MySQL InnoDB Cluster & ClusterSet crash course

By Mohammed Bashir
MySQL Evangelist

Introduction

Welcome to the MySQL InnoDB Cluster and ClusterSet Crash Course!

Embark on a journey to master MySQL InnoDB Cluster and ClusterSet with my comprehensive crash course PDF. This guide is crafted to provide you with insights into setting up and managing InnoDB Clusters, and to unravel the capabilities of ClusterSet for seamless database operations across multiple clusters.

My crash course includes real-world use cases, allowing you to apply these concepts in practical scenarios and gain hands-on experience. Whether you're a seasoned database administrator or new to MySQL, this resource is tailored to cater to all levels of expertise.

Explore the depths of InnoDB Cluster and ClusterSet to enhance your database infrastructure, ensuring high availability, scalability, and fault tolerance. Join us in this exploration of MySQL's powerful features, and let's unlock the full potential of InnoDB Cluster and ClusterSet together.

Note: This course is based on my personal experience and is not affiliated with official channels. For official courses, you may refer to Oracle University.

By Mohammed Abdallah Bashir
Senior DBA, MySQL Evangelist
Member of MySQL Customer Advisory Board
moabdalla.googlepages.com
moabdalla@gmail.com
+966598863344

Contents

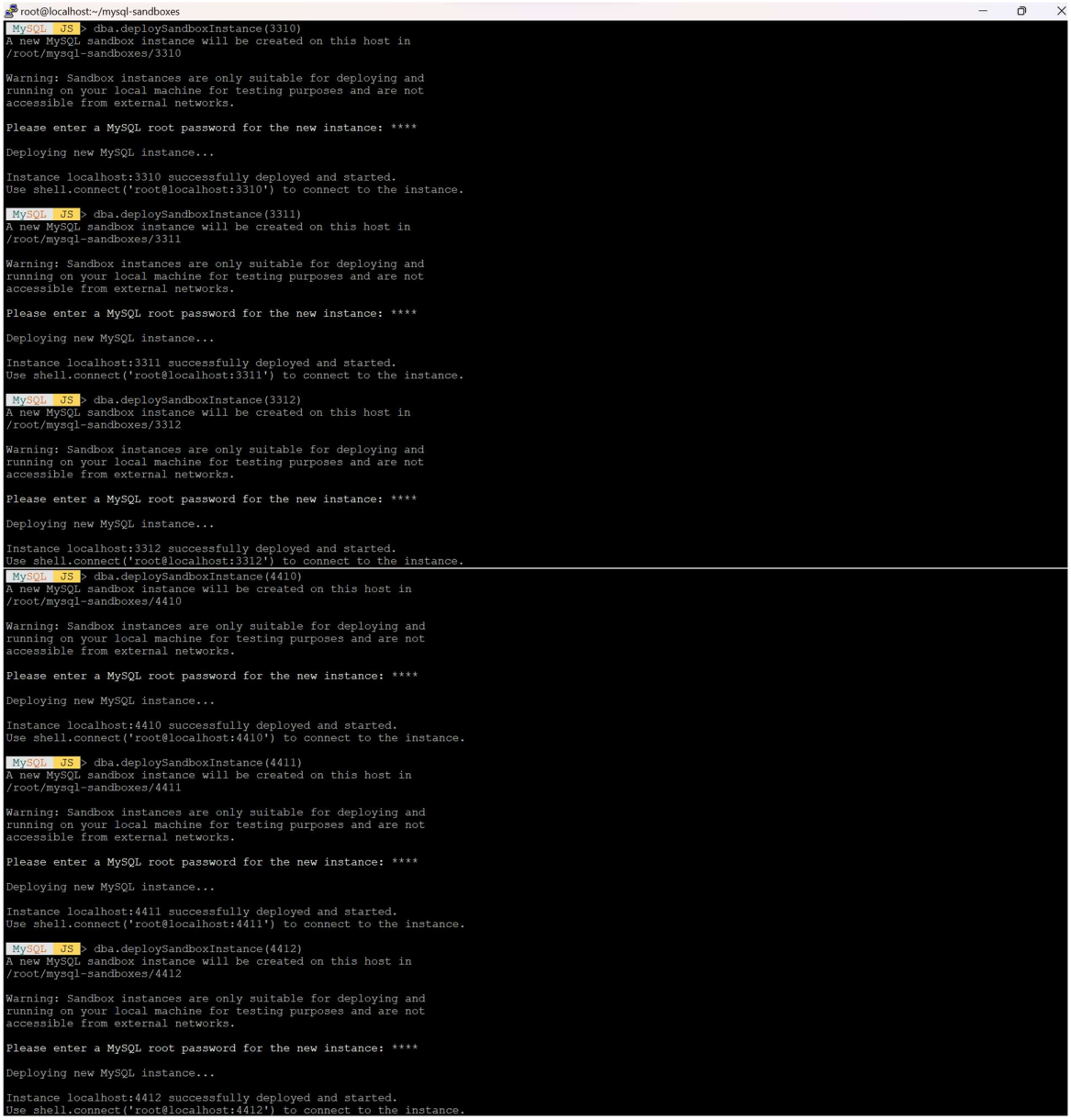
Install MySQL community 8.0.32

The Hands-on Labs below are designed for CentOS or Red Hat 7. If you intend to switch the operating system, please take into account the variations in OS commands and package downloads. Exercise caution during copy and paste operations; it is advisable to manually input commands.

sudo -i	Switch to root
yum update	Update Linux with last patches to clear variabilities
mkdir pkg	Create a folder for the database RPM
cd pkg/	Switch to the folder
wget 'https://cdn.mysql.com/archives/mysql-8.0/mysql-8.0.32-1.el7.x86_64.rpm-bundle.tar'	Download the RPM packages
wget 'https://cdn.mysql.com/archives/mysql-shell/mysql-shell-8.0.32-1.el7.x86_64.rpm'	Download MySQL Shell
tar -xvf mysql*	Unzipping the RPM file
yum localinstall mysql-*	Installing MySQL DB, shell, router
systemctl start mysqld.service	Start mysql service
less /var/log/mysqld.log	Get the temporary root password
mysql -uroot -p	Enter the temp password
ALTER USER 'root'@'localhost' IDENTIFIED BY Test@1420';	Change the root password from temp

