MariaDB

MariaDB Cluster Course Workbook

Prework

Tasks

1. Sign up at https://training-exercises.mariadb.com/ and notify your instructor that you have completed this task.

2. Log in to https://training-exercises.mariadb.com/ and click on the **`MariaDB Cluster`** training track. Next click on the **`More Info`** link then the **`Eligible`** link, and follow the on-screen instructions for creating your training lab virtual server(s).

3. Secure your training lab virtual server(s) by changing the **`centos`** user's password with the **`passwd`** command on your training lab virtual server(s) to a strong, powerful password that only you know.

Note: Throughout this course, the text in **bold letters** within the code boxes is what you have to type. Everything else is shown in a regular font.

```
[centos@ip-10-0-1-250 ~]$ passwd
Changing password for user centos.
Changing password for centos.
(current) UNIX password:
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[centos@ip-10-0-1-250 ~]$
```

Remember this password so that you can access your training lab virtual server(s) during the class.

4. In this class you will execute all of the commands as the OS root user. Below is an example of how to become the OS root user. Your IP address will differ. Your VMs' IP addresses along with the **`centos`** user's password are listed on https://training-exercises.mariadb.com/.

```
$ ssh centos@54.87.199.244
centos@54.87.199.244's password:
[centos@ip-10-0-1-250 ~]$ sudo -i
[root@ip-10-0-1-250 ~]# whoami
root
```

```
[root@ip-10-0-1-250 ~]#
```

5. On your class virtual server(s), you will need to install the following packages and updates.

Command	**Description**
`yum -y update`	This command brings the operating system up to date with all security patches and updates. **Note:** This may take several minutes.
`yum -y install epel-release`	The EPEL repository is an additional package repository that provides easy access to install packages for commonly used software. **Note:** You will need to install the `EPEL repository` before installing `sysbench`.
`yum -y install sysbench`	This command installs the `sysbench` tool for benchmarking.
`yum -y install socat`	This commands installs `socat`, a command-line utility that establishes two bidirectional byte streams and transfers data between them, on all of your class virtual servers.
`yum -y install nc`	This command installs the `nmap-ncat` utility for networking scanning.
`yum -y install wget`	This command installs `wget` which is used to retrieve content and files from various web servers. It supports downloads via FTP, SFTP, HTTP, and HTTPS.
`yum -y install psmisc`	Optional. This command installs the `psmisc` utility which provides the very hand `pstree` command for displaying the hierarchy of the running processes.
`yum -y install telnet`	Optional. This command installs the `telnet` utility which can be used to interactively send data over a TCP connection and also to verify TCP connectivity.
`yum -y install vim`	Optional. This command installs `vim`, which is an improved version of the classic `vi` editor featuring syntax highlighting and other goodies.
`yum -y install tcpdump`	Optional. This command installs the `tcpdump` utility that can be used to capture and analyze raw network traffic.
`yum -y install git`	This command installs `git` which is a distributed, open-source version control system (VCS).
`git clone https://github.com/mar`	The `git clone` command downloads the remote MariaDB training repository

`iadb-corporation/mariadb-training.git`	(**`https://github.com/mariadb-corporation/mariadb-training`**) that contains sample files that will be used in these lab exercises, and saves it into the **`/root/mariadb-training/`** directory on your class virtual server(s).

Perform the package installations and updates detailed above in a single step by executing the following command as the **`root`** user.

```
# yum -y update; yum -y install epel-release; yum -y install sysbench socat
nc wget git ; cd ~ ; git clone
https://github.com/mariadb-corporation/mariadb-training.git

# yum -y install psmisc telnet vim tcpdump
```

Lab Exercises for Lesson 2: Getting Started

Assumptions

- You will execute all of the commands as the OS `root` user.

Lab Exercise 2-1: Installing and Configuring MariaDB Enterprise Cluster

Overview

In this lab exercise you will deploy and secure a basic three node MariaDB Enterprise Cluster on the class virtual machines.

Duration

This lab exercise should take approximately 30 minutes to complete.

Tasks

1. As the `root` user, download and configure the MariaDB Enterprise package repository on the three database nodes (**mariadb-lab-instance-1**, **mariadb-lab-instance-2** and **mariadb-lab-instance-3**).

```
# wget
https://dlm.mariadb.com/enterprise-release-helpers/mariadb_es_repo_setup

# chmod +x mariadb_es_repo_setup
```

For this training the `CUSTOMER_DOWNLOAD_TOKEN` placeholder shown below will be substituted with the download token that the instructor has provided. In production environments, you will retrieve your Customer Download Token at https://customers.mariadb.com/downloads/token/.

So that you can gain hands-on experience performing a rolling upgrade of MariaDB Enterprise Cluster, set the `mariadb-server-version` flag to `10.5` to install the previous version of MariaDB Enterprise Server.

```
# ./mariadb_es_repo_setup --token="CUSTOMER_DOWNLOAD_TOKEN" --apply \
     --skip-maxscale \
      --skip-tools \
      --mariadb-server-version="10.5"

[info] Repository file successfully written to /etc/yum.repos.d/mariadb.repo

Loaded plugins: fastestmirror
Cleaning repos: base extras mariadb-es-main mariadb-maxscale mariadb-tools updates
Cleaning up list of fastest mirrors
```

On the three database nodes (**mariadb-lab-instance-1**, **mariadb-lab-instance-2** and **mariadb-lab-instance-3**), install MariaDB Enterprise Server and MariaDB Enterprise Backup using the **`yum install`** command.

```
# yum -y install MariaDB-server MariaDB-backup
```

Start MariaDB Enterprise Server with the **`systemctl start`** command on the three database nodes (**mariadb-lab-instance-1**, **mariadb-lab-instance-2** and **mariadb-lab-instance-3**).

```
# systemctl start mariadb
```

After installing MariaDB Enterprise Server, it is important to start and enable the service so that the service will be running. Enable the service on the three database nodes (**mariadb-lab-instance-1**, **mariadb-lab-instance-2** and **mariadb-lab-instance-3**) to ensure that if the database host reboots, the MariaDB Enterprise Server will start up on boot up.

```
# systemctl enable mariadb
Created symlink from
/etc/systemd/system/multi-user.target.wants/mariadb.service to
/usr/lib/systemd/system/mariadb.service.
```

2. On the three database nodes (**mariadb-lab-instance-1**, **mariadb-lab-instance-2** and **mariadb-lab-instance-3**), confirm that MariaDB Enterprise Server 10.5 is installed and running.

```
# mariadb
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 8
Server version: 10.5.13-9-MariaDB-enterprise MariaDB Enterprise Server
```

```
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

MariaDB [(none)]> SELECT VERSION();
+-----------------------------+
| VERSION()                   |
+-----------------------------+
| 10.5.13-9-MariaDB-enterprise |
+-----------------------------+
1 row in set (0.000 sec)

MariaDB [(none)]> exit
Bye
```

