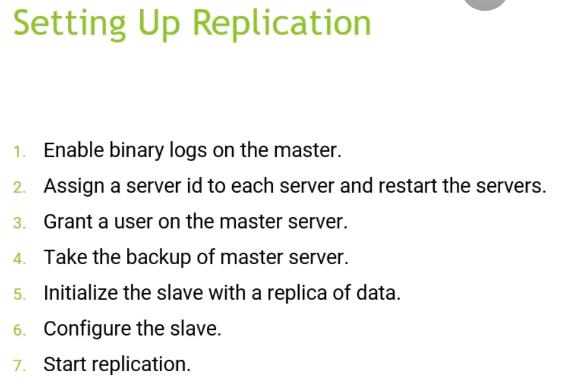
Mysql Replication

1. Check if binary logs enabled on master and server id is uniq and not 0.

show global variables like ‘%log\_bin%’;

show global variables like ‘%server\_id%’;

show global variables like ‘%expire%’;

show global variables like ‘%binlog%’;

if it’s not enabled, we need to enable it using mysql config file -- /etc/my.cnf

update the cnf to --

[mysqld]

server-id = 1

expire\_logs\_days = 10

binlog\_format = row

log\_bin = /var/log/mysql/mysql-bin

2. if reqd, create log\_bin file path, and give mysql as the owner to make update to it

mkdir /var/log/mysql

chown -R mysql.mysql /var/log/mysql

3. Update the same in slave server, with different server id. log\_bin is not reqd in slave for this topology.

4. Restart both servers. -- systemctl restart mysql

5. Create user for slave to read from bin log in master.

create user ‘slave-user’@’%’ identified by ‘[Password@1234](mailto:Password@1234)’;

6. Grant privileges to this user for replication

grant replication slave on \*.\* to ‘slave-user’@’%’;

grant replication client on \*.\* to ‘slave-user’@’%’;

7. Run Command – flush privileges;

8. Load dummy data to master mysql

9. Take backup of the master mysql before starting the replication to be fed to slave to come in sync

mysqldump --all-databases --master-data=2 --single-transaction -u root -p > dump.sql

10. Transfer the dump.sql to slave server using nc

11. Restore the backup to slave server

mysql -u root -p < dump.sql

12. Check first 70 lines in dump.sql to get master-log-file and master-log-pos to be updated in slave for replication

head -70 dump.sql

look for line -- CHANGE MASTER TO MASTER\_LOG\_FILE='mysql-bin.000002', MASTER\_LOG\_POS=66374800;

13. Login to slave mysql and start replication by running command

change master to master\_host=’<ip-addr>’, master\_user=’<slave-user>’, master\_password=’<slave-user-password>’, MASTER\_LOG\_FILE='mysql-bin.000002', MASTER\_LOG\_POS=66374800;

start slave;

14. check slave status

show slave status\G;

15. check and make slave read only

show global variables like ‘%read\_only%’;

set global read\_only=1;

set global super\_read\_only=1;

Load Dummy Data to Mysql

1. Download the dummy data --

wget 'https://codeload.github.com/datacharmer/test\_db/zip/master' -O master.zip

2. yum install unzip -y

3. unzip master.zip

4. cd test\_db-master/

5. mysql -u root -p < employees.sql

Transfer file from one server to another using netcat

1. Run Command to both server ---

yum install nc

2. Start nc to listen to any incoming request in reciever machine --

nc -l 9999 > dump.sql

3. Send the file from sender machine --

cat dump.sql | nc mysql2 9999

**GTID Based Replication**

1. Update the /etc/my.cnf file to config mysql server to work on gtid protocol.

[mysqld]

server-id = 1

expire\_logs\_days = 10

binlog\_format = row

log\_bin = /var/log/mysql/mysql-bin

gtid\_mode=on

enforce\_gtid\_consistency

2. Restart mysql -- systemctl restart mysqld

<to test, load dummy data to master node, you can check the result in gtid\_executed table>

3. Create Slave user and grant replication slave to it.

create user 'slave-user'@'%' identified with mysql\_native\_password by ['Password@1234](mailto:'Password@1234)';

grant replication slave on \*.\* to 'slave-user'@'%';

