InnoDB Cluster User Guide

Abstract

User documentation for InnoDB cluster.

For help with using MySQL, please visit either the MySQL Forums or MySQL Mailing Lists, where you can discuss your issues with other MySQL users.

For additional documentation on MySQL products, including translations of the documentation into other languages, and downloadable versions in variety of formats, including HTML and PDF formats, see the MySQL Documentation Library.

Document generated on: 2017-02-22 (revision: 50864)

Table of Contents

Preface and Legal Notices

This is the InnoDB cluster guide.

Legal Notices

documentation and will not be responsible for any loss, costs, or damages incurred due to the use of this documentation.

The information contained in this document is for informational sharing purposes only and should be considered in your capacity as a customer advisory board member or pursuant to your beta trial agreement only. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle.

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle Master Agreement, Oracle License and Services Agreement, Oracle PartnerNetwork Agreement, Oracle distribution agreement, or other license agreement which has been executed by you and Oracle and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms:

You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how Oracle disseminates it (that is, electronically for download on a Web site with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Oracle. Oracle and/or its affiliates reserve any and all rights to this documentation not expressly granted above.

Chapter 1 MySQL InnoDB Cluster

Table of Contents

This chapter introduces InnoDB cluster, which is available from MySQL labs.

1.1 Introducing InnoDB Cluster

InnoDB cluster provides a new way to create a group of MySQL Servers providing high availability and read-out scalability. The group of servers has a single master, called the primary, which acts as the writable master. Multiple slave servers, called secondaries, are replicas of the master. In the event of a failure of the primary, a secondary is promoted to the role of primary. The secondaries are read-only replicas of the primary that can be used for read-out scalability. To achieve this the following MySQL technologies work together:

- MySQL InnoDB cluster enables you to distribute your data across a set of MySQL Server instances.
- MySQL Router 2.1 with InnoDB cluster Metadata Schema support, which caches the metadata of the InnoDB cluster and performs high availability routing to the MySQL Server instances which make up the cluster.
- MySQL Shell with the AdminAPI implementation, which enables you to create and administer an InnoDB cluster, using either JavaScript and Python.

Important

The supplied MySQL labs versions of MySQL Shell and MySQL Router differ from the release versions. This documentation covers the functionality provided by the labs versions which is not covered in the existing documentation.

For more information about the current release versions of MySQL Shell and MySQL Router see:

- MySQL Shell User Guide
- MySQL Router

For information about the AdminAPI available in the MySQL Shell 1.0.x labs release, see the JavaScript reference documentation.

For more background information see Using MySQL as a Document Store.

1.2 Installing InnoDB Cluster

Download the InnoDB cluster archive from labs.mysql.com. Uncompress the archive and install:

- MySQL Server 5.7.x with Group Replication
- MySQL Router 2.1.x
- MySQL Shell 1.0.x with AdminAPI and mysqlprovision

1.3 Getting Started with InnoDB Cluster

This section explains how to use MySQL Shell with AdminAPI to set up an InnoDB cluster and configure MySQL Router to achieve high availability. This demonstration uses sandbox instances supplied with the labs release.

Deploying Local Sandbox Instances

The first step is to deploy sandbox MySQL Server instances using MySQL Shell. A minimum of three instances are required to create an InnoDB cluster that is tolerant to the failure of one instance.

For this simple example the instances are all running on the same host machine. It is equally possible to run the instances on different host machines.

Start MySQL Shell from a command prompt by executing:

```
$ mysqlsh
```

Note

Do not run MySQL Shell as root.

When MySQL Shell has started, ensure you are in JavaScript mode by issuing `\js`, then execute:

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

The argument to `deployLocalInstance()` is the TCP port number where the MySQL Server instance listens for connections. By default the sandbox is created in a directory named `$HOME/mysql-sandboxes/port`.

The root password for the instance is prompted for.

Repeat the above command two more times using different port numbers, resulting in three MySQL Server instances.

```
mysql-js> dba.deployLocalInstance(3320)
mysql-js> dba.deployLocalInstance(3330)
```

Initializing the InnoDB Cluster

The next step is to initialize the InnoDB cluster while connected to the seed MySQL Server instance. The seed instance is the instance that you want to replicate. In this tutorial, the sandbox instances are blank instances, therefore we can choose any instance.

Connect MySQL Shell to the seed instance, in this case the one at port 3310:

```
mysql-js> \connect root@localhost:3310
```

Create the InnoDB cluster:

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

The parameter passed to the `createCluster()` function is a symbolic name given to this InnoDB cluster. The resulting InnoDB cluster is assigned to the `cluster` variable. This function deploys the metadata to the selected instance, configures it for replication and adds the instance as the seed of the new InnoDB cluster. A MASTER key is also prompted for, which is required for management of the InnoDB cluster.

Important

Do not lose the MASTER key because it is required for managing the InnoDB cluster.

After validating that the instance is properly configured, it is added to the InnoDB cluster as the seed instance and the replication subsystem is started.

The supplied sandbox instances are pre-configured to work with replication, but if you use a pre-existing instance, it is possible that some configuration options might not be set in a compatible way. The `createCluster()` command ensures that the settings are correct and if not, it changes their values. If a change requires MySQL Server to be restarted, you are prompted to restart it manually whenever is convenient.