3. On the three database nodes (**mariadb-lab-instance-1**, **mariadb-lab-instance-2** and **mariadb-lab-instance-3**), stop the MariaDB server.

```
# systemctl stop mariadb
```

4. On the three database nodes (**mariadb-lab-instance-1**, **mariadb-lab-instance-2** and **mariadb-lab-instance-3**), set both the `TimeoutStartSec` and `TimeoutStopSec` timeouts and the limit size (`LimitNOFILE`) for the number of files that a user (in this case, the `mysql` user) may have open at any given time for systemd. Add these settings in a new file named `custom.conf`.

```
# vi /etc/systemd/system/mariadb.service.d/custom.conf
[Service]
TimeoutStartSec=30min
TimeoutStopSec=30min
LimitNOFILE=655260

# systemctl daemon-reload
```

5. Obtain the private IP addresses of your nodes by executing the command `ip addr show eth0` on each node, and viewing the `inet` address for the `eth0` interface.

```
# ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group
default qlen 1000
```

```
    link/ether 00:1a:4a:16:01:9d brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.187/24 brd 10.0.2.255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::225e:37a0:6d16:9d41/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

In this example the private IP address of the node is: **`10.0.2.187`**.

6. Check that each of the nodes can reach each other over ports 3306, 4567, 4568, 4444 and 22. Since MariaDB is shut down on the nodes, you can expect to see a "Ncat: Connection refused." for all ports except port 22, which should return a "Ncat: Connected to [...]" message. If you see a different message then you will need to further adjust the node's firewall settings.

```
# nc -vz 10.0.2.187 3306
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Connection refused.

# nc -vz 10.0.2.187 4567
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Connection refused.

# nc -vz 10.0.2.187 4568
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Connection refused.

# nc -vz 10.0.2.187 4444
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Connection refused.

# nc -vz 10.0.2.187 22
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Connected to 10.0.2.82:22.
Ncat: 0 bytes sent, 0 bytes received in 0.01 seconds.
```

Replace the IP address above with the actual IP address of the node you are attempting to access.

7. On each node (**mariadb-lab-instance-1**, **mariadb-lab-instance-2** and **mariadb-lab-instance-3**), create a new configuration file called **`/etc/my.cnf.d/z-custom.cnf`**. Configuration files are read in alphabetical order so prefixing the custom configuration file's name with **`z-`** ensures that the custom configuration file will override the bundled configuration files in the **`/etc/my.cnf.d`** directory.

```
# vi /etc/my.cnf.d/z-custom.cnf
```

8. On each node (**mariadb-lab-instance-1**, **mariadb-lab-instance-2** and **mariadb-lab-instance-3**), add the following options under the `[galera]` section. At this point the `wsrep_sst_auth` user and `wsrep_sst_auth` password are not checked at startup so both can be safely defined in the server configuration file. You will create the `wsrep_sst_auth` user in the database later on in this exercise.

On each node (**mariadb-lab-instance-1**, **mariadb-lab-instance-2** and **mariadb-lab-instance-3**), replace the example values for the IP address listed for `wsrep_cluster_address` with your own values for the node.

```
[galera]
wsrep_on=ON
wsrep_provider=/usr/lib64/galera/libgalera_enterprise_smm.so
wsrep_cluster_address="gcomm://10.0.2.187,10.0.2.82,10.0.2.93"
wsrep_auto_increment_control=ON
wsrep_sst_method=mariabackup
wsrep_sst_auth=backupuser:MariaDB_123
```

Replace the IP addresses above with the actual IP addresses of your nodes.

9. On each node (**mariadb-lab-instance-1**, **mariadb-lab-instance-2** and **mariadb-lab-instance-3**), add some options under the `[mariadb]` section.

```
[mariadb]
log_error=/var/lib/mysql/error.log
max_connections=100
innodb_autoinc_lock_mode=2
bind_address=0.0.0.0
binlog_format=row
```

Save your changes and exit the file.

10. On Node 1 only (**mariadb-lab-instance-1**), start MariaDB using the `galera_new_cluster` wrapper script. Starting the first node bootstraps the cluster and registers the cluster name and cluster id.

```
# galera_new_cluster
```

11. Review the error log if MariaDB fails to start up.

```
# journalctl -u mariadb
```

or

```
# systemctl status mariadb.service -l
```

12. Connect to the MariaDB Enterprise Server with the MariaDB Client and view the cluster status, and the cluster variables listed below. The **wsrep_cluster_size** server variable should show that there is 1 node in the cluster.

```
# mariadb

MariaDB [(none)]> SHOW GLOBAL STATUS LIKE 'wsrep_cluster_%';
+----------------------------+--------------------------------------+
| Variable_name              | Value                                |
+----------------------------+--------------------------------------+
| wsrep_cluster_weight       | 1                                    |
| wsrep_cluster_capabilities |                                      |
| wsrep_cluster_conf_id      | 1                                    |
| wsrep_cluster_size         | 1                                    |
| wsrep_cluster_state_uuid   | 846da951-72e5-11ec-8ee0-f26c4e9d3ae7 |
| wsrep_cluster_status       | Primary                              |
+----------------------------+--------------------------------------+
6 rows in set (0.001 sec)
```

If you would like to take a look at more cluster variables, review the output from the **SHOW STATUS LIKE 'wsrep%'** and **SHOW VARIABLES LIKE 'wsrep%'** statements.

13. On Node 1 (**mariadb-lab-instance-1**), create the cluster replication user, **backupuser**.

```
MariaDB [(none)]> CREATE USER 'backupuser'@'localhost' IDENTIFIED BY
'MariaDB_123';
Query OK, 0 rows affected (0.002 sec)

MariaDB [(none)]> GRANT RELOAD, LOCK TABLES, PROCESS, REPLICATION CLIENT ON
*.* TO 'backupuser'@'localhost';
Query OK, 0 rows affected (0.002 sec)
```

14. On Node 1 (**mariadb-lab-instance-1**), create a couple of `labtest` users that will be used in later lab exercises in this workbook.

