \use <schema> to set one.
MySQL Shell 1.0.11

Copyright (c) 2016, 2017, Oracle and/or its affiliates. All
rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type '\help' or '\?' for help; '\quit' to exit.

Currently in JavaScript mode. Use \sql to switch to SQL mode and
execute queries.
mysql-js> \sql
Switching to SQL mode... Commands end with ;
mysql-sql> select @@port;
+--------+
| @@port |
+--------+
|   3306 |
+--------+
1 row in set (0.00 sec)
```

Putting the InnoDB Cluster to for High Availability

To test if high availability works, simulate an unexpected halt by killing an instance. The cluster detects the fact that the instance left the cluster and reconfigures itself. Exactly how the cluster reconfigures itself depends on whether you are using a single-primary or multi-primary cluster, and the role the instance serves within the cluster.

If the current primary leaves the cluster, one of the secondary instances iselected as the new primary, with instances prioritized by the lowest server_uuid. MySQL Router redirects read-write connections to the newly elected primary.

If a current secondary leaves the cluster, MySQL Router stops redirecting read-only connections to the instance.

There are various ways to simulate an instance leaving a cluster, for example you can forcibly stop the MySQL server on an instance, or use the AdminAPI *dba.killSandboxInstance()* if testing a sandbox deployment.

```
mysql-js> dba.killSandboxInstance(3310)
```

Login to an instance via mysql router

```
[root@prodserver44 mysqlrouter]#  mysqlsh
innob_user@prodserver55:6446
Creating a Session to 'innob_user@prodserver55:6446'
Enter password:
Your MySQL connection id is 46
Server version: 5.7.21-log MySQL Community Server (GPL)
No default schema selected; type \use <schema> to set one.
MySQL Shell 1.0.11

Copyright (c) 2016, 2017, Oracle and/or its affiliates. All
rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type '\help' or '\?' for help; '\quit' to exit.

Currently in JavaScript mode. Use \sql to switch to SQL mode and
execute queries.
mysql-js> \sql
Switching to SQL mode... Commands end with ;
mysql-sql> select @@hostname;
+-------------+
| @@hostname  |
+-------------+
| prodserver55 |
+-------------+
1 row in set (0.00 sec)
mysql-sql> select @@port;
+--------+
| @@port |
+--------+
|   3306 |
+--------+
1 row in set (0.00 sec)
mysql-sql> use testdb ;
Query OK, 0 rows affected (0.00 sec)
mysql-sql> select * from testsample;
+----+-----------+----------+------------------+----------------
---+
| id | firstname | lastname | email            | reg_date
|
+----+-----------+----------+------------------+----------------
---+
|  1 | John      | Doe      | john@example.com | 2000-02-13
1:22:23 |
| 2 | Harry     | Potter   | john@example.com| 2002-00-00
0:00:00  |
| 3 | Arav      | Paoe     | john@example.com | 2012-02-13
1:22:23 |
| 4 | Micheael  | Vio      | john@example.com | 2018-03-13
1:22:23 |
| 5 | Asibo     | Dildon   | john@example.com | 2017-02-13
1:22:23 |
| 6 | Venu      | Gopal    | john@example.com | 2018-01-13
1:22:23 |
| 7 | Umman     | Chandy   | john@example.com | 2018-07-13
1:22:23 |
+----+-----------+----------+-----------------+----------------
---+
7 rows in set (0.01 sec)
mysql-sql>
```

Here we will stop the mysql instance in prodserver55:3306

```
[root@prodserver55 mysqlrouter]# systemctl stop mysqld
[root@prodserver55 mysqlrouter]# systemctl status mysqld
● mysqld.service - MySQL Server
   Loaded: loaded (/usr/lib/systemd/system/mysqld.service;
disabled; vendor preset: disabled)
   Active: inactive (dead)
     Docs: man:mysqld(8)
           http://dev.mysql.com/doc/refman/en/using-systemd.html

Feb 16 16:48:35 prodserver55 systemd[1]: Started MySQL Server.
Feb 16 17:08:42 prodserver55 systemd[1]: Stopping MySQL Server...
Feb 16 17:08:54 prodserver55 systemd[1]: Stopped MySQL Server.
```

MySQL Router status :

