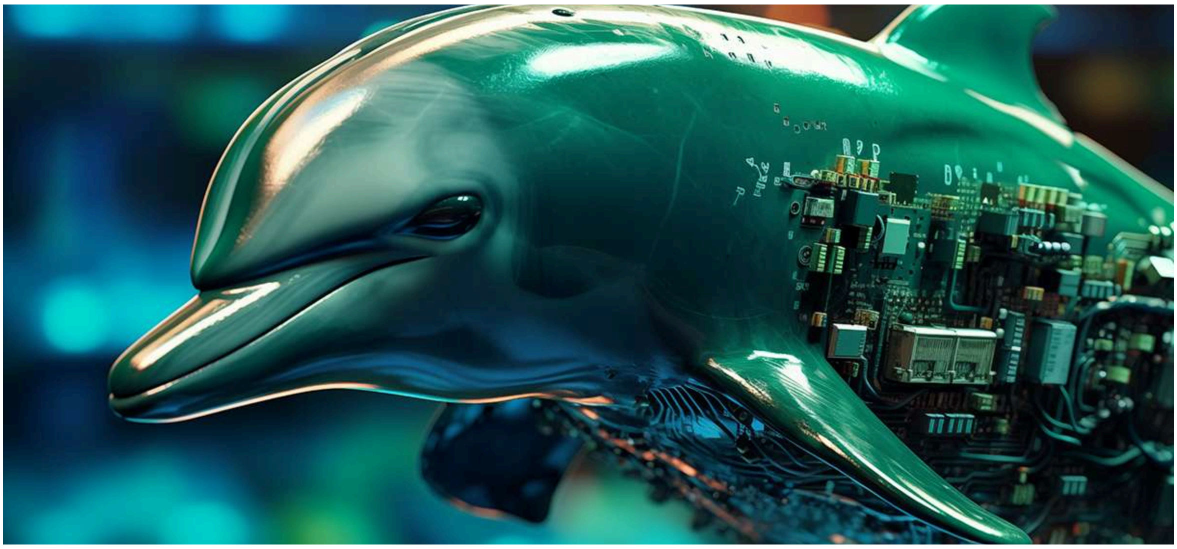


Insight for DBAs MySQL

Subscribe to RSS Feed

Orchestrator (for Managing MySQL) High Availability Using Raft

February 14, 2025

Anil Joshi

As we know, Orchestrator is a MySQL high availability and replication management tool that aids in managing farms of MySQL servers. In this blog post, we discuss how to make the Orchestrator (which manages MySQL) itself fault-tolerant and highly available.

When considering HA for the Orchestrator one of the popular choices will be using the Raft consensus.

What is Raft?

Raft is a consensus protocol/ algorithm where multiple nodes composed of a (Leader) and (Followers) agree on the same state and value. The Leaders are decided by the quorum and voting, and it is the responsibility of the Leader Raft to do all the decision-making and changes. The other node just follows or syncs with the Leader without involving any direct changes.

When Raft is used with Orchestrator, it provides high availability, solves network partitioning, and ensures fencing on the Isolated node.