4. Take backup of master node and import it to slave node

mysqldump --all-databases --master-data=2 --single-transaction --triggers --routines --events -u root -p > dump.sql

scp dump.sql mysql2:/home

On slave --

mysql -uroot -p < /home/dump.sql

5. Check if gtid\_purged like variable is at the same position as of gtid\_executed in slave node.

6. Run Command and start slave --

change master to master\_host='mysql1', master\_user='slave\_user', master\_password='Password@1234', master\_auto\_position=1;

start slave;

7. Check slave status --

show slave status\G;

**InnoDB ReplicaSet**

Setup --

we will have 3 servers, with 2 mysql sevrer and 1 mysql router sevrer

mysql1 → install mysql server and mysql shell

mysql2 → install mysql server and mysql shell

mysql3 → install mysql shell and mysql router

Intsall Mysql Server →

1. Get the yum rpm package comapatible to os

curl -sSLO <https://dev.mysql.com/get/mysql80-community-release-el7-7.noarch.rpm>

2. Install RPM

sudo rpm -ivh mysql80-community-release-el7-7.noarch.rpm

3. Install Mysql Server

sudo yum install mysql-server

4. Install Mysql Shell

sudo yum install mysql-shell

5. Install Mysql Router

sudo yum install mysql-router-community

6. Login to myslq1 server using mysql shell

mysqlsh -uri [root@localhost](mailto:root@localhost)

7. Configure Replicaset Instance in master

a. Run command to configure replica set instance

dba.configureReplicaSetInstance()

b. for master select option 2 to create admin account for the replica set

(2) Create a new admin account for InnoDB ReplicaSet with minimal required grants)

c. Enter Account Details --

Account Name – InnodbReplicaSet

Account Password -- [Password@1234](mailto:Password@1234)

d. Select Y for the setup to change mysql configuration

8. Do the same setup in all the mysql servers --

9. In master-

1. connect with replica set account name

Run Command -- \c InnodbReplicaSet@mysql1

2. Now we can create replica set. Run command

rs = dba.createReplicaSet(‘MySqlReplicaSet’)

3. Add other instances to replicaset. Run Comamnd

rs.addInstance(‘InnodbReplicaSet@mysql1:3306’)

4. You can check the status of the replica set

rs.status()

10. ReplicaSet is configured, now we will setup mysql router

1. Run Command – mysqlrouter --bootstrap [InnodbReplicaSet@mysql1](mailto:InnodbReplicaSet@mysql1) --user mysqlrouter

This will bootstrap the mysql router in replicaset and add itself with user mysqlrouter

2. Start the mysql router – systemctl start mysqlrouter

**Semi Synchronus Replication**

Implementing semi sync replication with 1 master and 1 slave-user. Async replication is already running.

1. Check in all the mysql nodes if plugins can be installed dynamically.

Select @@have\_dynamic\_loading;

2. Check if semi sync plugins already installed.

Show plugins;

3. We need to install master semi sync plugin and slave semi sync plugins

install plugin rpl\_semi\_sync\_master soname ‘semisync\_master.so’; (install)

set global rpl\_semi\_sync\_master\_enabled=1; (enable)

select @@rpl\_semi\_sync\_master\_timeout; (check if timeout is correctly set, after this timeout, it would be changed to semi sync replication)

install plugin rpl\_semi\_sync\_slave soname ‘semisync\_slave.so’; (install)

set global rpl\_semi\_sync\_slave\_enabled=1; (enable)

4. To apply these changes, we need to stop and start i/o thread in slave-user

stop slave io\_thread;

start slave io\_thread;

Semi Sync Replication is running now. To stop semi sync and go back to async replication --

set global rpl\_semi\_sync\_master\_enabled=0;

set global rpl\_semi\_sync\_slave\_enabled=0;

stop slave io\_thread;

start slave io\_thread;

**Multi-Threaded Slaves**

Setup is having mysql replication with 1 master and 1 slave.