```
mysql-sql> select * from testsample;
ERROR: 2013 (HY000): Lost connection to MySQL server during query
The global session got disconnected.
Attempting to reconnect to 'innob_user@prodserver55:6446'..
The global session was successfully reconnected.
mysql-sql> select @@hostname;
+--------------+
| @@hostname   |
+--------------+
| prodserver11 |
+--------------+
1 row in set (0.01 sec)
mysql-sql> select @@port;
+--------+
| @@port |
+--------+
|   3306 |
+--------+
1 row in set (0.00 sec)
mysql-sql> use testdb ;
Query OK, 0 rows affected (0.00 sec)
mysql-sql> select * from testsample;
+----+-----------+----------+------------------+----------------
---+
| id | firstname | lastname | email            | reg_date
|
+----+-----------+----------+------------------+----------------
---+
|  1 | John      | Doe      | john@example.com | 2000-02-13
1:22:23 |
|  2 | Harry     | Potter   | john@example.com| 2002-00-00
0:00:00  |
|  3 | Arav      | Paoe     | john@example.com | 2012-02-13
1:22:23 |
|  4 | Micheael  | Vio      | john@example.com | 2018-03-13
1:22:23 |
|  5 | Asibo     | Dildon   | john@example.com | 2017-02-13
1:22:23 |
|  6 | Venu      | Gopal    | john@example.com | 2018-01-13
1:22:23 |
|  7 | Umman     | Chandy   | john@example.com | 2018-07-13
1:22:23 |
+----+-----------+----------+------------------+----------------
---+
7 rows in set (0.01 sec)
```

Cluster status :

```
mysql-js> var cluster = dba.getCluster()
mysql-js> cluster.status()
{
    "clusterName": "prodcluster",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "prodserver11:3306",
        "status": "OK_NO_TOLERANCE",
        "statusText": "Cluster is NOT tolerant to any failures. 1
member is not active",
        "topology": {
            "prodserver55:3306": {
                "address": "prodserver55:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "(MISSING)"
            },
            "prodserver11:3306": {
                "address": "prodserver11:3306",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            },
            "prodserver22:3306": {
                "address": "prodserver22:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            }
        }
    }
}
```

Describing Structure of InnoDB Cluster

To get information about the structure of the InnoDB cluster itself, use the cluster.describe() function:

```
mysql-js> cluster.describe();
{
    "clusterName": "prodcluster",
    "defaultReplicaSet": {
        "instances": [
            {
                "host": "prodserver55:3306",
                "label": "prodserver55:3306",
                "role": "HA"
            },
            {
                "host": "prodserver11:3306",
                "label": "prodserver11:3306",
                "role": "HA"
            },
            {
                "host": "prodserver22:3306",
                "label": "prodserver22:3306",
                "role": "HA"
            }
        ],
        "name": "default"
    }
}
```

Removing instances from database cluster

This can be done with the *removeInstance()* method, as in the following example:

```
mysql-js> cluster.removeInstance('innob_user@prodserver55:3306')
```

How To Rejoin A Node To The Cluster?

If an instance leaves the cluster, for example because it lost connection and did not or could not automatically rejoin the cluster, it might be necessary to rejoin it to the cluster at a later stage. To rejoin an instance to a cluster issue *cluster.rejoinInstance().*

First validate the instance using *dba.validateInstance()*

```
mysql-js> dba.validateInstance('innob_user@prodserver55:3306')
```

Once the instance is valid for cluster usage issue below command

```
mysql-js> cluster.rejoinInstance('innob_user@prodserver55:3306');
```

Then connect to the instance, run MySQL Shell locally and issue *dba.configureLocalInstance()*. This ensures the InnoDB cluster configuration is persisted to the instance's option file to enable it to rejoin the cluster automatically.

Restoring a Cluster from Quorum Loss

If a instance (or instances) fail, then a cluster can lose its quorum, which is the ability to vote in a new primary. In this case you can re-establish quorum using the method *cluster.forceQuorumUsingPartitionOf()*, as shown in the following MySQL Shell example:

```
mysql-js> var cluster =  dba.getCluster("prodcluster")
  // The cluster lost its quorum and its status shows
  // "status": "NO_QUORUM"


mysql-js>
cluster.forceQuorumUsingPartitionOf("innob_user@prodserver55:3306
")
```

Rebooting a Cluster from a Major Outage

If your cluster suffers from a complete outage, you can ensure it is reconfigured correctly using *dba.rebootClusterFromCompleteOutage()*. In the event that a cluster has completely stopped, the instances must be started and only then can the cluster be started.

Make sure that all the three instances are up and running

```
mysql> SELECT * FROM
performance_schema.replication_group_members;
+---------------------------+-----------------------------------
--+--------------+-------------+--------------+
| CHANNEL_NAME              | MEMBER_ID
| MEMBER_HOST  | MEMBER_PORT | MEMBER_STATE |
+---------------------------+-----------------------------------
--+--------------+-------------+--------------+
| group_replication_applier | 516d2ea6-0d94-11e8-825b-
000c2958682d | prodserver11 |       3306 | OFFLINE      |
| group_replication_applier | d4786d8c-0d9e-11e8-
b10c-000c2958682d | prodserver22 |       3306 | OFFLINE      |
| group_replication_applier | da2d0b92-0d8a-11e8-
bc4c-000c2958682d | prodserver55 |       3306 | OFFLINE      |
+---------------------------+-----------------------------------
--+--------------+-------------+--------------+
3 rows in set (0.00 sec)
```