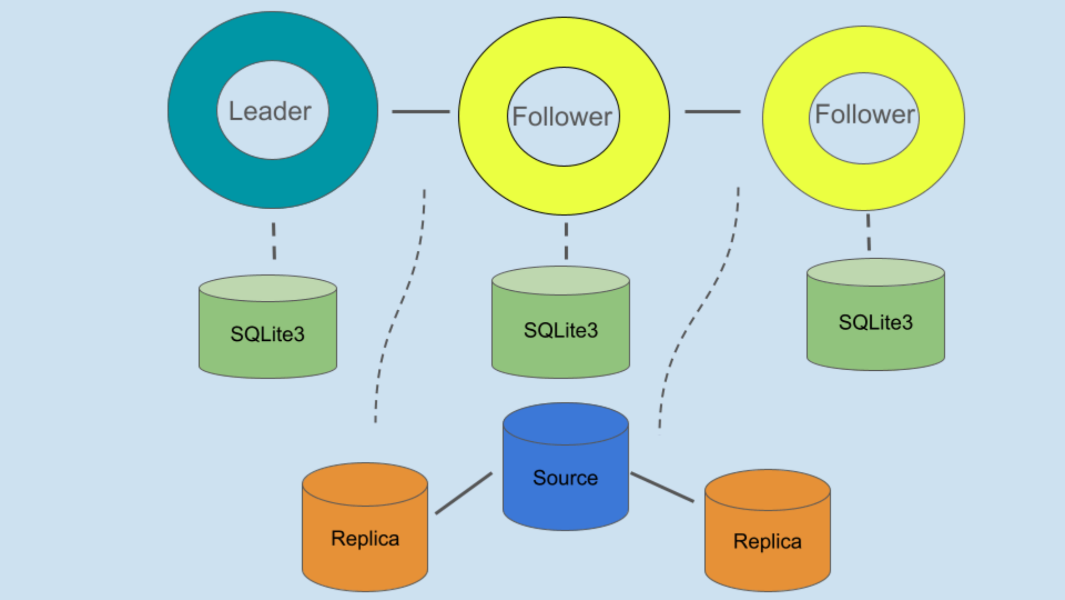
Deployment

Next, we will see how we can deploy an Orchestrator/Raft based setup with the below topology.

For demo purposes, I am using the same server for both Orchestrator/Raft and MySQL.

```
172.31.20.60  Node1
172.31.16.8   Node2
172.31.23.135 Node3
```

So, we have the topology below, which we are going to deploy. Each Raft/Orchestrator node has its own separate SQLite database instance.



Orchestrator Raft

Installation

For this demo, I am installing the packages via Percona distribution. However we can also install the Orchestrator packages from Percona or Openark repositories directly.

```
shell> sudo yum install -y https://repo.percona.com/yum/percona-release-latest.noarch.r
shell> sudo percona-release setup pdps-8.0
shell> sudo yum install -y percona-orchestrator percona-orchestrator-cli percona-orches
shell> sudo yum install -y percona-server-server
```

> Note – Openark is no longer active, and the last update was quite some time ago(“2021”). Therefore, we can rely on the Percona repositories, which have the latest release last pushed on （“2024”).

Orchestrator/Raft configuration

1) Create database-specific users/tables on the Source database node (**Node1**).

```
mysql>  CREATE DATABASE meta;

mysql> CREATE TABLE meta.cluster (
    anchor TINYINT NOT NULL,
    cluster_name VARCHAR(128) CHARACTER SET ascii NOT NULL DEFAULT '',
    cluster_domain VARCHAR(128) CHARACTER SET ascii NOT NULL DEFAULT '',
    PRIMARY KEY (anchor)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

mysql> INSERT INTO meta.cluster (anchor, cluster_name, cluster_domain) VALUES (1, 'tes
```

> Note – Orchestrator will fetch the cluster details from this table.

```
mysql> CREATE USER 'orchestrator'@'%' IDENTIFIED BY 'Orc@1234';
mysql> GRANT SUPER, PROCESS, REPLICATION SLAVE, RELOAD ON *.* TO 'orchestrator'@'%';
mysql> GRANT SELECT ON mysql.slave_master_info TO 'orchestrator'@'%';
mysql> GRANT DROP ON `_pseudo_gtid_`.* TO 'orchestrator'@'%';
mysql> GRANT SELECT ON meta.* TO 'orchestrator'@'%';
```

> Note – These credentials will be used by the Orchestrator to connect to the MySQL backends.

2) Then, we need to copy the orchestrator template file to the /etc/orchestrator.conf.json and perform the necessary changes in the mentioned sections.

```
shell> sudo cp  /usr/local/orchestrator/orchestrator-sample.conf.json /etc/orchestrato
```

- Replace the MySQL topology credentials with the created ones.

```
"MySQLTopologyUser": "orchestrator",
"MySQLTopologyPassword": "Orc@1234",
```

- Remove the below options since we are relying on the SQLite3 database to manage the Orchestrator backend.

```
"MySQLOrchestratorHost": "127.0.0.1",
"MySQLOrchestratorPort": 3306,
"MySQLOrchestratorDatabase": "orchestrator",
"MySQLOrchestratorUser": "orc_server_user",
"MySQLOrchestratorPassword": "orc_server_password",
```

In case we use MySQL as an orchestrator backend then we need the below two changes.

> Create Orchestrator schema and related credentials.
>
> ```
> mysql> CREATE DATABASE IF NOT EXISTS orchestrator;
> mysql> CREATE USER 'orchestrator'@'localhost' identified by 'Orc@1234';
> mysql> GRANT ALL PRIVILEGES ON orchestrator.* TO 'orchestrator'@'localhost';
> ```
>
> You need to replace the details with Orchestrator managing database (MySQL) information.
>
> ```
> "MySQLOrchestratorHost": "127.0.0.1",
> "MySQLOrchestratorPort": 3306,
> "MySQLOrchestratorDatabase": "orchestrator",
> "MySQLOrchestratorUser": "orchestrator",
> "MySQLOrchestratorPassword": "Orc@1234
> ```