```
MariaDB [(none)]> CREATE USER 'labtest'@'localhost' IDENTIFIED BY
'MariaDB_123';
Query OK, 0 rows affected (0.002 sec)

MariaDB [(none)]> CREATE USER 'labtest'@'10.0.2.%' IDENTIFIED BY
'MariaDB_123';
Query OK, 0 rows affected (0.002 sec)

MariaDB [(none)]> GRANT ALL ON *.* TO 'labtest'@'localhost';
Query OK, 0 rows affected (0.002 sec)

MariaDB [(none)]> GRANT ALL ON *.* TO 'labtest'@'10.0.2.%';
Query OK, 0 rows affected (0.002 sec)

MariaDB [(none)]> exit
Bye
```

Replace the IP address above with the actual IP address of your nodes.

15. Add Node 2 (**mariadb-lab-instance-2**) and then Node 3 (**mariadb-lab-instance-3**) to the cluster by starting up MariaDB on each node.

Wait until Node 2 has successfully completed its start up before starting up Node 3.

```
# systemctl start mariadb
```

16. Review the error log if MariaDB fails to start up.

```
# journalctl -u mariadb
```

or

```
# systemctl status mariadb.service -l
```

17. On Node 2 (**mariadb-lab-instance-2**) and Node 3 (**mariadb-lab-instance-3**), log into MariaDB with the MariaDB Client and check the cluster status.

```
# mariadb

MariaDB [(none)]> SHOW GLOBAL STATUS LIKE 'wsrep_cluster_%';
+----------------------------+--------------------------------------+
| Variable_name              | Value                                |
+----------------------------+--------------------------------------+
| wsrep_cluster_weight       | 3                                    |
| wsrep_cluster_capabilities |                                      |
| wsrep_cluster_conf_id      | 3                                    |
| wsrep_cluster_size         | 3                                    |
| wsrep_cluster_state_uuid   | 5b032e6a-72e6-11ec-bcec-2732ad6acbe7 |
| wsrep_cluster_status       | Primary                              |
+----------------------------+--------------------------------------+
6 rows in set (0.001 sec)
```

Take note of the values for **`wsrep_cluster_size`** and **`wsrep_cluster_status`**.

18. All nodes should now be running and consistent.

Lab Exercise 2-2: Testing MariaDB Enterprise Cluster Replication

Overview

In this lab exercise you will test cluster replication by creating a new table and inserting rows into the table on one node, and then verifying that the table and rows were created on the other nodes.

Duration

This lab exercise should take approximately 10 minutes to complete.

Tasks

1. On Node 1 (**mariadb-lab-instance-1**), create a new database named **`seal`** and table named **`table1`**, and insert rows into **`table1`**.

```
# mariadb
```

```
MariaDB [(none)]> CREATE DATABASE seal;
Query OK, 1 row affected (0.003 sec)

MariaDB [(none)]> CREATE TABLE seal.table1 (col1 INT UNSIGNED KEY);
Query OK, 0 rows affected (0.012 sec)

MariaDB [(none)]> INSERT INTO seal.table1 VALUES (1),(2),(3);
Query OK, 3 rows affected (0.010 sec)
Records: 3  Duplicates: 0  Warnings: 0

MariaDB [(none)]> exit
Bye
```

2. On Node 2 (**mariadb-lab-instance-2**), check that the rows inserted into `table1` on Node 1 show up in `table1` on Node 2. Then insert another row into `table1`.

```
# mariadb

MariaDB [(none)]> SELECT * FROM seal.table1;
+------+
| col1 |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (0.000 sec)

MariaDB [(none)]> INSERT INTO seal.table1 VALUES (4);
Query OK, 1 row affected (0.003 sec)

MariaDB [(none)]> exit
Bye
```

3. On Node 3 (**mariadb-lab-instance-3**), check that the row inserted into `table1` on Node 2 shows up in `table1` on Node 3.

```
# mariadb

MariaDB [(none)]> SELECT * FROM seal.table1;
+------+
| col1 |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
+------+
4 rows in set (0.000 sec)

MariaDB [(none)]> exit
Bye
```

Lab Exercise 2-3: Upgrading MariaDB Enterprise Cluster

Overview

In this exercise you will perform a rolling upgrade of MariaDB Enterprise Cluster. In a production environment, you would make certain to backup your data before performing the upgrade.

Duration

This lab exercise should take approximately 20 minutes to complete.

Tasks

1. On all three nodes (**mariadb-lab-instance-1**, **mariadb-lab-instance-2** and **mariadb-lab-instance-3**), update the repository information to version 10.6, and clean the cache.

```
# ./mariadb_es_repo_setup --token="CUSTOMER_DOWNLOAD_TOKEN" --apply \
      --mariadb-server-version="10.6" \
      --skip-maxscale \
      --skip-tools

# yum clean all
Loaded plugins: fastestmirror
Cleaning repos: base epel extras mariadb-main mariadb-maxscale mariadb-tools
updates
Cleaning up list of fastest mirrors
```

2. On Node 1 (**mariadb-lab-instance-1**), stop the MariaDB Server.

```
# systemctl stop mariadb
```

3. On Node 1 (**mariadb-lab-instance-1**), remove MariaDB and then install the newer version.

```
# yum -y erase MariaDB-server MariaDB-backup
# yum -y install MariaDB-server MariaDB-backup
```

4. On Node 1 (**mariadb-lab-instance-1**), edit the `z-custom.cnf` configuration file and the `wsrep_ssl_mode` system variable for Galera TLS usage with SST. MariaDB ES 10.6 has this enabled by default, but the other cluster nodes, which are still running MariaDB Enterprise Server 10.5 at this point, have the `wsrep_ssl_mode` disabled. Therefore, we need to set `wsrep_ssl_mode` to `PROVIDER` so that MariaDB Enterprise Cluster obtains its TLS configuration from the `wsrep_provider_options` system variable.

```
# vi /etc/my.cnf.d/z-custom.cnf

[galera]
...
wsrep_ssl_mode = PROVIDER
...
```

5. On Node 1 (**mariadb-lab-instance-1**), start the MariaDB Server.

```
# systemctl start mariadb
```

6. On Node 1 (**mariadb-lab-instance-1**), update the system database files by running the `mariadb-upgrade` script.

```
# mariadb-upgrade
Phase 1/7: Checking and upgrading mysql database
Processing databases
mysql
mysql.column_stats                                  OK
[...]
mysql.wsrep_cluster                                 OK
mysql.wsrep_cluster_members                         OK
mysql.wsrep_streaming_log                          OK
Phase 2/7: Installing used storage engines... Skipped
Phase 3/7: Fixing views
mysql.user                                         OK
Phase 4/7: Running 'mysql_fix_privilege_tables'
Phase 5/7: Fixing table and database names
Phase 6/7: Checking and upgrading tables
Processing databases
information_schema
performance_schema
seal
seal.table1                                        OK
Phase 7/7: Running 'FLUSH PRIVILEGES'
OK
```

7. Repeat steps 2-6 on Node 2 (**mariadb-lab-instance-2**), then again on Node 3 (**mariadb-lab-instance-3**).

8. After all nodes have been upgraded, verify cluster integrity by executing the **`SHOW GLOBAL STATUS LIKE 'wsrep_cluster_%'`** statement on any of the three nodes.