Create sandbox:

Mysqlsh	Login to the console
`\c root@localhost`	Password as above Test@1420
dba.deploySandboxInstance(3310)	Deploy sandboxes, during Sandbox deployment new password for each instance will be requested
dba.deploySandboxInstance(3311)	
dba.deploySandboxInstance(3312)	
dba.deploySandboxInstance(4410)	
dba.deploySandboxInstance(4420)	
dba.deploySandboxInstance(4430)	



Create InnoDB cluster:

mysqlsh	Switch to mysql shell note that it has three modes (\js, \py, \sql)
`\c root@localhost:3310`	`Connect to the first sandbox`
var ruh=dba.createCluster('ruh')	Creating Riyadh production cluster
ruh.status()	Cluster status
ruh.addInstance('root@localhost:3311')	Adding the second instance replica
ruh.addInstance('root@localhost:3312')	Adding the third instance replica
ruh.status()	Cluster status with three nodes

```
MySQL  localhost:3310 ssl  JS > var ruh=dba.createCluster('ruh')
A new InnoDB Cluster will be created on instance '127.0.0.1:3310'.

Validating instance configuration at localhost:3310...
NOTE: Instance detected as a sandbox.
Please note that sandbox instances are only suitable for deploying test clusters for use within the same host.

This instance reports its own address as 127.0.0.1:3310

Instance configuration is suitable.
NOTE: Group Replication will communicate with other members using '127.0.0.1:3310'. Use the localAddress option to override.

Creating InnoDB Cluster 'ruh' on '127.0.0.1:3310'...

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.

MySQL  localhost:3310 ssl  JS > ruh.status()
{
    "clusterName": "ruh",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "127.0.0.1:3310",
        "ssl": "REQUIRED",
        "status": "OK_NO_TOLERANCE",
        "statusText": "Cluster is NOT tolerant to any failures.",
        "topology": {
            "127.0.0.1:3310": {
                "address": "127.0.0.1:3310",
                "memberRole": "PRIMARY",
                "mode": "R/W",
                "readReplicas": {},
                "replicationLag": "applier_queue_applied",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            }
        },
        "topologyMode": "Single-Primary"
    },
    "groupInformationSourceMember": "127.0.0.1:3310"
}
MySQL  localhost:3310 ssl  JS > ruh.addInstance('root@localhost:3311')

NOTE: The target instance '127.0.0.1:3311' has not been pre-provisioned (GTID set is empty). The Shell is unable to decide whether incremental 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 '127.0.0
.1:3311' with a physical snapshot from an existing cluster member. To use this method by default, set the 'recoveryMethod' option to 'clone'.

The incremental 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): c
Validating instance configuration at localhost:3311...
NOTE: Instance detected as a sandbox.
Please note that sandbox instances are only suitable for deploying test clusters for use within the same host.

This instance reports its own address as 127.0.0.1:3311

Instance configuration is suitable.
NOTE: Group Replication will communicate with other members using '127.0.0.1:3311'. Use the localAddress option to override.

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.
Clone based state recovery is now in progress.

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: 127.0.0.1:3311 is being cloned from 127.0.0.1:3310
** Stage DROP DATA: Completed
** Clone Transfer
    FILE COPY  ############################################################  100%  Completed
    PAGE COPY  ############################################################  100%  Completed
    REDO COPY  ############################################################  100%  Completed

NOTE: 127.0.0.1:3311 is shutting down...
```

```
 MySQL  localhost:3310 ssl  JS > ruh.addInstance('root@localhost:3312')

NOTE: The target instance '127.0.0.1:3312' has not been pre-provisioned (GTID set is empty). The Shell is unable to decide whether incremental 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 '127.0.0
.1:3312' with a physical snapshot from an existing cluster member. To use this method by default, set the 'recoveryMethod' option to 'clone'.

The incremental 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): c
Validating instance configuration at localhost:3312...
NOTE: Instance detected as a sandbox.
Please note that sandbox instances are only suitable for deploying test clusters for use within the same host.

This instance reports its own address as 127.0.0.1:3312

Instance configuration is suitable.
NOTE: Group Replication will communicate with other members using '127.0.0.1:3312'. Use the localAddress option to override.

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.
Clone based state recovery is now in progress.

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: 127.0.0.1:3312 is being cloned from 127.0.0.1:3311
** Stage DROP DATA: Completed
** Clone Transfer
    FILE COPY  ############################################################  100%  Completed
    PAGE COPY  ############################################################  100%  Completed
    REDO COPY  ############################################################  100%  Completed

NOTE: 127.0.0.1:3312 is shutting down...
```

```
 MySQL  localhost:3310 ssl  JS > ruh.status()
{
    "clusterName": "ruh",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "127.0.0.1:3310",
        "ssl": "REQUIRED",
        "status": "OK",
        "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
        "topology": {
            "127.0.0.1:3310": {
                "address": "127.0.0.1:3310",
                "memberRole": "PRIMARY",
                "mode": "R/W",
                "readReplicas": {},
                "replicationLag": "applier_queue_applied",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            },
            "127.0.0.1:3311": {
                "address": "127.0.0.1:3311",
                "memberRole": "SECONDARY",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLag": "applier_queue_applied",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            },
            "127.0.0.1:3312": {
                "address": "127.0.0.1:3312",
                "memberRole": "SECONDARY",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLag": "applier_queue_applied",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            }
        },
        "topologyMode": "Single-Primary"
    },
    "groupInformationSourceMember": "127.0.0.1:3310"
```