1. To start MTS we need to first stop the slave and configure for MTS and then start back

stop slave;

2. set parallel worker count for slave

set global slave\_parallel\_workers = 8;

3. set parallel type

set global slave\_parallel\_type = “LOGICAL\_CLOCK”;

4. set preserver commit order

set global slave\_preserve\_commit\_order = “ON”;

5. set max queue length for a worker

set global slave\_pending\_jobs\_size\_max = 1073741824;

6. start slave

start slave;

After this,we can see multi threads trying to syncup master. But still lag for slave is increasing. We can control the speed of writes in master to improve the performance of slave replication

1. set global binlog\_group\_commit\_sync\_delay = 10000;

2. set global binlog\_group\_commit\_sync\_no\_delay\_count = 50;

To stop the MTS Run Below Command --

1. To start MTS we need to first stop the slave and configure for MTS and then start back

stop slave;

2. set parallel worker count for slave

set global slave\_parallel\_workers = 0;

3. set parallel type

set global slave\_parallel\_type = “Database”;

4. set preserver commit order

set global slave\_preserve\_commit\_order = “OFF”;

5. start slave

start slave;

Group Replication With Single Mode

Setup → 3 mysql server, contribute to mysql group replication  
192.168.70.10 mysql1

192.168.70.20 mysql2

192.168.70.30 mysql3

> Configure an Instance for Group Replication

update /etc/my.cnf

# General replication settings

disabled\_storage\_engines="MyISAM,BLACKHOLE,FEDERATED,ARCHIVE,MEMORY"

gtid\_mode = ON

enforce\_gtid\_consistency = ON

master\_info\_repository = TABLE

relay\_log\_info\_repository = TABLE

binlog\_checksum = NONE

log\_slave\_updates = ON

log\_bin = binlog

binlog\_format = ROW

transaction\_write\_set\_extraction = XXHASH64

loose-group\_replication\_bootstrap\_group = OFF

loose-group\_replication\_start\_on\_boot = OFF

loose-group\_replication\_ssl\_mode = REQUIRED

loose-group\_replication\_recovery\_use\_ssl = 1

# Shared replication group configuration

loose-group\_replication\_group\_name = "a3357a67-357e-4fc9-8f6d-7362fbabac61"

loose-group\_replication\_ip\_whitelist = "192.168.70.10,192.168.70.20,192.168.70.30"

loose-group\_replication\_group\_seeds = "192.168.70.10:33061,192.168.70.20:33061,192.168.70.30:33061"

# Host specific replication configuration

server\_id = 1

bind-address = "192.168.70.10"

report\_host = "192.168.70.10"

loose-group\_replication\_local\_address = "192.168.70.10:33061"

# Single or Multi-primary mode? Uncomment these two lines

# for multi-primary mode, where any host can accept writes

#loose-group\_replication\_single\_primary\_mode = OFF

#loose-group\_replication\_enforce\_update\_everywhere\_checks = ON

Run Below command for all the mysql servers to create user for group replication --

SET SQL\_LOG\_BIN=0;

CREATE USER 'repl'@'%' IDENTIFIED BY 'Password@1234' REQUIRE SSL;

GRANT REPLICATION SLAVE ON \*.\* TO 'repl'@'%';

FLUSH PRIVILEGES;

SET SQL\_LOG\_BIN=1;

CHANGE REPLICATION SOURCE TO SOURCE\_USER='repl', SOURCE\_PASSWORD='Password@1234' FOR CHANNEL 'group\_replication\_recovery';

INSTALL PLUGIN group\_replication SONAME 'group\_replication.so';

SHOW PLUGINS;

Run Below command for the first mysql instance in the group --

SET GLOBAL group\_replication\_bootstrap\_group=ON;

START GROUP\_REPLICATION;

SET GLOBAL group\_replication\_bootstrap\_group=OFF;

SELECT \* FROM performance\_schema.replication\_group\_members;

Run Below command for other mysql servers --

START GROUP\_REPLICATION;

SELECT \* FROM performance\_schema.replication\_group\_members;

For auto join the group, update /etc/my.cnf

loose-group\_replication\_start\_on\_boot = ON

and restart mysql server

--------------------------------------------------------------------------------------------------

1. Config the GTID on all mysql server for group replication

disabled\_storage\_engines="MyISAM,BLACKHOLE,FEDERATED,ARCHIVE,MEMORY"

server\_id=1

gtid\_mode=ON

enforce\_gtid\_consistency=ON

binlog\_checksum=NONE

binlog\_checksum setting disables checksums for events written to the binary log, which default to being enabled. From MySQL 8.0.21, Group Replication supports the presence of checksums in the binary log and can use them to verify the integrity of events on some channels, so you can use the default setting.