- Replace the existing value with the below query to fetch the cluster details from the MySQL node directly.

```
DetectClusterAliasQuery": "SELECT ifnull(max(cluster_name), '''') as cluster_alias f
```

- Add the below SQLite3 configuration. This only applicable when using SQLite database instead of MySQL backend.

```
"BackendDB": "sqlite",
"SQLite3DataFile": "/var/lib/orchestrator/orchestrator.db",
```

- Auto-failover settings.

```
"RecoverMasterClusterFilters": [
    "testcluster"
  ],
  "RecoverIntermediateMasterClusterFilters": [
    "testcluster"
  ],
```

> RecoverMasterClusterFilters => It defines which cluster should be auto failover/recover.
> RecoverIntermediateMasterClusterFilters => It resembles whether recovery/failure for intermediate masters allow. Intermediate masters are the replica hosts, which have their replicas as well.

- Now perform the Raft-related configuration.

Node1**:**

```
"DefaultRaftPort": 10008,
"RaftAdvertise": "172.31.20.60",
"RaftBind": "172.31.20.60",
"RaftDataDir": "/var/lib/orchestrator",
"RaftEnabled": true,
"RaftNodes": [
    "172.31.20.60",
    "172.31.16.8",
    "172.31.23.135"
  ],
```

Node2**:**

```
"DefaultRaftPort": 10008,
"RaftAdvertise": "172.31.16.8",
"RaftBind": "172.31.16.8",
"RaftDataDir": "/var/lib/orchestrator",
"RaftEnabled": true,
"RaftNodes": [
    "172.31.20.60",
    "172.31.16.8",
    "172.31.23.135"
  ],
```

Node3**:**

```
"DefaultRaftPort": 10008,
"RaftAdvertise": "172.31.23.135",
"RaftBind": "172.31.23.135",
"RaftDataDir": "/var/lib/orchestrator",
"RaftEnabled": true,
"RaftNodes": [
    "172.31.20.60",
    "172.31.16.8",
  "172.31.23.135"
  ],
```

> **Note** – Here we mainly replace the **RaftAdvertise**/**RaftBind** configuration for each node. We need to also make sure the communication between the nodes is allowed on the given Raft port (10008).

3) Then, we can create the Raft data directory on each node.

```
shell> mkdir -p /var/lib/orchestrator
```

4) Finally, we can start the Orchestrator service on each node.

```
shell> systemctl start orchestrator
```

Node Discovery:

From the Orchestrator UI- **http://ec2-54-147-20-38.compute-1.amazonaws.com:3000/web/status** directly we can do the initial Node discovery process.

Node Discovery

So here is our MySQL topology consisting of all 3 nodes.

MySQL Topology

Accessing Orchestrator managing database(SQLlite3):

As we are using SQLite3, we can use the below way to access the tables and information from the insight of the database.

```
shell> sqlite3 /var/lib/orchestrator/orchestrator.db
SQLite version 3.34.1 2021-01-20 14:10:07
Enter ".help" for usage hints.

sqlite> .tables
```

Output**:**

```
access_token
active_node
agent_seed
agent_seed_state
async_request
audit
blocked_topology_recovery
candidate_database_instance
cluster_alias
…
node_health
node_health_history
orchestrator_db_deployments
orchestrator_metadata
raft_log
raft_snapshot
raft_store
topology_failure_detection
topology_recovery
topology_recovery_steps
```

Health/Service:

Next, we can check the logs of each node to confirm the status.

```
shell> journalctl -u orchestrator
```