Bootstrap MySQL router

mysqlrouter --bootstrap root@localhost:3310 --user=mysqlrouter	It's an Orchestrator to keep the application up and running, Note using '--force' when re-bootstrap. MySQL router usually deployed in application server. You can install as many router as you want according to business need

```
[root@localhost ~]# mysqlrouter --bootstrap root@localhost:3310 --user=mysqlrouter
Please enter MySQL password for root:
Error: The given Router instance is already configured for a cluster named ''.
If you'd like to replace it, please use the --force configuration option.
[root@localhost ~]# mysqlrouter --bootstrap root@localhost:3310 --user=mysqlrouter --force
Please enter MySQL password for root:
# Bootstrapping system MySQL Router instance...

- Creating account(s) (only those that are needed, if any)
- Using existing certificates from the '/var/lib/mysqlrouter' directory
- Verifying account (using it to run SQL queries that would be run by Router)
- Storing account in keyring
- Adjusting permissions of generated files
- Creating configuration /etc/mysqlrouter/mysqlrouter.conf

Existing configuration backed up to '/etc/mysqlrouter/mysqlrouter.conf.bak'

Existing dynamic state backed up to '/var/lib/mysqlrouter/state.json.bak'

# MySQL Router configured for the ClusterSet 'TestCs'

After this MySQL Router has been started with the generated configuration

    $ /etc/init.d/mysqlrouter restart
or
    $ systemctl start mysqlrouter
or
    $ mysqlrouter -c /etc/mysqlrouter/mysqlrouter.conf

ClusterSet 'TestCs' can be reached by connecting to:

## MySQL Classic protocol

- Read/Write Connections: localhost:6446
- Read/Only Connections:  localhost:6447

## MySQL X protocol

- Read/Write Connections: localhost:6448
- Read/Only Connections:  localhost:6449
```

Create ClusterSet DR:

var cs=ruh.createClusterSet('TestCs')	Deploying ClusterSet plugin
cs.status()	ClusterSet status
jed = cs.createReplicaCluster('root@localhost:4410', 'TestDr ')	Adding the replica Cluster
cs.status()	ClusterSet status
jed.status()	Replica cluster status
jed.addInstance('root@localhost:4411')	Adding the second replica on jed Cluster
jed.addInstance('root@localhost:4412')	Adding the third replica on jed Cluster
cs.status()	ClusterSet status
jed.status()	Replica cluster status

```
 MySQL  localhost:3310 ssl  JS > var cs=ruh.createClusterSet('TestCs')
A new ClusterSet will be created based on the Cluster 'ruh'.

* Validating Cluster 'ruh' for ClusterSet compliance.

* Creating InnoDB ClusterSet 'TestCs' on 'ruh'...

* Updating metadata...

ClusterSet successfully created. Use ClusterSet.createReplicaCluster() to add Replica Clusters to it.

 MySQL  localhost:3310 ssl  JS > cs.status()
{
    "clusters": {
        "ruh": {
            "clusterRole": "PRIMARY",
            "globalStatus": "OK",
            "primary": "127.0.0.1:3310"
        }
    },
    "domainName": "TestCs",
    "globalPrimaryInstance": "127.0.0.1:3310",
    "primaryCluster": "ruh",
    "status": "HEALTHY",
    "statusText": "All Clusters available."
}
```

```
 MySQL  localhost:3310 ssl  JS > jed = cs.createReplicaCluster('root@localhost:4410', 'TestDr')
Setting up replica 'TestDr' of cluster 'ruh' at instance '127.0.0.1:4410'.

A new InnoDB Cluster will be created on instance '127.0.0.1:4410'.

Validating instance configuration at localhost:4410...
NOTE: Instance detected as a sandbox.
Please note that sandbox instances are only suitable for deploying test clusters for use within the same host.

This instance reports its own address as 127.0.0.1:4410

Instance configuration is suitable.
NOTE: Group Replication will communicate with other members using '127.0.0.1:4410'. Use the localAddress option to override.


* Checking transaction state of the instance...

NOTE: The target instance '127.0.0.1:4410' has not been pre-provisioned (GTID set is empty). The Shell is unable to decide whether replication can completely
 recover its state.
The safest and most convenient way to provision a new instance is through automatic clone provisioning, which will completely overwrite the state of '127.0.0
.1:4410' with a physical snapshot from an existing clusterset member. To use this method by default, set the 'recoveryMethod' option to 'clone'.

WARNING: It should be safe to rely on replication to incrementally recover the state of the new Replica Cluster if you are sure all updates ever executed in
the ClusterSet were done with GTIDs enabled, there are no purged transactions and the instance used to create the new Replica Cluster contains the same GTID
set as the ClusterSet 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): c
Waiting for clone process of the new member to complete. Press ^C to abort the operation.
* Waiting for clone to finish...
NOTE: 127.0.0.1:4410 is being cloned from 127.0.0.1:3310
** Stage DROP DATA: Completed
** Clone Transfer
    FILE COPY  ############################################################  100%  Completed
    PAGE COPY  ############################################################  100%  Completed
    REDO COPY  ############################################################  100%  Completed
* Clone process has finished: 73.97 MB transferred in about 1 second (~73.97 MB/s)

Creating InnoDB Cluster 'TestDr' on '127.0.0.1:4410'...

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.
```

```
 MySQL  localhost:3310 ssl  JS > cs.status()
{
    "clusters": {
        "TestDr": {
            "clusterRole": "REPLICA",
            "clusterSetReplicationStatus": "OK",
            "globalStatus": "OK"
        },
        "ruh": {
            "clusterRole": "PRIMARY",
            "globalStatus": "OK",
            "primary": "127.0.0.1:3310"
        }
    },
    "domainName": "TestCs",
    "globalPrimaryInstance": "127.0.0.1:3310",
    "primaryCluster": "ruh",
    "status": "HEALTHY",
    "statusText": "All Clusters available."
}
```

```
 MySQL  localhost:3310 ssl  JS > jed.status()
{
    "clusterName": "TestDr",
    "clusterRole": "REPLICA",
    "clusterSetReplicationStatus": "OK",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "127.0.0.1:4410",
        "ssl": "REQUIRED",
        "status": "OK_NO_TOLERANCE",
        "statusText": "Cluster is NOT tolerant to any failures.",
        "topology": {
            "127.0.0.1:4410": {
                "address": "127.0.0.1:4410",
                "memberRole": "PRIMARY",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLagFromImmediateSource": "",
                "replicationLagFromOriginalSource": "",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            }
        },
        "topologyMode": "Single-Primary"
    },
    "domainName": "TestCs",
    "groupInformationSourceMember": "127.0.0.1:4410",
    "metadataServer": "127.0.0.1:3310"
```