2. Set Config for GTID based replication for older mysql versions – else default is enough

log\_bin=binlog

log\_slave\_updates=ON

binlog\_format=ROW

master\_info\_repository=TABLE

relay\_log\_info\_repository=TABLE

transaction\_write\_set\_extraction=XXHASH64

3. Set Group Replication Settings

plugin\_load\_add='group\_replication.so'

group\_replication\_group\_name="0011b45a-69c8-49d4-a911-a5ba2e16706c"

group\_replication\_start\_on\_boot=off

group\_replication\_local\_address= "192.168.70.10:33061"

group\_replication\_group\_seeds= "192.168.70.10:33061,192.168.70.20:33061,192.168.70.30:33061"

group\_replication\_bootstrap\_group=off

a. plugin-load-add adds the Group Replication plugin to the list of plugins which the server loads at startup.

b. Configuring [group\_replication\_group\_name](https://dev.mysql.com/doc/refman/8.0/en/group-replication-system-variables.html" \l "sysvar_group_replication_group_name) tells the plugin that the group that it is joining, or creating, is named "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa". The value of [group\_replication\_group\_name](https://dev.mysql.com/doc/refman/8.0/en/group-replication-system-variables.html" \l "sysvar_group_replication_group_name) must be a valid UUID.

c. Configuring the [group\_replication\_start\_on\_boot](https://dev.mysql.com/doc/refman/8.0/en/group-replication-system-variables.html" \l "sysvar_group_replication_start_on_boot) variable to off instructs the plugin to not start operations automatically when the server starts. This is important when setting up Group Replication as it ensures you can configure the server before manually starting the plugin. Once the member is configured you can set [group\_replication\_start\_on\_boot](https://dev.mysql.com/doc/refman/8.0/en/group-replication-system-variables.html" \l "sysvar_group_replication_start_on_boot) to on so that Group Replication starts automatically upon server boot.

d. Configuring [group\_replication\_local\_address](https://dev.mysql.com/doc/refman/8.0/en/group-replication-system-variables.html" \l "sysvar_group_replication_local_address) sets the network address and port which the member uses for internal communication with other members in the group. Group Replication uses this address for internal member-to-member connections involving remote instances of the group communication engine (XCom, a Paxos variant).

e. Configuring [group\_replication\_group\_seeds](https://dev.mysql.com/doc/refman/8.0/en/group-replication-system-variables.html" \l "sysvar_group_replication_group_seeds) sets the hostname and port of the group members which are used by the new member to establish its connection to the group. These members are called the seed members. Once the connection is established, the group membership information is listed in the Performance Schema table [replication\_group\_members](https://dev.mysql.com/doc/refman/8.0/en/performance-schema-replication-group-members-table.html).

The server that starts the group does not make use of this option, since it is the initial server and as such, it is in charge of bootstrapping the group. In other words, any existing data which is on the server bootstrapping the group is what is used as the data for the next joining member. The second server joining asks the one and only member in the group to join, any missing data on the second server is replicated from the donor data on the bootstrapping member, and then the group expands. The third server joining can ask any of these two to join, data is synchronized to the new member, and then the group expands again. Subsequent servers repeat this procedure when joining. A joining member must communicate with a seed member using the same protocol (IPv4 or IPv6) that the seed member advertises in the [group\_replication\_group\_seeds](https://dev.mysql.com/doc/refman/8.0/en/group-replication-system-variables.html" \l "sysvar_group_replication_group_seeds) option.

f. Configuring [group\_replication\_bootstrap\_group](https://dev.mysql.com/doc/refman/8.0/en/group-replication-system-variables.html" \l "sysvar_group_replication_bootstrap_group) instructs the plugin whether to bootstrap the group or not. In this case, even though s1 is the first member of the group we set this variable to off in the option file. Instead we configure [group\_replication\_bootstrap\_group](https://dev.mysql.com/doc/refman/8.0/en/group-replication-system-variables.html" \l "sysvar_group_replication_bootstrap_group) when the instance is running, to ensure that only one member actually bootstraps the group.