```
# mariadb

MariaDB [(none)]> SHOW GLOBAL STATUS LIKE 'wsrep_cluster_%';
+----------------------------+--------------------------------------+
| Variable_name              | Value                                |
+----------------------------+--------------------------------------+
| wsrep_cluster_weight       | 3                                    |
| wsrep_cluster_capabilities |                                      |
| wsrep_cluster_conf_id      | 11                                   |
| wsrep_cluster_size         | 3                                    |
| wsrep_cluster_state_uuid   | b0877816-7c97-11eb-93b5-f72f64d9039d |
| wsrep_cluster_status       | Primary                              |
+----------------------------+--------------------------------------+
6 rows in set (0.001 sec)

MariaDB [(none)]> SELECT VERSION();
+----------------------------+
| VERSION()                  |
+----------------------------+
| 10.6.5-2-MariaDB-enterprise |
+----------------------------+
1 row in set (0.000 sec)
```

Lab Exercises for Lesson 3: State Transfers

Assumptions

- You will execute all of the commands as the OS **`root`** user.

Lab Exercise 3-1: Performing a State Snapshot Transfer (SST)

Overview

In this lab exercise you will perform a state snapshot transfer with one of the nodes of your three node MariaDB Enterprise Cluster. You will also observe how MariaDB Cluster uses the **`grastate.dat`** file.

Duration

This lab exercise should take approximately 10 minutes to complete.

Tasks

1. On Node 3 (**mariadb-lab-instance-3**), disconnect from the MariaDB server and check the contents of the **`grastate.dat`** file, especially the **`seqno`** line.

```
MariaDB [(none)]> exit
Bye

# cat /var/lib/mysql/grastate.dat
# GALERA saved state
version: 2.1
uuid:    b0877816-7c97-11eb-93b5-f72f64d9039d
seqno:   -1
safe_to_bootstrap: 0
```

2. Stop the MariaDB server on Node 3 (**mariadb-lab-instance-3**).

```
# systemctl stop mariadb
```

3. Check the **grastate.dat** file again. You will notice that while the cluster node is running **seqno** is set to **-1**. After the cluster node is stopped, the **seqno** field has the last writeset ID that was locally committed. Your actual value may vary from what is displayed in the output below.

```
# cat /var/lib/mysql/grastate.dat
# GALERA saved state
version: 2.1
uuid:    b0877816-7c97-11eb-93b5-f72f64d9039d
seqno:   19
safe_to_bootstrap: 0
```

4. On Node 3 (**mariadb-lab-instance-3**), delete the contents of the data directory. An empty data directory will force an SST transfer when the node rejoins the cluster.

```
# rm -rf /var/lib/mysql/*
```

5. Open a second terminal window to the server of Node 3 (**mariadb-lab-instance-3**), and start the one-line script given here; it will wait for the appearance of the log file (because SST will delete everything from the data directory) and will show it on the screen.

```
# cd /var/lib/mysql

# while :; do if [ -f error.log ]; then tail -f error.log; else sleep 1; fi done
```

6. On the first terminal window to the server of Node 3 (**mariadb-lab-instance-3**), restart MariaDB.

```
# systemctl start mariadb
```

Monitor the error log to see what information is written in the second terminal window.

Once the SST completes, stop the log on the second screen with Ctrl+C.

Lab Exercise 3-2: Performing an Incremental State Transfer (IST)

Overview

In this lab exercise you will perform an incremental transfer with one of the nodes of your 3 node MariaDB Cluster.

Duration

This lab exercise should take approximately 10 minutes to complete.

Tasks

1. On Node 3 stop MariaDB server.

```
# systemctl stop mariadb
```

2. On Node 1, change some data in the **seal** database. For example, add another row into the **table1** table.

```
MariaDB [(none)]> USE seal;

MariaDB [seal]> SHOW TABLES;
+----------------+
| Tables_in_seal |
+----------------+
| table1         |
+----------------+
1 row in set (0.000 sec)

MariaDB [seal]> SELECT * FROM table1;
+------+
| col1 |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
+------+
4 rows in set (0.000 sec)
```

```
MariaDB [seal]> INSERT INTO table1 VALUES (5);
Query OK, 1 row affected (0.003 sec)

MariaDB [seal]> SELECT * FROM table1;
+------+
| col1 |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
+------+
5 rows in set (0.000 sec)
```

3. Still on Node 1, view some of the **`wsrep`** status variables.

```
MariaDB [seal]> SHOW GLOBAL STATUS LIKE 'wsrep_last_committed';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| wsrep_last_committed | 23    |
+----------------------+-------+
1 row in set (0.001 sec)

MariaDB [seal]> SHOW GLOBAL STATUS LIKE 'wsrep_local_cached_downto';
+---------------------------+-------+
| Variable_name             | Value |
+---------------------------+-------+
| wsrep_local_cached_downto | 1     |
+---------------------------+-------+
1 row in set (0.001 sec)
```

This shows how to determine the oldest and newest entry in the Galera cache. Your actual value may vary from what is displayed in the output above.

Now you should be able to calculate the difference and know how many entries are in the Galera cache.

4. On Node 3, view the **`grastate.dat`** file that is located in the data directory and compare the value for **`seqno`** with the **`wsrep_last_committed`** value retrieved on Node 1. The two values should differ.

```
# cat /var/lib/mysql/grastate.dat
# GALERA saved state
version: 2.1
uuid:    b0877816-7c97-11eb-93b5-f72f64d9039d
seqno:   21
safe_to_bootstrap: 0
```

Your actual value may vary from what is displayed in the output below.

5. On the second terminal window to the server of Node 3 that you have, start a tail on the error log of MariaDB. This will show only the most recent journal entries, and continuously print new entries as they are appended to the journal.

```
# tail -f /var/lib/mysql/error.log
```

6. Now on the first terminal window to the server of Node 3, restart MariaDB. Monitor the journal log to see what information is written in the second terminal window. Once the IST is done, stop the follow-up mode in the second terminal with Ctrl+C and exit the session.

```
# systemctl start mariadb
```

Lab Exercises for Lesson 4: Caveats

Assumptions

- You will execute all of the commands as the OS **`root`** user.

Lab Exercise 4-1: Controlling AUTO_INCREMENT

Overview

In this lab exercise you will create an **`AUTO_INCREMENT`** table and insert rows into the table. Then you will query the database to observe how nodes in a MariaDB Cluster control the **`AUTO_INCREMENT`** attribute.

Duration

This lab exercise should take approximately 15 minutes to complete.

Tasks

1. On Node 1 (**mariadb-lab-instance-1**), connect to the **`seal`** database (if necessary) and create a table named **`a`** to use for **`auto_increment`** testing.