MySQL localhost:3310 ssl JS > jed.addInstance('localhost:4411')
NOTE: A GTID set check of the MySQL instance at '127.0.0.1:4411' determined that it is missing transactions that were purged from all cluster members.
NOTE: The target instance '127.0.0.1:4411' has not been pre-provisioned (GTID set is empty). The Shell is unable to determine whether the instance has pre-ex
isting data that would be overwritten with clone based recovery.
The safest and most convenient way to provision a new instance is through automatic clone provisioning, which will completely overwrite the state of '127.0.0
.1:4411' with a physical snapshot from an existing cluster member. To use this method by default, set the 'recoveryMethod' option to 'clone'.
Please select a recovery method [C]lone/[A]bort (default Clone): c
Validating instance configuration at localhost:4411...
NOTE: Instance detected as a sandbox.
Please note that sandbox instances are only suitable for deploying test clusters for use within the same host.
This instance reports its own address as 127.0.0.1:4411
Instance configuration is suitable.
NOTE: Group Replication will communicate with other members using '127.0.0.1:4411'. Use the localAddress option to override.
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...
* Waiting for the Cluster to synchronize with the PRIMARY Cluster...
** Transactions replicated ## 100%
* Configuring ClusterSet managed replication channel...
** Changing replication source of 127.0.0.1:4411 to 127.0.0.1:3310
Monitoring recovery process of the new cluster member. Press ^C to stop monitoring and let it continue in background.
Clone based state recovery is now in progress.
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: 127.0.0.1:4411 is being cloned from 127.0.0.1:4410
** Stage DROP DATA: Completed
** Clone Transfer
FILE COPY ## 100% Completed
PAGE COPY ## 100% Completed
REDO COPY ## 100% Completed
MySQL localhost:3310 ssl JS > jed.addInstance('localhost:4412')
NOTE: A GTID set check of the MySQL instance at '127.0.0.1:4412' determined that it is missing transactions that were purged from all cluster members.
NOTE: The target instance '127.0.0.1:4412' has not been pre-provisioned (GTID set is empty). The Shell is unable to determine whether the instance has pre-ex
isting data that would be overwritten with clone based recovery.
The safest and most convenient way to provision a new instance is through automatic clone provisioning, which will completely overwrite the state of '127.0.0
.1:4412' with a physical snapshot from an existing cluster member. To use this method by default, set the 'recoveryMethod' option to 'clone'.
Please select a recovery method [C]lone/[A]bort (default Clone): c
Validating instance configuration at localhost:4412...
NOTE: Instance detected as a sandbox.
Please note that sandbox instances are only suitable for deploying test clusters for use within the same host.
This instance reports its own address as 127.0.0.1:4412
Instance configuration is suitable.
NOTE: Group Replication will communicate with other members using '127.0.0.1:4412'. Use the localAddress option to override.
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...
* Waiting for the Cluster to synchronize with the PRIMARY Cluster...
** Transactions replicated ## 100%
* Configuring ClusterSet managed replication channel...
** Changing replication source of 127.0.0.1:4412 to 127.0.0.1:3310
Monitoring recovery process of the new cluster member. Press ^C to stop monitoring and let it continue in background.
Clone based state recovery is now in progress.
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: 127.0.0.1:4412 is being cloned from 127.0.0.1:4411
** Stage DROP DATA: Completed
** Clone Transfer
FILE COPY ## 100% Completed
PAGE COPY ## 100% Completed
REDO COPY ## 100% Completed

```
MySQL  localhost:3310 ssl  JS > jed.status()
{
    "clusterName": "TestDr",
    "clusterRole": "REPLICA",
    "clusterSetReplicationStatus": "OK",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "127.0.0.1:4410",
        "ssl": "REQUIRED",
        "status": "OK",
        "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
        "topology": {
            "127.0.0.1:4410": {
                "address": "127.0.0.1:4410",
                "memberRole": "PRIMARY",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLagFromImmediateSource": "",
                "replicationLagFromOriginalSource": "",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            },
            "127.0.0.1:4411": {
                "address": "127.0.0.1:4411",
                "memberRole": "SECONDARY",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLagFromImmediateSource": "",
                "replicationLagFromOriginalSource": "",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            },
            "127.0.0.1:4412": {
                "address": "127.0.0.1:4412",
                "memberRole": "SECONDARY",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLagFromImmediateSource": "",
                "replicationLagFromOriginalSource": "",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
```

Re-Bootstrap MySQL router

After creating the ClusterSet plugin and incorporating DR replicas, bootstrap the MySQL router to update its metadata according to the new configuration.

`mysqlrouter --bootstrap root@localhost:3310 --user=mysqlrouter --force`	It's an Orchestrator to keep the application up and running via one IP, Note using '--force' when re-bootstrap. MySQL router usually deployed in application server.

```
[root@localhost ~]# mysqlrouter --bootstrap root@localhost:3310 --user=mysqlrouter
Please enter MySQL password for root:
Error: The given Router instance is already configured for a cluster named ''.
If you'd like to replace it, please use the --force configuration option.
[root@localhost ~]# mysqlrouter --bootstrap root@localhost:3310 --user=mysqlrouter --force
Please enter MySQL password for root:
# Bootstrapping system MySQL Router instance...

- Creating account(s) (only those that are needed, if any)
- Using existing certificates from the '/var/lib/mysqlrouter' directory
- Verifying account (using it to run SQL queries that would be run by Router)
- Storing account in keyring
- Adjusting permissions of generated files
- Creating configuration /etc/mysqlrouter/mysqlrouter.conf

Existing configuration backed up to '/etc/mysqlrouter/mysqlrouter.conf.bak'

Existing dynamic state backed up to '/var/lib/mysqlrouter/state.json.bak'

# MySQL Router configured for the ClusterSet 'TestCs'

After this MySQL Router has been started with the generated configuration

    $ /etc/init.d/mysqlrouter restart
or
    $ systemctl start mysqlrouter
or
    $ mysqlrouter -c /etc/mysqlrouter/mysqlrouter.conf

ClusterSet 'TestCs' can be reached by connecting to:

## MySQL Classic protocol

- Read/Write Connections: localhost:6446
- Read/Only Connections:  localhost:6447

## MySQL X protocol

- Read/Write Connections: localhost:6448
- Read/Only Connections:  localhost:6449
```