> User Credentials for Distributed Recovery

Group Replication uses a distributed recovery process to synchronize group members when joining them to the group. Distributed recovery involves transferring transactions from a donor's binary log to a joining member using a replication channel named group\_replication\_recovery. You must therefore set up a replication user with the correct permissions so that Group Replication can establish direct member-to-member replication channels. If group members have been set up to support the use of a remote cloning operation as part of distributed recovery, which is available from MySQL 8.0.17, this replication user is also used as the clone user on the donor, and requires the correct permissions for this role too.

The same replication user must be used for distributed recovery on every group member. The process of creating the replication user for distributed recovery can be captured in the binary log, and then you can rely on distributed recovery to replicate the statements used to create the user. Alternatively, you can disable binary logging before creating the replication user, and then create the user manually on each member, for example if you want to avoid the changes being propagated to other server instances. If you do this, ensure you re-enable binary logging once you have configured the user.

To create the replication user for distributed recovery, follow these steps:

1. If you want to disable binary logging in order to create the replication user separately on each instance, do so by issuing the following statement:

SET SQL\_LOG\_BIN=0;

2. Create a MySQL user with the following privileges

CREATE USER *rpl\_user*@'%' IDENTIFIED BY '*Password@1234*';

GRANT REPLICATION SLAVE ON \*.\* TO *rpl\_user*@'%';

GRANT CONNECTION\_ADMIN ON \*.\* TO *rpl\_user*@'%';

GRANT BACKUP\_ADMIN ON \*.\* TO *rpl\_user*@'%';

GRANT GROUP\_REPLICATION\_STREAM ON \*.\* TO *rpl\_user*@'%';

FLUSH PRIVILEGES;

3. If you disabled binary logging, enable it again as soon as you have created the user,

SET SQL\_LOG\_BIN=1;

[REPLICATION SLAVE](https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html#priv_replication-slave), which is required for making a distributed recovery connection to a donor to retrieve data.

[CONNECTION\_ADMIN](https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html#priv_connection-admin), which ensures that Group Replication connections are not terminated if one of the servers involved is placed in offline mode.

[BACKUP\_ADMIN](https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html#priv_backup-admin), if the servers in the replication group are set up to support cloning. This privilege is required for a member to act as the donor in a cloning operation for distributed recovery.

[GROUP\_REPLICATION\_STREAM](https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html#priv_group-replication-stream), if the MySQL communication stack is in use for the replication group. This privilege is required for the user account to be able to establish and maintain connections for Group Replication using the MySQL communication stack.

4. When you have created the replication user, you must supply the user credentials to the server for use with distributed recovery. You can do this by setting the user credentials as the credentials for the group\_replication\_recovery channel

CHANGE REPLICATION SOURCE TO SOURCE\_USER='*rpl\_user*', SOURCE\_PASSWORD='*Password@1234*' FOR CHANNEL 'group\_replication\_recovery';

User credentials specified on [START GROUP\_REPLICATION](https://dev.mysql.com/doc/refman/8.0/en/start-group-replication.html) are saved in memory only, and are removed by a [STOP GROUP\_REPLICATION](https://dev.mysql.com/doc/refman/8.0/en/stop-group-replication.html) statement or server shutdown. You must issue a [START GROUP\_REPLICATION](https://dev.mysql.com/doc/refman/8.0/en/start-group-replication.html) statement to provide the credentials again, so you cannot start Group Replication automatically with these credentials.