```
# mariadb seal

MariaDB [seal]> CREATE TABLE a (i INT PRIMARY KEY AUTO_INCREMENT, n VARCHAR(10));
Query OK, 0 rows affected (0.012 sec)
```

2. Again on Node 1 (**mariadb-lab-instance-1**), insert a few rows into the **`auto_increment`** test table, and then view the results.

```
MariaDB [seal]> INSERT INTO a (n) VALUES ('n1'),('n1'),('n1');
Query OK, 3 rows affected (0.004 sec)
Records: 3  Duplicates: 0  Warnings: 0

MariaDB [seal]> SELECT * FROM a;
+---+------+
| i | n    |
+---+------+
| 3 | n1   |
| 6 | n1   |
| 9 | n1   |
+---+------+
3 rows in set (0.000 sec)
```

Your actual values may vary from what is displayed in the output above.

3. Still on Node 1 (**mariadb-lab-instance-1**), view the value of the **`auto_increment`** variable. Pay attention to the increment (3 as the number of nodes in the cluster) and the offset (3 as the node position in the cluster).

```
MariaDB [seal]> SHOW VARIABLES LIKE 'auto_increment%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 3     |
| auto_increment_offset    | 3     |
+--------------------------+-------+
2 rows in set (0.001 sec)
```

Your actual values may vary from what is displayed in the output above.

4. On Node 2 (**mariadb-lab-instance-2**), connect to the **`seal`** database and insert a few rows into the **`auto_increment`** test table, and then view the results.

```
# mariadb seal

MariaDB [seal]> INSERT INTO a (n) VALUES ('n2'),('n2'),('n2');
Query OK, 3 rows affected (0.003 sec)
Records: 3  Duplicates: 0  Warnings: 0

MariaDB [seal]> SELECT * FROM a;
+----+------+
| i  | n    |
+----+------+
|  3 | n1   |
|  6 | n1   |
|  9 | n1   |
| 10 | n2   |
| 13 | n2   |
| 16 | n2   |
+----+------+
6 rows in set (0.000 sec)
```

5. Again on Node 2 (**mariadb-lab-instance-2**), view the value of the **auto_increment** variable. Pay attention to the increment (3 as the number of nodes in the cluster) and the offset (1 as the node position in the cluster - different from the previous Node 1).

```
MariaDB [seal]> SHOW VARIABLES LIKE 'auto_increment%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 3     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
2 rows in set (0.001 sec)
```

6. On Node 3 (**mariadb-lab-instance-3**), connect to the MariaDB server using the MariaDB Client and view the value of the **auto_increment** variable. Pay attention to the increment (3 as the number of nodes in the cluster) and the offset (2 as the node position in the cluster - different from the previous nodes).

```
# mariadb

MariaDB [(none)]> SHOW VARIABLES LIKE 'auto_increment%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 3     |
| auto_increment_offset    | 2     |
+--------------------------+-------+
2 rows in set (0.001 sec)
```

7. Again on Node 3 (**mariadb-lab-instance-3**), insert a few rows into the **auto_increment** test table, and then view the results.

```
MariaDB [(none)]> USE seal;

MariaDB [seal]> INSERT INTO a (n) VALUES ('n3'),('n3'),('n3');
Query OK, 3 rows affected (0.003 sec)
```

```
Records: 3  Duplicates: 0  Warnings: 0

MariaDB [seal]> SELECT * FROM a;
+----+------+
| i  | n    |
+----+------+
|  3 | n1   |
|  6 | n1   |
|  9 | n1   |
| 10 | n2   |
| 13 | n2   |
| 16 | n2   |
| 17 | n3   |
| 20 | n3   |
| 23 | n3   |
+----+------+
9 rows in set (0.000 sec)
```

8. On Node 2 (**mariadb-lab-instance-2**), disable MariaDB Cluster's **`auto_increment`** control by globally setting **`wsrep_auto_increment_control`** to **`0.`**

```
MariaDB [seal]> SET GLOBAL wsrep_auto_increment_control=0;
Query OK, 0 rows affected (0.000 sec)
```

9. On Node 3 (**mariadb-lab-instance-3**), stop the MariaDB server.

```
MariaDB [(none)]> exit
Bye

# systemctl stop mariadb
```

10. On Node 1 (**mariadb-lab-instance-1**) check the value of the **`auto_increment`** variable.

```
MariaDB [(none)]> SHOW VARIABLES LIKE 'auto_increment%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 2     |
| auto_increment_offset    | 2     |
+--------------------------+-------+
2 rows in set (0.001 sec)
```

11. On Node 2 (**mariadb-lab-instance-2**) check the value of the **`auto_increment`** variable.

```
MariaDB [(seal)]> SHOW VARIABLES LIKE 'auto_increment%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 1     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
2 rows in set (0.001 sec)
```

12. On Node 3 (**mariadb-lab-instance-3**), start the MariaDB server.

```
# systemctl start mariadb
```

13. On Node 1 (**mariadb-lab-instance-1**) check the value of the **`auto_increment`** variable.

```
MariaDB [(none)]> SHOW VARIABLES LIKE 'auto_increment%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 3     |
| auto_increment_offset    | 2     |
+--------------------------+-------+
2 rows in set (0.001 sec)
```

14. On Node 2 (**mariadb-lab-instance-2**) check the value of the **`auto_increment`** variable.

```
MariaDB [(none)]> SHOW VARIABLES LIKE 'auto_increment%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 1     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
2 rows in set (0.001 sec)
```

15. On Node 2 (**mariadb-lab-instance-2**), restore auto increment control to automatic and check the value of the **`auto_increment`** variable.

```
MariaDB [(none)]> SET GLOBAL wsrep_auto_increment_control=1;
Query OK, 0 rows affected (0.000 sec)

MariaDB [(none)]> SHOW VARIABLES LIKE 'auto_increment%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 3     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
2 rows in set (0.001 sec)
```

You should observe that the **`auto_increment_increment`** variable has been restored to its original value as shown in step 5.

Lab Exercise 4-2: Spotting Causality Failure

Overview

In this lab exercise you will execute a script called **causal_read_test.sh** to generate load against the cluster.

Duration

This lab exercise should take approximately 15 minutes to complete.

Tasks

1. On Node 1 (**mariadb-lab-instance-1**), prepare for the first test run by logging into the MariaDB server with the MariaDB Client, and creating a new database named **test** and a new table named **t** in the **test** database.