Cluster failover in the primary Cluster (Riyadh):

ruh.setPrimaryInstance('localhost:3311')	Changing the primary node for maintenance
ruh.status()	Cluster status
ruh.setPrimaryInstance('localhost:3312')	
ruh.status()	
ruh.setPrimaryInstance('localhost:3312')	
ruh.status()	

```
MySQL  localhost:3310 ssl  JS > ruh.setPrimaryInstance('localhost:3311')
Setting instance 'localhost:3311' as the primary instance of cluster 'ruh'...

Instance '127.0.0.1:3310' was switched from PRIMARY to SECONDARY.
Instance '127.0.0.1:3311' was switched from SECONDARY to PRIMARY.
Instance '127.0.0.1:3312' remains SECONDARY.

The instance 'localhost:3311' was successfully elected as primary.
```

```
MySQL  localhost:3310 ssl  JS > ruh.status()
{
    "clusterName": "ruh",
    "clusterRole": "PRIMARY",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "127.0.0.1:3311",
        "ssl": "REQUIRED",
        "status": "OK",
        "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
        "topology": {
            "127.0.0.1:3310": {
                "address": "127.0.0.1:3310",
                "memberRole": "SECONDARY",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLagFromImmediateSource": "",
                "replicationLagFromOriginalSource": "",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            },
            "127.0.0.1:3311": {
                "address": "127.0.0.1:3311",
                "memberRole": "PRIMARY",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            },
            "127.0.0.1:3312": {
                "address": "127.0.0.1:3312",
                "memberRole": "SECONDARY",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLagFromImmediateSource": "",
                "replicationLagFromOriginalSource": "",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            }
        },
        "topologyMode": "Single-Primary"
    },
    "domainName": "TestCs",
    "groupInformationSourceMember": "127.0.0.1:3311"
}
```

```
MySQL  localhost:3310 ssl  JS > ruh.setPrimaryInstance('localhost:3312')
Setting instance 'localhost:3312' as the primary instance of cluster 'ruh'...

Instance '127.0.0.1:3310' remains SECONDARY.
Instance '127.0.0.1:3311' was switched from PRIMARY to SECONDARY.
Instance '127.0.0.1:3312' was switched from SECONDARY to PRIMARY.

The instance 'localhost:3312' was successfully elected as primary.
```

```
MySQL  localhost:3310 ssl  JS > ruh.status()
{
    "clusterName": "ruh",
    "clusterRole": "PRIMARY",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "127.0.0.1:3312",
        "ssl": "REQUIRED",
        "status": "OK",
        "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
        "topology": {
            "127.0.0.1:3310": {
                "address": "127.0.0.1:3310",
                "memberRole": "SECONDARY",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLagFromImmediateSource": "",
                "replicationLagFromOriginalSource": "",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            },
            "127.0.0.1:3311": {
                "address": "127.0.0.1:3311",
                "memberRole": "SECONDARY",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLagFromImmediateSource": "",
                "replicationLagFromOriginalSource": "",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            },
            "127.0.0.1:3312": {
                "address": "127.0.0.1:3312",
                "memberRole": "PRIMARY",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            }
        },
        "topologyMode": "Single-Primary"
```

```
MySQL  localhost:3310 ssl  JS > ruh.setPrimaryInstance('localhost:3310')
Setting instance 'localhost:3310' as the primary instance of cluster 'ruh'...

Instance '127.0.0.1:3310' was switched from SECONDARY to PRIMARY.
Instance '127.0.0.1:3311' remains SECONDARY.
Instance '127.0.0.1:3312' was switched from PRIMARY to SECONDARY.

The instance 'localhost:3310' was successfully elected as primary.
```

```
MySQL  localhost:3310 ssl  JS > ruh.status()
{
    "clusterName": "ruh",
    "clusterRole": "PRIMARY",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "127.0.0.1:3310",
        "ssl": "REQUIRED",
        "status": "OK",
        "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
        "topology": {
            "127.0.0.1:3310": {
                "address": "127.0.0.1:3310",
                "memberRole": "PRIMARY",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            },
            "127.0.0.1:3311": {
                "address": "127.0.0.1:3311",
                "memberRole": "SECONDARY",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLagFromImmediateSource": "",
                "replicationLagFromOriginalSource": "",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            },
            "127.0.0.1:3312": {
                "address": "127.0.0.1:3312",
                "memberRole": "SECONDARY",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLagFromImmediateSource": "",
                "replicationLagFromOriginalSource": "",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            }
        },
        "topologyMode": "Single-Primary"
    },
    "domainName": "TestCs",
    "groupInformationSourceMember": "127.0.0.1:3310"
}
```

Cluster failover in the Secondary Cluster (Jeddah):