Adding Instances to InnoDB Cluster

The next step is to add replicas to the InnoDB cluster. Any transactions that were executed by the seed instance are re-executed by each replica as they are added. To demonstrate we use the sandbox instances that we created earlier. The seed instance in this example is brand new, has little to no data and replication was enabled since it was created. Therefore there is very little that newly added instances need to catch up with.

Add the second instance to the InnoDB cluster:

```
mysql-js> cluster.addInstance('root@localhost:3320')
```

The root user's password is prompted for.

Add the third instance:

```
mysql-js> cluster.addInstance('root@localhost:3330')
```

The root user's password is prompted for.

With three instances in the InnoDB cluster, you can check its status:

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

This command queries the current status of the InnoDB cluster and produces a short report. The state of each instance shows one of `ONLINE`, `OFFLINE` or `RECOVERING`. `RECOVERING` means that the instance is receiving updates from the seed instance and should eventually switch to `ONLINE`. An instance shown as `OFFLINE` may have lost connection to the other instances.

Another useful detail shown in the report 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 an unexpected halt), one of the remaining two instances automatically takes over its place and becomes the new PRIMARY.

Deploying 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 is possible for applications to implement that logic by themselves, MySQL Router can do that for you, with minimal work.

The recommended deployment of MySQL Router is on the same host as the application. In this tutorial, everything is running on the same host, so deploy MySQL Router to the same host.

Important

You need the MASTER key of the InnoDB cluster to configure MySQL Router.

Assuming MySQL Router is already installed, the only required step is to bootstrap it with the location of the metadata server:

```
$ sudo mysqlrouter --bootstrap localhost:3310
```

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).

Once bootstrapped and configured, start MySQL Router (or set up a service for it to start automatically when the system boots):

```
$ mysqlrouter &
```

You can now connect a MySQL client, such as MySQL Shell to one of the incoming MySQL Router ports and see how the client gets transparently connected to one of the InnoDB cluster instances. To see which instance you are actually connected to, simply query the `port` status variable.

```
$ mysqlsh --uri root@localhost:6442
mysql-js> \sql
Switching to SQL mode... Commands end with ;
mysql-sql> select @@port;
+--------+
| @@port |
+--------+
|   3310 |
+--------+
1 row in set (0.00 sec)
```

Checking the InnoDB Cluster

To check the status of the InnoDB cluster at a later time, you can get a reference to the InnoDB cluster object by connecting to any of its instances. However if you want to make changes to the InnoDB cluster, you must connect to the `PRIMARY`. For information about how the InnoDB cluster is running, use the `status()` function:

```
mysql-js> cluster = dba.getCluster()
mysql-js> cluster.status()
{
    "clusterName": "dev",
    "defaultReplicaSet": {
        "status": "Cluster tolerant to up to ONE failure.",
```

```
        "topology": {
            "localhost:3310": {
                "address": "localhost:3310",
                "status": "ONLINE",
                "role": "HA",
                "mode": "R/W",
                "leaves": {
                    "localhost:3320": {
                        "address": "localhost:3320",
                        "status": "ONLINE",
                        "role": "HA",
                        "mode": "R/O",
                        "leaves": {}
                    },
                    "localhost:3330": {
                        "address": "localhost:3330",
                        "status": "ONLINE",
                        "role": "HA",
                        "mode": "R/O",
                        "leaves": {}
                    }
                }
            }
        }
    }
}
```

Information is displayed such as the InnoDB cluster name and topology, the default ReplicaSet, which instance is currently PRIMARY and so on.

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

```
mysql-js> cluster.describe();
{
    "clusterName": "dev",
    "adminType": "local",
    "defaultReplicaSet": {
        "name": "default",
        "instances": [
            {
                "name": "localhost:3310",
                "host": "localhost:3310",
                "role": "HA"
            },
            {
                "name": "localhost:3320",
                "host": "localhost:3320",
                "role": "HA"
            },
            {
                "name": "localhost:3330",
                "host": "localhost:3330",
                "role": "HA"
            }
        ]
    }
}
```

The output from this function shows the structure of the InnoDB cluster including all of its configuration information, ReplicaSets and Instances.

To test if failover really works, simulate an unexpected halt by killing the `PRIMARY` instance using the `dba.killLocalInstance()` function and check that one of the other instances takes over automatically.

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

Then you can again check which instance you are connected to. The first `SELECT` fails since the connection to the original `PRIMARY` was lost. MySQL Shell automatically reconnects for you and when you issue the command again the new port is confirmed.

```
mysql-sql> SELECT @@port;
ERROR: 2013 (HY000): Lost connection to MySQL server during query
The global session got disconnected.
Attempting to reconnect to 'root@localhost:6446'...
The global session was successfully reconnected.
mysql-sql> SELECT @@port;
+--------+
| @@port |
+--------+
|   3330 |
+--------+
1 row in set (0.00 sec)
```

This shows that the InnoDB cluster provided us with automatic failover, that MySQL Router has automatically reconnected us to the new `PRIMARY` instance, and that we have high availability.

Now we can bring the instance that we killed back online.

```
mysql-js> dba.startLocalInstance(3310)
mysql-js> cluster.rejoinInstance('root@localhost:3310')
mysql-js> cluster.status()
```