```
mysql-js> var i1='innob_user@prodserver55:3306'
mysql-js>  shell.connect(i1)
mysql-js> dba.rebootClusterFromCompleteOutage('prodcluster')
```

```
mysql> SELECT * FROM
performance_schema.replication_group_members;
+---------------------------+-----------------------------------
--+--------------+-------------+--------------+
| CHANNEL_NAME              | MEMBER_ID
| MEMBER_HOST  | MEMBER_PORT | MEMBER_STATE |
+---------------------------+-----------------------------------
--+--------------+-------------+--------------+
| group_replication_applier | 516d2ea6-0d94-11e8-825b-
000c2958682d | prodserver11 |       3306 | ONLINE       |
| group_replication_applier | d4786d8c-0d9e-11e8-
b10c-000c2958682d | prodserver22 |       3306 | ONLINE       |
| group_replication_applier | da2d0b92-0d8a-11e8-
bc4c-000c2958682d | prodserver55 |       3306 | ONLINE       |
+---------------------------+-----------------------------------
--+--------------+-------------+--------------+
3 rows in set (0.00 sec)
```

Also verify the cluster status using *status()* command

```
mysql-js> var cluster = dba.getCluster()
mysql-js> cluster.status()
```

Rescanning a Cluster

If changes to an instance's configuration are made without using AdminAPI, you need to rescan the cluster to update the InnoDB cluster metadata.

Rescan the cluster with *cluster.rescan()* to update the InnoDB cluster metadata.

Checking Instance State

The *cluster.checkInstanceState()* function can be used to to verify the existing data on an instance does not prevent it from joining a cluster.

```
mysql-js>
cluster.checkInstanceState('innob_user@prodserver55:3406')


Please provide the password for 'innob_user@prodserver55:3406':
Analyzing the instance replication state...

The instance 'prodserver55:3406' is valid for the cluster.
The instance is fully recoverable.

{
    "reason": "recoverable",
    "state": "ok"
}
```

The output of this function can be one of the following:

OK new: the instance has not executed any GTID transactions, therefore it cannot conflict with the GTIDs executed by the cluster.

OK recoverable: the instance has executed GTIDs which do not conflict with the executed GTIDs of the cluster seed instances.

ERROR diverged: the instance has executed GTIDs which diverge with the executed GTIDs of the cluster seed instances.

ERROR lost_transactions: the instance has more executed GTIDs than the executed GTIDs of the cluster seed instances.

Dissolving an InnoDB Cluster

To dissolve an InnoDB cluster you connect to a read-write instance, for example the primary in a single-primary cluster, and use the *Cluster.dissolve()* command.

```
mysql-js> var cluster = dba.getCluster()
mysql-js> cluster.dissolve({force:true})
```

Sample my.cnf

```
[mysqld]
#
# Remove leading # and set to the amount of RAM for the most
important data
# cache in MySQL. Start at 70% of total RAM for dedicated server,
else 10%.
# innodb_buffer_pool_size = 128M
#
# Remove leading # to turn on a very important data integrity
option: logging
# changes to the binary log between backups.
# log_bin
#
# Remove leading # to set options mainly useful for reporting
servers.
# The server defaults are faster for transactions and fast
SELECTs.
# Adjust sizes as needed, experiment to find the optimal values.
# join_buffer_size = 128M
# sort_buffer_size = 2M
# read_rnd_buffer_size = 2M
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
default-time-zone=SYSTEM
server_id=101
innodb_file_per_table=1
log-bin=mysql-binlog
relay-log=relay-log
binlog_format= ROW
bind-address = 0.0.0.0
slow_query_log=ON
long_query_time=0
innodb_monitor_enable=all
performance_schema=ON
report-host=prodserver55
report-port = 3306
log-slave-updates=1
gtid-mode=ON
enforce-gtid-consistency=ON
master-info-repository=TABLE
relay-log-info-repository=TABLE
transaction-write-set-extraction=XXHASH64
slave-parallel-workers=3
slave-preserve-commit-order=1
slave-parallel-type=LOGICAL_CLOCK
binlog_checksum=NONE
# Disabling symbolic-links is recommended to prevent assorted
```