```
# mariadb

MariaDB [(none)]> CREATE DATABASE test;
Query OK, 1 row affected (0.003 sec)

MariaDB [(none)]> USE test;
MariaDB [test]> CREATE TABLE t (id int unsigned primary key);
Query OK, 0 rows affected (0.012 sec)
```

2. On all three database nodes (**mariadb-lab-instance-1**, **mariadb-lab-instance-2** and **mariadb-lab-instance-3**), increase the depth of the flow control queue from the default of 16 to 1,000 with the `SET GLOBAL` statement. You will learn more about flow control in the next lesson.

```
MariaDB [(none)]> SET GLOBAL wsrep_provider_options = 'gcs.fc_limit=1000';
```

3. For this lab exercise, you will use the **causal_read_test.sh** script that was downloaded from the MariaDB git repository into the `/root/mariadb-training/sample_scripts` directory on your class virtual machine(s).

The **causal_read_test.sh** script will generate workload by inserting and selecting 1000 rows.

Open a second terminal window to Node 1 (**mariadb-lab-instance-1**) and copy the **causal_read_test.sh** script from the /mariadb-training/sample_scripts into the `centos` user's home directory.

```
# cd /home/centos
# cp /root/mariadb-training/sample_scripts/causal_read_test.sh .
```

4. In the second terminal window to Node 1 (**mariadb-lab-instance-1**), open the **causal_read_test.sh** script.

```
# vi causal_read_test.sh
```

5. Replace the `NODE-ONE-IP-ADDRESS` and `NODE-TWO-IP-ADDRESS` placeholders with the IP addresses of your own Node 1 (**mariadb-lab-instance-1**) and Node 2 (**mariadb-lab-instance-2**), and the `PASSWORD` placeholder with the password that you assigned to your `labtest` user during Lab Exercise 2-1.

```
#!/bin/bash
PASSWORD="PASSWORD"
NODE1="NODE-ONE-IP-ADDRESS"
NODE2="NODE-TWO-IP-ADDRESS"
[...]
```

Save the changes and exit out of the file.

6. Still in the second terminal window to Node 1 (**mariadb-lab-instance-1**), make the **causal_read_test.sh** script executable.

```
# chmod 755 causal_read_test.sh
```

7. Still in the second terminal window to Node 1 (**mariadb-lab-instance-1**), run the **causal_read_test.sh** script to generate the workload. While viewing the output from the script, you will observe that the script will echo `"Failed"` to standard out if the `SELECT` does not see the record previously inserted on the other node and show the number of failures at the end.

```
# /home/centos/causal_read_test.sh > output_test1
```

8. In the second terminal window to Node 1 (**mariadb-lab-instance-1**), view the contents of the `output_test1` file.

```
# less output_test1
[...]
351 causal read failures per 1000 reads
```

Hit the `'q'` key to exit out of the file.

9. On Node 1 (**mariadb-lab-instance-1**), prepare for the second test run by logging into the MariaDB server and emptying the table `t` completely using the `TRUNCATE TABLE` statement.

```
MariaDB [test]> SELECT COUNT(*) FROM t;
+----------+
| count(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.001 sec)

MariaDB [test]> TRUNCATE TABLE t;
Query OK, 0 rows affected (0.008 sec)

MariaDB [test]> SELECT COUNT(*) FROM t;
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.000 sec)
```

10. On Node 2 (**mariadb-lab-instance-1**), configure read causality by setting **`wsrep_sync_wait`** to **`1`**. This will switch the node from the default mode of optimistic transactions (where the client is released after they get certified by the remote nodes) to the pessimistic mode (where the client is released after they get applied by the remote node).

```
MariaDB [test]> SET GLOBAL wsrep_sync_wait = 1;
Query OK, 0 rows affected (0.000 sec)
```

11. On Node 1 (**mariadb-lab-instance-1**), re-run the **causal_read_test.sh** script to generate the workload and then view the output from the script. The script will echo **`"Failed"`** to standard out if the **SELECT** does not see the record previously inserted on the other node.

```
# /home/centos/causal_read_test.sh > output_test2
```

12. On Node 1 (**mariadb-lab-instance-1**), view the contents of the **`output_test2`** file and compare the output with the output seen during the first run.

```
# less output_test2
[...]
0 causal read failures per 1000 reads
```

Hit the **`'q'`** key to exit out of the file.

13. Compare the output from this second test run with the output from the first test run.

14. On any node of the cluster, clean up by dropping the test database.

```
# mariadb

MariaDB [(none)]> DROP DATABASE test;
Query OK, 1 row affected (0.009 sec)
```

15. On all three database nodes (**mariadb-lab-instance-1**, **mariadb-lab-instance-2** and **mariadb-lab-instance-3**), set the depth of the flow control queue back to the default of 16 with the `SET GLOBAL` statement.

```
# mariadb
MariaDB [(none)]> SET GLOBAL wsrep_provider_options = 'gcs.fc_limit=16';
Query OK, 1 row affected (0.009 sec)
```

Lab Exercise 4-3: Configuring Flow Control

Overview

In this lab exercise you will configure flow control within the cluster and observe the effect of this change by generating a heavy load using `mariadb-slap`. The `mariadb-slap` utility is part of MariaDB Server and can be used to generate artificial load on the database server by automatically creating and running queries against the database server. This can be used for various purposes from simple benchmarking of the server's performance to load simulations that are necessary to test various server configurations, or network throughput.

Duration

This lab exercise should take approximately 15 minutes to complete.

Tasks

1. On Node 2 (**mariadb-lab-instance-2**), connect to the MariaDB server using the MariaDB Client and configure the flow control limit to a low limit using the `SET GLOBAL wsrep_provider_options` statement. The value set will lower the threshold for triggering flow control from the default of 16 worksets in the queue to two worksets. This change should cause the flow control to engage much earlier, slowing down both replication and ingestion.