`jed.setPrimaryInstance('localhost:4411')`	Changing the primary node for maintainance
`jed.status()`	Cluster status
`jed.setPrimaryInstance('localhost:4412')`	
`jed.status()`	
`jed.setPrimaryInstance('localhost:4412')`	
`jed.status()`	

```
MySQL  localhost:3310 ssl  JS > jed.setPrimaryInstance('localhost:4411')
Setting instance 'localhost:4411' as the primary instance of cluster 'TestDr'...

Instance '127.0.0.1:4410' was switched from PRIMARY to SECONDARY.
Instance '127.0.0.1:4411' was switched from SECONDARY to PRIMARY.
Instance '127.0.0.1:4412' remains SECONDARY.

The instance 'localhost:4411' was successfully elected as primary.
MySQL  localhost:3310 ssl  JS > jed.status()
{
    "clusterName": "TestDr",
    "clusterRole": "REPLICA",
    "clusterSetReplicationStatus": "OK",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "127.0.0.1:4411",
        "ssl": "REQUIRED",
        "status": "OK",
        "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
        "topology": {
            "127.0.0.1:4410": {
                "address": "127.0.0.1:4410",
                "memberRole": "SECONDARY",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLagFromImmediateSource": "",
                "replicationLagFromOriginalSource": "",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            },
            "127.0.0.1:4411": {
                "address": "127.0.0.1:4411",
                "memberRole": "PRIMARY",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLagFromImmediateSource": "",
                "replicationLagFromOriginalSource": "",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            },
            "127.0.0.1:4412": {
                "address": "127.0.0.1:4412",
                "memberRole": "SECONDARY",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLagFromImmediateSource": "",
                "replicationLagFromOriginalSource": "",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            }
        },
        "topologyMode": "Single-Primary"
    },
    "domainName": "TestCs",
    "groupInformationSourceMember": "127.0.0.1:4411",
    "metadataServer": "127.0.0.1:3310"
}
MySQL  localhost:3310 ssl  JS > jed.setPrimaryInstance('localhost:4412')
Setting instance 'localhost:4412' as the primary instance of cluster 'TestDr'...

Instance '127.0.0.1:4410' remains SECONDARY.
Instance '127.0.0.1:4411' was switched from PRIMARY to SECONDARY.
Instance '127.0.0.1:4412' was switched from SECONDARY to PRIMARY.

The instance 'localhost:4412' was successfully elected as primary.
MySQL  localhost:3310 ssl  JS > jed.status()
{
    "clusterName": "TestDr",
    "clusterRole": "REPLICA",
    "clusterSetReplicationStatus": "OK",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "127.0.0.1:4412",
        "ssl": "REQUIRED",
        "status": "OK",
        "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
        "topology": {
            "127.0.0.1:4410": {
                "address": "127.0.0.1:4410",
                "memberRole": "SECONDARY",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLagFromImmediateSource": "",
                "replicationLagFromOriginalSource": "",
                "role": "HA",
```

Dr rehearsal (fail and failback):

cs.status()	ClusterSet status
`cs.setPrimaryCluster('TestDr', {dryRun:true})`	Test the failover with dryRun feature to get the result before performing the DR
`cs.setPrimaryCluster('TestDr')`	Switch the primary cluster to jed
cs.status	
`cs.setPrimaryCluster('ruh', {dryRun:true})`	Test the failback with dryRun feature to get the result before performing the failback
`cs.setPrimaryCluster('ruh')`	Switchback the primary cluster to Riyadh

```
 MySQL  localhost:3310 ssl  JS > cs.setPrimaryCluster('TestDr', {dryRun:true})
Switching the primary cluster of the clusterset to 'TestDr'
NOTE: dryRun enabled, no changes will be made
* Verifying clusterset status
** Checking cluster TestDr
  Cluster 'TestDr' is available
** Checking cluster ruh
  Cluster 'ruh' is available

** Waiting for the promoted cluster to apply pending received transactions...
* Refreshing replication account of demoted cluster
* Synchronizing transaction backlog at 127.0.0.1:4410

* Updating metadata

* Updating topology
** Changing replication source of 127.0.0.1:3311 to 127.0.0.1:4410
** Changing replication source of 127.0.0.1:3312 to 127.0.0.1:4410
** Changing replication source of 127.0.0.1:3310 to 127.0.0.1:4410
* Acquiring locks in replicaset instances
** Pre-synchronizing SECONDARIES
** Acquiring global lock at PRIMARY
** Acquiring global lock at SECONDARIES

* Synchronizing remaining transactions at promoted primary

* Updating replica clusters
Cluster 'TestDr' was promoted to PRIMARY of the clusterset. The PRIMARY instance is '127.0.0.1:4410'

dryRun finished.

 MySQL  localhost:3310 ssl  JS > cs.setPrimaryCluster('TestDr')
Switching the primary cluster of the clusterset to 'TestDr'
* Verifying clusterset status
** Checking cluster TestDr
  Cluster 'TestDr' is available
** Checking cluster ruh
  Cluster 'ruh' is available

** Waiting for the promoted cluster to apply pending received transactions...
* Refreshing replication account of demoted cluster
* Synchronizing transaction backlog at 127.0.0.1:4410
** Transactions replicated  ############################################################  100%
```

```
* Updating replica clusters
Cluster 'TestDr' was promoted to PRIMARY of the clusterset. The PRIMARY instance is '127.0.0.1:4410'

 MySQL  localhost:3310 ssl  JS > cs.setPrimaryCluster('ruh', {dryRun:true})
Switching the primary cluster of the clusterset to 'ruh'
NOTE: dryRun enabled, no changes will be made
* Verifying clusterset status
** Checking cluster TestDr
  Cluster 'TestDr' is available
** Checking cluster ruh
  Cluster 'ruh' is available

** Waiting for the promoted cluster to apply pending received transactions...
* Refreshing replication account of demoted cluster
* Synchronizing transaction backlog at 127.0.0.1:3310

* Updating metadata

* Updating topology
** Changing replication source of 127.0.0.1:4411 to 127.0.0.1:3310
** Changing replication source of 127.0.0.1:4412 to 127.0.0.1:3310
** Changing replication source of 127.0.0.1:4410 to 127.0.0.1:3310
* Acquiring locks in replicaset instances
** Pre-synchronizing SECONDARIES
** Acquiring global lock at PRIMARY
** Acquiring global lock at SECONDARIES

* Synchronizing remaining transactions at promoted primary

* Updating replica clusters
Cluster 'ruh' was promoted to PRIMARY of the clusterset. The PRIMARY instance is '127.0.0.1:3310'

dryRun finished.

 MySQL  localhost:3310 ssl  JS > cs.setPrimaryCluster('ruh')
Switching the primary cluster of the clusterset to 'ruh'
* Verifying clusterset status
** Checking cluster TestDr
  Cluster 'TestDr' is available
** Checking cluster ruh
  Cluster 'ruh' is available

** Waiting for the promoted cluster to apply pending received transactions...
* Refreshing replication account of demoted cluster
```

```
* Refreshing replication account of demoted cluster
* Synchronizing transaction backlog at 127.0.0.1:3310
** Transactions replicated  ############################################################  100%


* Updating metadata

* Updating topology
** Changing replication source of 127.0.0.1:4411 to 127.0.0.1:3310
** Changing replication source of 127.0.0.1:4412 to 127.0.0.1:3310
** Changing replication source of 127.0.0.1:4410 to 127.0.0.1:3310
* Acquiring locks in replicaset instances
** Pre-synchronizing SECONDARIES
** Acquiring global lock at PRIMARY
** Acquiring global lock at SECONDARIES

* Synchronizing remaining transactions at promoted primary
** Transactions replicated  ############################################################  100%


* Updating replica clusters
Cluster 'ruh' was promoted to PRIMARY of the clusterset. The PRIMARY instance is '127.0.0.1:3310'

 MySQL  localhost:3310 ssl  JS > cs.status()
{
    "clusters": {
        "TestDr": {
            "clusterRole": "REPLICA",
            "clusterSetReplicationStatus": "OK",
            "globalStatus": "OK"
        },
        "ruh": {
            "clusterRole": "PRIMARY",
            "globalStatus": "OK",
            "primary": "127.0.0.1:3310"
        }
    },
    "domainName": "TestCs",
    "globalPrimaryInstance": "127.0.0.1:3310",
    "primaryCluster": "ruh",
    "status": "HEALTHY",
    "statusText": "All Clusters available."
}
```