We will see some voting and state changing in the below logs. So, Node2(172.31.16.8) becomes the leader while other nodes follow it.

```
Feb 02 15:09:38 ip-172-31-20-60.ec2.internal orchestrator[18395]: 2025-02-02 15:09:38
Feb 02 15:09:38 ip-172-31-20-60.ec2.internal orchestrator[18395]: 2025/02/02 15:09:38
Feb 02 15:09:38 ip-172-31-20-60.ec2.internal orchestrator[18395]: 2025/02/02 15:09:38
Feb 02 15:09:38 ip-172-31-20-60.ec2.internal orchestrator[18395]: 2025/02/02 15:09:38
Feb 02 15:09:38 ip-172-31-20-60.ec2.internal orchestrator[18395]: 2025/02/02 15:09:38
Feb 02 15:09:39 ip-172-31-20-60.ec2.internal orchestrator[18395]: 2025/02/02 15:09:39
Feb 02 15:09:39 ip-172-31-20-60.ec2.internal orchestrator[18395]: 2025/02/02 15:09:39
Feb 02 15:09:40 ip-172-31-20-60.ec2.internal orchestrator[18395]: 2025/02/02 15:09:40
Feb 02 15:09:40 ip-172-31-20-60.ec2.internal orchestrator[18395]: 2025/02/02 15:09:40
Feb 02 15:09:41 ip-172-31-20-60.ec2.internal orchestrator[18395]: 2025/02/02 15:09:41
Feb 02 15:09:41 ip-172-31-20-60.ec2.internal orchestrator[18395]: 2025/02/02 15:09:41
Feb 02 15:09:41 ip-172-31-20-60.ec2.internal orchestrator[18395]: 2025/02/02 15:09:41
...
Feb 02 15:33:13 ip-172-31-20-60.ec2.internal orchestrator[18395]: 2025-02-02 15:33:13
Feb 02 15:33:18 ip-172-31-20-60.ec2.internal orchestrator[18395]: 2025-02-02 15:33:18
```

We can also use the below curl command to get the status.

```
shell> curl http://localhost:3000/api/status|jq .
```

Output:

```
{
  "Code": "OK",
  "Message": "Application node is healthy",
  "Details": {
    "Healthy": true,
    "Hostname": "ip-172-31-16-8.ec2.internal",
    "Token": "52c01a982d4169dc145b7693d0f86100a952949f6a83d7ef4db6ad5dafe45a8a",
    "IsActiveNode": true,
    "ActiveNode": {
      "Hostname": "172.31.16.8:10008",
      "Token": "",
      "AppVersion": "",
      "FirstSeenActive": "",
      "LastSeenActive": "",
      "ExtraInfo": "",
      "Command": "",
      "DBBackend": "",
      "LastReported": "0001-01-01T00:00:00Z"
    },
    "Error": null,
    "AvailableNodes": [
      {
        "Hostname": "ip-172-31-16-8.ec2.internal",
        "Token": "52c01a982d4169dc145b7693d0f86100a952949f6a83d7ef4db6ad5dafe45a8a",
        "AppVersion": "3.2.6-15",
        "FirstSeenActive": "2025-02-02T17:49:18Z",
        "LastSeenActive": "2025-02-04T18:01:12Z",
        "ExtraInfo": "",
        "Command": "",
        "DBBackend": "/var/lib/orchestrator/orchestrator.db",
        "LastReported": "0001-01-01T00:00:00Z"
      }
    ],
    "RaftLeader": "172.31.16.8:10008",
    "IsRaftLeader": true,
    "RaftLeaderURI": "http://172.31.16.8:3000",
    "RaftAdvertise": "172.31.16.8",
    "RaftHealthyMembers": [
      "172.31.23.135",
      "172.31.20.60",
      "172.31.16.8"
    ]
  }
}
```

In the Orchestrator UI itself we can check the Raft details.

Raft Nodes

Raft Failover/Switchover:

Now consider the current raft-leader Node1(172.31.20.60).

```
shell> orchestrator-client -c raft-leader
```

Output:

```
172.31.20.60:10008
```

If we stop Node1 we can see that one of the follower nodes (Node2) becomes the new leader.

```
Shell > systemctl stop orchestrator
```

Node2**:**

```
Feb 02 18:04:11 ip-172-31-16-8.ec2.internal orchestrator[1854]: 2025/02/02 18:04:11 [D
lines 3028-3073/3073
```

Node3**:**

```
Feb 02 18:02:22 ip-172-31-23-135.ec2.internal orchestrator[1859]: 2025/02/02 18:02:22
Feb 02 18:02:22 ip-172-31-23-135.ec2.internal orchestrator[1859]: 2025/02/02 18:02:22
Feb 02 18:02:25 ip-172-31-23-135.ec2.internal orchestrator[1859]: 2025-02-02 18:02:25 [
```

So, the new leader is Node2(172.31.16.8) now.

```
shell> orchestrator-client -c raft-leader
```

Output:

```
172.31.16.8:10008
```

We can also manually trigger the switchover using the command **raft-elect-leader** from the current leader node.

```
shell> orchestrator-client -c raft-elect-leader -hostname ip-172-31-20-60.ec2.internal
```

Output:

```
ip-172-31-20-60.ec2.internal
```

Basically Raft leader node is responsible for making all topology related changes and recovery. Other nodes just sync/exchange information.