> Install Group Replication Plugin

It is first necessary to ensure that the Group Replication plugin is installed on server s1. If you used plugin\_load\_add='group\_replication.so' in the option file then the Group Replication plugin is already installed, and you can proceed to the next step. Otherwise, you must install the plugin manually;

INSTALL PLUGIN group\_replication SONAME 'group\_replication.so';

SHOW PLUGINS;

> Bootstrapping the Group

The process of starting a group for the first time is called bootstrapping. You use the [group\_replication\_bootstrap\_group](https://dev.mysql.com/doc/refman/8.0/en/group-replication-system-variables.html" \l "sysvar_group_replication_bootstrap_group) system variable to bootstrap a group. The bootstrap should only be done by a single server, the one that starts the group and only once. This is why the value of the [group\_replication\_bootstrap\_group](https://dev.mysql.com/doc/refman/8.0/en/group-replication-system-variables.html" \l "sysvar_group_replication_bootstrap_group) option was not stored in the instance's option file. If it is saved in the option file, upon restart the server automatically bootstraps a second group with the same name. This would result in two distinct groups with the same name

SET GLOBAL group\_replication\_bootstrap\_group=ON;

START GROUP\_REPLICATION USER='*rpl\_user*', PASSWORD='*Password@1234*';

SET GLOBAL group\_replication\_bootstrap\_group=OFF;

Once the [START GROUP\_REPLICATION](https://dev.mysql.com/doc/refman/8.0/en/start-group-replication.html) statement returns, the group has been started

SELECT \* FROM performance\_schema.replication\_group\_members;

> Test Group Replication

For the purpose of demonstrating that the server is indeed in a group and that it is able to handle load, create a table and add some content to it.

CREATE DATABASE test;

USE test;

CREATE TABLE t1 (c1 INT PRIMARY KEY, c2 TEXT NOT NULL);

INSERT INTO t1 VALUES (1, 'Luis');

Check the content of table t1 and the binary log.

SELECT \* FROM t1;

SHOW BINLOG EVENTS;

As seen above, the database and the table objects were created and their corresponding DDL statements were written to the binary log. Also, the data was inserted into the table and written to the binary log, so it can be used for distributed recovery by state transfer from a donor's binary log.

Setup ProxySql

1. install proxysql in centos -

yum update -y

yum install -y <https://github.com/sysown/proxysql/releases/download/v2.5.3/proxysql-2.5.3-1-centos7.x86_64.rpm>

2. install mysql client

yum install mysql -y

3. start proxy sql

systemctl start proxysql

systemctl status proxysql

4. login into proxysql using admin creds – password(admin)

mysql -u admin -p -h 127.0.0.1 -P 6032 --prompt='ProxySQLAdmin> '

5. update the admin password for proxysql

UPDATE global\_variables SET variable\_value='admin:Password@1234' WHERE variable\_name='admin-admin\_credentials';

6. update the admin user to runtime as well as disk

LOAD ADMIN VARIABLES TO RUNTIME;

SAVE ADMIN VARIABLES TO DISK;

7. Configure Mysql group replication servers

a. download relevant functions to be imported into group replication for proxy sql monitoring

curl -OL <https://gist.github.com/lefred/77ddbde301c72535381ae7af9f968322/raw/5e40b03333a3c148b78aa348fd2cd5b5dbb36e4d/addition_to_sys.sql>

b. import the sql script to primary node in group replication

mysql -u root -p < addition\_to\_sys.sql

c. create a user for proxy sql in group replication for monitoring

mysql -u root -p

CREATE USER 'monitor'@'%' IDENTIFIED with mysql\_native\_password BY ['Password@1234](mailto:'Password@1234)';

GRANT SELECT on sys.\* to 'monitor'@'%';

GRANT SELECT on performance\_schema.\* to 'monitor'@'%';

FLUSH PRIVILEGES;

8. Configure Monitoring in ProxySql in ProxySqlAdmin

UPDATE global\_variables SET variable\_value='monitor' WHERE variable\_name='mysql-monitor\_username';

UPDATE global\_variables SET variable\_value='Password@1234' WHERE variable\_name='mysql-monitor\_password';

9. update the runtime and disk configuration

LOAD MYSQL VARIABLES TO RUNTIME;

SAVE MYSQL VARIABLES TO DISK;

10. Add Mysql nodes to Proxysql Server Pool

INSERT INTO mysql\_group\_replication\_hostgroups (writer\_hostgroup, backup\_writer\_hostgroup, reader\_hostgroup, offline\_hostgroup, active, max\_writers, writer\_is\_also\_reader, max\_transactions\_behind) VALUES (2, 4, 3, 1, 1, 1, 0, 300);

INSERT INTO mysql\_servers(hostgroup\_id, hostname, port) VALUES (2, '192.168.70.10', 3306);