Complete outages use case:

`dba.killSandbonInstance(3312)`	Stop the sandbox
`dba.killSandbonInstance(3311)`	
`dba.killSandbonInstance(3310)`	
`ruh.status()`	Cluster is down
`dba.startSandbonInstance(3310)`	Start the sanboxes
`dba.startSandbonInstance(3311)`	
`dba.startSandbonInstance(3312)`	
`dba.rebootClusterFromCompleteOutage()`	To restore the cluster from complete loss/outage

`stop group_replication;`
`set global super_read_only=off;`
`drop database mysql_innodb_cluster_metadata;`
`reset master;`
`reset slave all;`
`\js -- after stopping all slaves go to master node for next step`
`dba.rebootClusterFromCompleteOutage();`
	In case if one node failed to join the cluster, showing that it's already part of the cluster, then do this step and add it again by: Cluster.addInstance('node ip')

cs.status()	
ruh.status()	
jed.status()	
\q	exit

```
 MySQL  localhost:3310 ssl  JS > dba.killSandboxInstance(3312)

Killing MySQL instance...

Instance localhost:3312 successfully killed.

 MySQL  localhost:3310 ssl  JS > dba.killSandboxInstance(3311)

Killing MySQL instance...

Instance localhost:3311 successfully killed.

 MySQL  localhost:3310 ssl  JS > dba.killSandboxInstance(3310)

Killing MySQL instance...

Instance localhost:3310 successfully killed.
```

```
MySQL  localhost:4410 ssl  JS > var cs=dba.getClusterSet()
MySQL  localhost:4410 ssl  JS > cs.status()
{
    "clusters": {
        "TestDr": {
            "clusterErrors": [
                "WARNING: Replication from the Primary Cluster not in expected state"
            ],
            "clusterRole": "REPLICA",
            "clusterSetReplicationStatus": "ERROR",
            "globalStatus": "NOT_OK",
            "status": "OK",
            "statusText": "Cluster is ONLINE and can tolerate up to ONE failure."
        },
        "ruh": {
            "clusterErrors": [
                "ERROR: Could not connect to any ONLINE members but there are unreachable instances that could still be ONLINE."
            ],
            "clusterRole": "PRIMARY",
            "clusterSetReplicationStatus": "UNKNOWN",
            "globalStatus": "UNKNOWN",
            "primary": null,
            "status": "UNREACHABLE",
            "statusText": "Could not connect to any ONLINE members"
        }
    },
    "domainName": "TestCs",
    "globalPrimaryInstance": null,
    "primaryCluster": "ruh",
    "status": "UNAVAILABLE",
    "statusText": "Primary Cluster is not reachable from the Shell, assuming it to be unavailable."
}
```

```
MySQL  localhost:4410 ssl  JS > dba.startSandboxInstance(3312)

Starting MySQL instance...

Instance localhost:3312 successfully started.

MySQL  localhost:4410 ssl  JS > dba.startSandboxInstance(3311)

Starting MySQL instance...

Instance localhost:3311 successfully started.

MySQL  localhost:4410 ssl  JS > dba.startSandboxInstance(3310)

Starting MySQL instance...

Instance localhost:3310 successfully started.

MySQL  localhost:4410 ssl  JS > \c root@localhost:3310
Creating a session to 'root@localhost:3310'
Fetching schema names for auto-completion... Press ^C to stop.
Closing old connection...
Your MySQL connection id is 60
Server version: 8.0.32 MySQL Community Server - GPL
No default schema selected; type \use <schema> to set one.
MySQL  localhost:3310 ssl  JS > ruh.status()
The cluster object is disconnected. Please use dba.getCluster() to obtain a fresh cluster handle. (RuntimeError)
MySQL  localhost:3310 ssl  JS > dba.getCluster('ruh')
Dba.getCluster: This function is not available through a session to an InnoDB Cluster that belongs to an InnoDB ClusterSet but is not ONLINE (MYSQLSH 51314)
MySQL  localhost:3310 ssl  JS >
```

```
MySQL  localhost:3310 ssl  JS > cs.status()
{
    "clusters": {
        "TestDr": {
            "clusterRole": "REPLICA",
            "clusterSetReplicationStatus": "OK",
            "globalStatus": "NOT_OK",
            "status": "OK",
            "statusText": "Cluster is ONLINE and can tolerate up to ONE failure."
        },
        "ruh": {
            "clusterErrors": [
                "ERROR: Cluster members are reachable but they're all OFFLINE."
            ],
            "clusterRole": "PRIMARY",
            "globalStatus": "NOT_OK",
            "primary": null,
            "status": "OFFLINE",
            "statusText": "All members of the group are OFFLINE"
        }
    },
    "domainName": "TestCs",
    "globalPrimaryInstance": null,
    "primaryCluster": "ruh",
    "status": "UNAVAILABLE",
    "statusText": "Primary Cluster is not available. ClusterSet availability may be restored by restoring the Primary Cluster or failover."
}
```

```
 MySQL  localhost:3310 ssl  JS > dba.rebootClusterFromCompleteOutage()
Restoring the Cluster 'ruh' from complete outage...

Cluster instances: '127.0.0.1:3310' (OFFLINE), '127.0.0.1:3311' (OFFLINE), '127.0.0.1:3312' (OFFLINE)
Waiting for instances to apply pending received transactions...
Validating instance configuration at localhost:3310...
NOTE: Instance detected as a sandbox.
Please note that sandbox instances are only suitable for deploying test clusters for use within the same host.