```
security risks
symbolic-links=0
port=3306
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
log_timestamps= SYSTEM
disabled_storage_engines = MyISAM,BLACKHOLE,FEDERATED,CSV,ARCHIVE
group_replication_start_on_boot = ON
group_replication = ON
group_replication_allow_local_disjoint_gtids_join = OFF
group_replication_allow_local_lower_version_join = OFF
group_replication_auto_increment_increment = 7
group_replication_bootstrap_group = OFF
group_replication_components_stop_timeout = 31536000
group_replication_compression_threshold = 1000000
group_replication_enforce_update_everywhere_checks = OFF
group_replication_flow_control_applier_threshold = 25000
group_replication_flow_control_certifier_threshold = 25000
group_replication_flow_control_mode = QUOTA
group_replication_force_members
group_replication_group_name = d0187a28-0dc8-11e8-
aa4c-000c2958682d
group_replication_group_seeds =
prodserver55:13406,prodserver55:13506
group_replication_gtid_assignment_block_size = 1000000
group_replication_ip_whitelist = AUTOMATIC
group_replication_local_address = prodserver55:13306
group_replication_member_weight = 50
group_replication_poll_spin_loops = 0
group_replication_recovery_complete_at = TRANSACTIONS_APPLIED
group_replication_recovery_reconnect_interval = 60
group_replication_recovery_retry_count = 10
group_replication_recovery_ssl_ca
group_replication_recovery_ssl_capath
group_replication_recovery_ssl_cert
group_replication_recovery_ssl_cipher
group_replication_recovery_ssl_crl
group_replication_recovery_ssl_crlpath
group_replication_recovery_ssl_key
group_replication_recovery_ssl_verify_server_cert = OFF
group_replication_recovery_use_ssl = ON
group_replication_single_primary_mode = ON
group_replication_ssl_mode = REQUIRED
group_replication_transaction_size_limit = 0
group_replication_unreachable_majority_timeout = 0
auto_increment_increment = 1
auto_increment_offset = 2
[client]
```

```
  socket=/var/lib/mysql/mysql.sock
  port=3306
  protocol=TCP
```

Group Replication Flow Control

When using MySQL Group Replication, it's possible that some members are lagging behind the group. Due to load, hardware limitation, etc… This lag can become problematic to keep good certification behavior regarding performance and keep the possible certification failure as low as possible.

Every member of the Group send some statistics about its queues (applier queue and certification queue) to the other members. Then every node decide to slow down or not if they realize that one node reached the threshold for one of the queue:

group_replication_flow_control_applier_threshold (default is 25000)

group_replication_flow_control_certifier_threshold (default is 25000)

When group_replication_flow_control_mode is set to QUOTA on the node seeing that one of the other members of the cluster is lagging behind (threshold reached), it will throttle the write operations to the the minimum quota. This quota is calculated based on the number of transactions applied in the last second, and then it is reduced below that by subtracting the "over the quota" messages from the last period.

Handling Slowness due to Flow Control

In the performance_schema.replication_group_member_stats table, you have the mount of transaction in the apply queue (COUNT_TRANSACTIONS_REMOTE_IN_APPLIER_QUEUE):

We can use the below Query

```
mysql> SELECT MEMBER_HOST, COUNT_TRANSACTIONS_IN_QUEUE
TRX_LOCAL_Q,
              COUNT_TRANSACTIONS_REMOTE_IN_APPLIER_QUEUE
TRX_APPLY_Q
       FROM performance_schema.replication_group_member_stats t1
       JOIN performance_schema.replication_group_members t2
         ON t2.MEMBER_ID=t1.MEMBER_ID;
+-------------+-------------+-------------+
| MEMBER_HOST | TRX_LOCAL_Q | TRX_APPLY_Q |
+-------------+-------------+-------------+
| mysql1      |           0 |           0 |
| mysql3      |           0 |        8415 |
| mysql2      |           0 |           0 |
+-------------+-------------+-------------+
```

Also find the SYS Schema view from lefred blog

```
select * from sys.gr_member_routing_candidate_status;
+------------------+-----------+---------------------+----------
----------+
| viable_candidate | read_only | transactions_behind |
transactions_to_cert |
+------------------+-----------+---------------------+----------
----------+
| YES              | NO        |                   0 |
0 |
+------------------+-----------+---------------------+----------
----------+
```

You can download the file also from here

When the incident is too long and flow controls starts to kick in, all the cluster will start slowing down… if this is an incident expected to be long to solve or a maintenance, the first thing we would like to do is to stop sending MySQL traffic to the node (reads and/or writes in case of Multi-Primary cluster). Then we want that the specific node stops sending its statistics to the other nodes.

We can stop sending its statistics related to the flow control between the nodes :

```
mysql> set global group_replication_flow_control_mode='DISABLED';
```

Now that this node won't trigger any flow control anymore, the cluster will runas its optimal speed and this provides you extra time to fix the problem or finish the maintenance.

As soon as the maintenance is finished and the queue recovered, you can set it back to ‘QUOTA’.

Nover 10, 2018 Uncat

Published by dinfratechsource

Welcome to dinfratechsource – A blog that contains a lot of useful information for Linux, Unix,Database administrators, newbies and everybody who is interested in these operating systems,Cloud Administration,Bigdata & Open Source Database in general. This is one stop for all the infrastructure solutions. View all posts by dinfratechsource

UP ↑