```
# mariadb

MariaDB [(none)]> SET GLOBAL wsrep_provider_options = 'gcs.fc_limit=2';
Query OK, 0 rows affected (0.000 sec)
```

2. On both Node 1 (**mariadb-lab-instance-1**) and Node 2 (**mariadb-lab-instance-2**), check that the values of the flow control status are zero, or very close to zero, with the MariaDB Client. These values are computed separately by each node so some small variations may exist, especially in the average queue size on Node 2, which so far has received more worksets over the replication link than Node 1. Take a note of the values for the average depth of the send and receive queues on each of the two nodes.

On Node 1 (**mariadb-lab-instance-1**):

```
MariaDB [(none)]> SHOW STATUS LIKE 'wsrep_flow_control_%';
+------------------------------+-------+
| Variable_name                | Value |
+------------------------------+-------+
| wsrep_flow_control_paused_ns | 0     |
| wsrep_flow_control_paused    | 0     |
| wsrep_flow_control_sent      | 0     |
| wsrep_flow_control_recv      | 0     |
| wsrep_flow_control_active    | false |
| wsrep_flow_control_requested | false |
+------------------------------+-------+
6 rows in set (0.001 sec)

MariaDB [(none)]> SHOW STATUS LIKE 'wsrep_local_send_queue_avg';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| wsrep_local_send_queue_avg | 0     |
+----------------------------+-------+
1 row in set (0.001 sec)

MariaDB [(none)]> SHOW STATUS LIKE 'wsrep_local_recv_queue_avg';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| wsrep_local_recv_queue_avg | 0     |
+----------------------------+-------+
1 row in set (0.000 sec)
```

Node 2 (**mariadb-lab-instance-2**):

```
MariaDB [(none)]> SHOW STATUS LIKE 'wsrep_flow_control_%';
+------------------------------+-------+
| Variable_name                | Value |
+------------------------------+-------+
| wsrep_flow_control_paused_ns | 0     |
| wsrep_flow_control_paused    | 0     |
| wsrep_flow_control_sent      | 0     |
| wsrep_flow_control_recv      | 0     |
| wsrep_flow_control_active    | false |
| wsrep_flow_control_requested | false |
+------------------------------+-------+
6 rows in set (0.001 sec)

MariaDB [(none)]> SHOW STATUS LIKE 'wsrep_local_send_queue_avg';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| wsrep_local_send_queue_avg | 0     |
+----------------------------+-------+
1 row in set (0.001 sec)

MariaDB [(none)]> SHOW STATUS LIKE 'wsrep_local_recv_queue_avg';
+----------------------------+------------+
| Variable_name              | Value      |
+----------------------------+------------+
| wsrep_local_recv_queue_avg | 0.00337349 |
+----------------------------+------------+
1 row in set (0.001 sec)
```

3. On Node 1 (**mariadb-lab-instance-1**), generate heavy load on the database server using **`mariadb-slap`**.

```
# mariadb-slap -a --auto-generate-sql-add-autoincrement \
--commit=50 --concurrency=50 --engine=InnoDB \
--number-char-cols=10 --number-int-cols=5 \
--iterations=100
```

4. On Node 1 (**mariadb-lab-instance-1**) and Node 2 (**mariadb-lab-instance-2**), repeat the queries that were executed in step 2. Observe the differences such as for how long was flow control triggered and what were the average depths of the sending and receiving queues on each of the nodes.

Node 1 (**mariadb-lab-instance-1**):

```
MariaDB [(none)]> SHOW STATUS LIKE 'wsrep_flow_control_%';
```

```
+-------------------------------+--------------+
| Variable_name                 | Value        |
+-------------------------------+--------------+
| wsrep_flow_control_paused_ns  | 242641900842 |
| wsrep_flow_control_paused     | 0.00270622   |
| wsrep_flow_control_sent       | 0            |
| wsrep_flow_control_recv       | 560          |
| wsrep_flow_control_active     | false        |
| wsrep_flow_control_requested  | false        |
+-------------------------------+--------------+
6 rows in set (0.001 sec)

MariaDB [(none)]> SHOW STATUS LIKE 'wsrep_local_send_queue_avg';
+----------------------------+---------+
| Variable_name              | Value   |
+----------------------------+---------+
| wsrep_local_send_queue_avg | 1.27776 |
+----------------------------+---------+
1 row in set (0.001 sec)

MariaDB [(none)]> SHOW STATUS LIKE 'wsrep_local_recv_queue_avg';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| wsrep_local_recv_queue_avg | 0     |
+----------------------------+-------+
1 row in set (0.000 sec)
```

Node 2 (**mariadb-lab-instance-2**):

```
MariaDB [(none)]> SHOW STATUS LIKE 'wsrep_flow_control_%';
+-------------------------------+--------------+
| Variable_name                 | Value        |
+-------------------------------+--------------+
| wsrep_flow_control_paused_ns  | 242615042709 |
| wsrep_flow_control_paused     | 0.00269828   |
| wsrep_flow_control_sent       | 560          |
| wsrep_flow_control_recv       | 560          |
| wsrep_flow_control_active     | false        |
| wsrep_flow_control_requested  | false        |
+-------------------------------+--------------+
6 rows in set (0.001 sec)

MariaDB [(none)]> SHOW STATUS LIKE 'wsrep_local_send_queue_avg';
```

```
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| wsrep_local_send_queue_avg | 0     |
+----------------------------+-------+
1 row in set (0.001 sec)

MariaDB [(none)]> SHOW STATUS LIKE 'wsrep_local_recv_queue_avg';
+----------------------------+---------+
| Variable_name              | Value   |
+----------------------------+---------+
| wsrep_local_recv_queue_avg | 8.49287 |
+----------------------------+---------+
1 row in set (0.001 sec)
```

Compare the average depth of the sending queue on Node 1 (which received all of the database queries and sent worksets to Node 2) to the average depth of the receiving queue on Node 2 (which received all the worksets from Node 1).

Lab Exercise 4-4: Configuring True Parallel Replication

Overview

In this lab exercise you will set up true parallel replication within the cluster and observe how the level of parallelism affects the performance of a cluster's node and the speed of replication.

Duration

This lab exercise should take approximately 15 minutes to complete.

Tasks

1. Open a terminal session to Node 1 (**mariadb-lab-instance-1**), and open two terminal sessions to Node 2 (**mariadb-lab-instance-2**). These terminal sessions must be opened to the respective nodes before continuing with the lab exercises.

2. On Node 2 (**mariadb-lab-instance-2**), check the number of replica threads by reading the value of `wsrep_slave_threads` variable.

```
# mariadb

MariaDB [(none)]> SHOW GLOBAL VARIABLES LIKE 'wsrep_slave_threads';
+---------------------+-------+
| Variable_name       | Value |
+---------------------+-------+
| wsrep_slave_threads | 1     |
+---------------------+-------+
1 row in set (0.001 sec)
```

3. On Node 2 (**mariadb-lab-instance-2**) in the second terminal session, monitor the CPU usage on the server using **`top`**.

```
# top
```

4. On Node 1 (**mariadb-lab-instance-1**), generate a heavy load against the database using **`mariadb-slap`**.

```
# mariadb-slap --auto-generate-sql --auto-generate-sql-add-autoincrement \
--commit=50 --concurrency=50 --engine=InnoDB \
--number-char-cols=10 --number-int-cols=5 \
--iterations=100
```

On Node 2 (**mariadb-lab-instance-2**), monitor the CPU usage of the system in the second terminal session, and take note of how long it took to complete the run of the **`mariadb-slap`** command by looking at the value for "**Average number of seconds to run all queries**" in the output of **`mariadb-slap`**. This value represents one iteration, and 100 iterations were performed so the overall query run time was 100 times greater.

5. On Node 2 (**mariadb-lab-instance-2**) in the first terminal session, configure more replica threads by increasing **`wsrep_slave_threads`** to **`2`**.