This instance reports its own address as 127.0.0.1:3310

Instance configuration is suitable.
* Waiting for seed instance to become ONLINE...
127.0.0.1:3310 was restored.
Validating instance configuration at 127.0.0.1:3311...
NOTE: Instance detected as a sandbox.
Please note that sandbox instances are only suitable for deploying test clusters for use within the same host.

This instance reports its own address as 127.0.0.1:3311

Instance configuration is suitable.
Rejoining instance '127.0.0.1:3311' to cluster 'ruh'...

Re-creating recovery account...
NOTE: User 'mysql_innodb_cluster_4248744108'@'%' already existed at instance '127.0.0.1:3310'. It will be deleted and created again with a new password.

* Waiting for the Cluster to synchronize with the PRIMARY Cluster...
** Transactions replicated  ############################################################  100%

The instance '127.0.0.1:3311' was successfully rejoined to the cluster.

Validating instance configuration at 127.0.0.1:3312...
NOTE: Instance detected as a sandbox.
Please note that sandbox instances are only suitable for deploying test clusters for use within the same host.

This instance reports its own address as 127.0.0.1:3312

Instance configuration is suitable.
Rejoining instance '127.0.0.1:3312' to cluster 'ruh'...

Re creating recovery account...
NOTE: User 'mysql_innodb_cluster_963446693'@'%' already existed at instance '127.0.0.1:3310'. It will be deleted and created again with a new password.


* Waiting for the Cluster to synchronize with the PRIMARY Cluster...
** Transactions replicated  ############################################################  100%

The instance '127.0.0.1:3312' was successfully rejoined to the cluster.

The Cluster was successfully rebooted.

<Cluster:ruh>
```

```
 MySQL  localhost:3310 ssl  JS > var ruh=dba.getCluster('ruh')
 MySQL  localhost:3310 ssl  JS > cs.status()
{
    "clusters": {
        "TestDr": {
            "clusterRole": "REPLICA",
            "clusterSetReplicationStatus": "OK",
            "globalStatus": "OK"
        },
        "ruh": {
            "clusterRole": "PRIMARY",
            "globalStatus": "OK",
            "primary": "127.0.0.1:3310"
        }
    },
    "domainName": "TestCs",
    "globalPrimaryInstance": "127.0.0.1:3310",
    "primaryCluster": "ruh",
    "status": "HEALTHY",
    "statusText": "All Clusters available."
}
 MySQL  localhost:3310 ssl  JS > ruh.status()
{
    "clusterName": "ruh",
    "clusterRole": "PRIMARY",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "127.0.0.1:3310",
        "ssl": "REQUIRED",
        "status": "OK",
        "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
        "topology": {
            "127.0.0.1:3310": {
                "address": "127.0.0.1:3310",
                "memberRole": "PRIMARY",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            },
            "127.0.0.1:3311": {
                "address": "127.0.0.1:3311",
                "memberRole": "SECONDARY",
                "memberRole": "SECONDARY",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLagFromImmediateSource": "",
                "replicationLagFromOriginalSource": "",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            },
            "127.0.0.1:3312": {
                "address": "127.0.0.1:3312",
                "memberRole": "SECONDARY",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLagFromImmediateSource": "",
                "replicationLagFromOriginalSource": "",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            }
        },
        "topologyMode": "Single-Primary"
    },
    "domainName": "TestCs",
    "groupInformationSourceMember": "127.0.0.1:3310"
}
```

Patching:

It’s always important to download the patch from oracle website, patch and updates tab

yum localinstall "mysqlrouter rpm"	Upgrade it in the application server. check if you need to bootstrap
yum localinstall "Shell rpm"	Patch the MySQL shell
mysqlsh -- util check-for-server-upgrade rootuser@dbip --target-version=8.0.version --output-format=JSON --config-path=/etc/my.cnf	
yum locainstall mysql-commercial-*	On the DB VM, starting from
yum update	Patch the VM – Always start with secondary
systemctl stop mysqld	Optional
Reboot	Reboot to apply updates

MySQL InnoDB Cluster & ClusterSet

Mohammed Abdallah Bashir

Concluding the MySQL InnoDB Cluster and ClusterSet Crash Course, I commend your dedication to mastering these essential database tools. The skills you've acquired, rooted in a neutral perspective, set the stage for continual success in optimizing database infrastructure. This course is a continuous journey—apply your knowledge, experiment, and view challenges as growth opportunities. The resource remains at your disposal for future guidance. Thank you for being part of this educational adventure, and best wishes as you leverage the capabilities of MySQL InnoDB Cluster and ClusterSet in your professional endeavors.

#state_of_dolphine

https://slack.lefred.be/
https://github.com/lefred