Once we stop the Source database Node3(172.31.23.135) the auto failover happens automatically. These are the logs from the Leader Raft node.

```
Feb  2 18:16:21 ip-172-31-16-8 orchestrator[1854]: 2025-02-02 18:16:21 INFO topology_re
Feb  2 18:16:21 ip-172-31-16-8 orchestrator[1854]: [martini] Started GET /api/audit-rec
Feb  2 18:16:21 ip-172-31-16-8 orchestrator[1854]: [martini] Completed 200 OK in 782.70
Feb  2 18:16:21 ip-172-31-16-8 orchestrator[1854]: [martini] Started GET /api/maintenan
Feb  2 18:16:21 ip-172-31-16-8 orchestrator[1854]: [martini] Completed 200 OK in 2.3429
Feb  2 18:16:21 ip-172-31-16-8 orchestrator[1854]: 2025-02-02 18:16:21 DEBUG orchestrat
Feb  2 18:16:21 ip-172-31-16-8 orchestrator[1854]: 2025-02-02 18:16:21 INFO CommandRun
```

Summary

In this blog post, we explored one of the ways of setting up high availability for the Orchestrator tool. The Raft mechanism has the advantage that it comes with automatic fencing and fault tolerance by voting/consensus mechanism. The leader will be elected and the sole responsible for all changes and recoveries. In a production environment, we should have at least three nodes (odd number) to have a quorum/voting. Also, there are some other ways that exist for configuring HA in an orchestrator using (Semi HA and HA by the shared backend) which we can explore in some other blog posts.

About the Author

Anil Joshi

I am Anil Joshi, and I work for Percona as a support engineer. I've worked with some well-known Open Source database technologies (MySQL/MariaDB, MongoDB, and Redis) for almost ten years.I am keenly interested in learning new databases and writing database content.

Share This Post!

Subscribe ▾ Login

Join the discussion

B I U S {} [+]

2 COMMENTS Oldest ▾

Hari

3 months ago

Hi Anil,
This is really helpful. Just wondering if the setup can be done to handle multiple clusters. Suppose I have 3 different clusters in 3 nodes, can this setup be implemented for all the clusters?

+ 2 — Reply

> **Anil Joshi** Author
>
> Reply to Hari 3 months ago
>
> Hi Hari,
>
> Thanks for your feedback!
>
> Orchestrator is built to manage multiple cluster/topology so this should work. All you need to repeat the configuration and create the necessary users on each

different cluster.

All existing cluster details can be fetched based on the **DetectClusterAliasQuery**

```
"DetectClusterAliasQuery": "SELECT ifnull(max(cluster_name), '''') as cluster_
```

Topology discovery can be done separately as below or from the UI itself.

First Cluster:

```
shell> orchestrator-client -c discover -i 127.0.0.1:22435
shell> orchestrator-client -c discover -i 127.0.0.1:22436
shell> orchestrator-client -c discover -i 127.0.0.1:22437
```

Second Cluster:

```
shell> orchestrator-client -c discover -i 127.0.0.1:19401
shell> orchestrator-client -c discover -i 127.0.0.1:19402
shell> orchestrator-client -c discover -i 127.0.0.1:19403
```

+ 0 — Reply

Related Blog Articles

RECOMMENDED ARTICLES

May 29, 2025

Arunjitl

Aravinc

How to Safely Upgrade InnoDB Cluster From MySQL 8.0 to 8.4

Insight for DBAs

MySQL

May 28, 2025

Andrey

The Open Source Ripple Effect: How Valkey Is Redefining the Future of Caching, and Why It Matters

Insight for DBAs

Open Source

Valkey

MOST POPULAR ARTICLES

June 20, 2023

Sergey P

Deploy Django on Kubernetes With Percona Operator for PostgreSQL

Cloud

Insight for Developers

Percona Software

PostgreSQL

December 28, 2012

Miguel Aı

Nieto

Auditing Login attempts in MySQL

MySQL

May 16, 2016

Muhamn

Irfan

MySQL "Got an error reading commun

ication packet”

MySQL

May 27, 2025 David (

Beyond Guesswork: Enterprise-Grade PostgreSQL Tuning with pg_stat_statements

Insight for DBAs

PostgreSQL

Featured

MySQL 5.7 End of Life

Learn

Downloads

Resources

Products

MySQL

MongoDB

PostgreSQL

Cloud Native

Monitoring

Services

Support

Managed Services

Consulting

Policies

Training

Docs

Percona Toolkit

Use Cases

Emergency Support

High-Traffic Events

Performance Tuning

Migrations

Upgrades

Security

Discover

Blog

Community

Percona Community Forum

Events

About

About Percona

Partners

In The News

Careers

Terms of Use Privacy Copyright Legal Security Center