INSERT INTO mysql\_servers(hostgroup\_id, hostname, port) VALUES (3, '192.168.70.20', 3306);

INSERT INTO mysql\_servers(hostgroup\_id, hostname, port) VALUES (3, '192.168.70.30', 3306);

10. update the runtime and disk configuration

LOAD MYSQL SERVERS TO RUNTIME;

SAVE MYSQL SERVERS TO DISK;

11. Check if host groups are configured --

SELECT hostgroup\_id, hostname, status FROM runtime\_mysql\_servers;

12. Now for clients to connect, create user in group replication that will also be used by proxysql for authentication, create user in primary node of group replication

CREATE USER 'asterisk'@'%' IDENTIFIED with mysql\_native\_password BY 'Password@1234';

GRANT ALL PRIVILEGES on \*.\* to 'asterisk'@'%';

FLUSH PRIVILEGES;

13. create proxysql user with same creds for clients to connect

INSERT INTO mysql\_users(username, password, default\_hostgroup) VALUES ('asterisk', 'Password@1234', 2);

LOAD MYSQL USERS TO RUNTIME;

SAVE MYSQL USERS TO DISK;

14. To test,we can login into ProxySql Client

mysql -u asterisk -p -h 127.0.0.1 -P 6033 --prompt='ProxySQLClient> '

15. To split the read/write among different hostgroup --

INSERT INTO mysql\_query\_rules (rule\_id,active,match\_digest,destination\_hostgroup,apply) VALUES (1,1,'^SELECT.\*FOR UPDATE$',2,1);

INSERT INTO mysql\_query\_rules (rule\_id,active,match\_digest,destination\_hostgroup,apply) VALUES (2,1,'^SELECT',3,1);

LOAD MYSQL QUERY RULES TO RUNTIME;

SAVE MYSQL QUERY RULES TO DISK;

----------------------------------------------------------------------------------------------------------------------------

**InnoDB Cluster**

Setup 3 mysql servers, install mysql shell in any one server that will be admin of the cluster. Install mysql router in any 4th machine which will acts as load balancer.

1. Create Cluster user in all mysql server instances

create user 'clusteradmin'@'%' identified with mysql\_native\_password by ['Password@1234](mailto:'Password@1234)';

grant all privileges on \*.\* to 'clusteradmin'@'%' with grant option;

reset master;

2. Login to myslq1 server using mysql shell in server 1

mysqlsh -uri [root@localhost](mailto:root@localhost)

3. Switch to JS mode and check if instance configuration is correct for cluster. You will get some recommendations for cluster

\js

dba.checkInstanceConfiguration("[clusteradmin@mysql1](mailto:clusteradmin@mysql1):3306")

4. if there is an error and some recommendations are given the run configure instance to update the mysql server for cluster initiation

dba.configureInstance(['clusteradmin@mysql1](mailto:'clusteradmin@mysql1):3306')

dba.configureInstance(['clusteradmin@mysql2](mailto:'clusteradmin@mysql2):3306')

dba.configureInstance(['clusteradmin@mysql3](mailto:'clusteradmin@mysql3):3306')

5. now connect mysql shell to any one mysql server to establish cluster with mysql x adminapi

\c [clusteradmin@mysql1](mailto:clusteradmin@mysql1):3306

6. now create a cluster and store the instance to a variable cls

var cls = dba.createCluster("mycluster")

7. add other servers also to this instance cluster and choose clone as mode of replication

cls.addInstance(['clusteradmin@mysql2](mailto:'clusteradmin@mysql2):3306')

cls.addInstance(['clusteradmin@mysql3](mailto:'clusteradmin@mysql3):3306')

8. after this,all mysql servers are connected in a cluster mode. You can check the cluster status and metadata using

cls.status()

cls.describe()

9. bootstrap the mysql router to connect to primary node

mysqlrouter --bootstrap clusteradmin@mysql1 –user=mysqlrouter

10. Start mysql router --

systemctl start mysqlrouter

11. Test the connection from router --

mysql -u clusteradmin -P 6446 -h app -p (R/W)

mysql -u clusteradmin -P 6447 -h app -p (R)