```
MariaDB [(none)]> SET GLOBAL wsrep_slave_threads = 2;
Query OK, 0 rows affected (0.001 sec)

MariaDB [(none)]> SHOW GLOBAL VARIABLES LIKE 'wsrep_slave_threads';
+---------------------+-------+
| Variable_name       | Value |
+---------------------+-------+
| wsrep_slave_threads | 2     |
+----------------------+-------+
1 row in set (0.001 sec)
```

6. On Node 2 (**mariadb-lab-instance-2**), execute the same **`mariadb-slap`** command from Step 4, and monitor the CPU usage and execution time. Note that the CPU usage may go higher because the two cores will now be almost fully engaged by the processing of the worksets. While the **`mariadb-slap`** command is executing, view the processes running on Node 2 (**mariadb-lab-instance-2**) with the **`SHOW PROCESSLIST`** statement, and observe that there are now two threads applying the changes.

```
MariaDB [(none)]> SHOW PROCESSLIST;
+------+-------------+-----------+------+---------+-------+--------------------+------------------+----------+
| Id   | User        | Host      | db   | Command | Time  | State              | Info             | Progress |
+------+-------------+-----------+------+---------+-------+--------------------+------------------+----------+
|    2 | system user |           | NULL | Sleep   |     0 | committing         | NULL             |    0.000 |
|    1 | system user |           | NULL | Sleep   | 90827 | wsrep aborter idle | NULL             |    0.000 |
|   29 | root        | localhost | seal | Query   |     0 | starting           | show processlist |    0.000 |
| 2030 | system user |           | NULL | Sleep   |     0 | committing         | NULL             |    0.000 |
+------+-------------+-----------+------+---------+-------+--------------------+------------------+----------+
4 rows in set (0.000 sec)
```

After the run is complete, compare the execution time for the first run with that of the second one by comparing the values for "**Average number of seconds to run all queries**" from the **`mariadb-slap`** output in both runs.

7. On Node 2 (**mariadb-lab-instance-2**), increase **`wsrep_slave_threads`** again, perform Step 4 and 5 and compare the execution time.

```
MariaDB [(none)]> SET GLOBAL wsrep_slave_threads = 4;
Query OK, 0 rows affected (0.000 sec)

MariaDB [(none)]> SHOW GLOBAL VARIABLES LIKE 'wsrep_slave_threads';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| wsrep_slave_threads  | 4     |
+----------------------+-------+
1 row in set (0.001 sec)
```

Lab Exercises for Lesson 5: Benchmarking

Assumptions

- You will execute all of the commands as the OS **`root`** user.

Lab Exercise 5-1: Benchmarking Cluster Performance

Overview

In this lab exercise you will benchmark the cluster's performance using the benchmarking tool **`sysbench`**. You can use **`sysbench --help`** to display the general command-line syntax and options.

Duration

This lab exercise should take approximately 15 minutes to complete.

Tasks

1. On Node 1 (**mariadb-lab-instance-1**) verify the version of **`sysbench`** that was installed at the beginning of the class. It should be version 1.0.17 or higher. If **`sysbench`** is not installed on your servers, then please refer to the installation instructions at the beginning of this workbook.

```
# sysbench --version
sysbench 1.0.17
```

2. The **`sysbench`** scripts have been installed in the **`/usr/share/`** directory.

```
# ls -l /usr/share/sysbench
total 60
-rwxr-xr-x. 1 root root  1452 Mar 15  2019 bulk_insert.lua
-rw-r--r--. 1 root root 14369 Mar 15  2019 oltp_common.lua
-rwxr-xr-x. 1 root root  1290 Mar 15  2019 oltp_delete.lua
-rwxr-xr-x. 1 root root  2415 Mar 15  2019 oltp_insert.lua
-rwxr-xr-x. 1 root root  1265 Mar 15  2019 oltp_point_select.lua
-rwxr-xr-x. 1 root root  1649 Mar 15  2019 oltp_read_only.lua
```

```
-rwxr-xr-x. 1 root root  1824 Mar 15  2019 oltp_read_write.lua
-rwxr-xr-x. 1 root root  1118 Mar 15  2019 oltp_update_index.lua
-rwxr-xr-x. 1 root root  1127 Mar 15  2019 oltp_update_non_index.lua
-rwxr-xr-x. 1 root root  1440 Mar 15  2019 oltp_write_only.lua
-rwxr-xr-x. 1 root root  1919 Mar 15  2019 select_random_points.lua
-rwxr-xr-x. 1 root root  2118 Mar 15  2019 select_random_ranges.lua
drwxr-xr-x. 4 root root    46 Jul 14 20:44 tests
```

3. On Node 1 (**mariadb-lab-instance-1**), connect to the database using the MariaDB client and create a new database named **sbtest**. This will be the database that **sysbench** will use for benchmarking. Also set the global variable **max_connections** to a higher value so that the benchmarking test can run successfully.

```
# mariadb

MariaDB [(none)]> CREATE DATABASE sbtest;
MariaDB [(none)]> SET GLOBAL max_connections = 1025;
MariaDB [(none)]> exit
```

4. Prepare for the benchmark by running the following **sysbench** command which will create a **sbtest1** table within the **sbtest** database and insert 10000 records plus a secondary index.

```
# sysbench oltp_common --db-driver=mysql --mysql-user=root --mysql-db=sbtest
prepare
sysbench 1.0.17 (using system LuaJIT 2.0.4)

Creating table 'sbtest1'...
Inserting 10000 records into 'sbtest1'
Creating a secondary index on 'sbtest1'...
```

5. Now run the benchmark with the following command.

```
# sysbench oltp_read_write --time=10 --db-driver=mysql --mysql-user=root
--mysql-db=sbtest --threads=16 --report-interval=1 --rate=1000 run
sysbench 1.0.17 (using system LuaJIT 2.0.4)

Running the test with following options:
Number of threads: 16
Target transaction rate: 1000/sec
Report intermediate results every 1 second(s)
Initializing random number generator from current time
Initializing worker threads...

Threads started!

[ 1s ] thds: 16 tps: 750.22 qps: 15227.85 (r/w/o: 10688.62/2555.93/1983.29)
lat (ms,95%): 231.53 err/s: 1.00 reconn/s: 0.00
[ 1s ] queue length: 204, concurrency: 16
[...]
9s ] queue length: 1906, concurrency: 16
[ 10s ] thds: 16 tps: 768.88 qps: 15366.66 (r/w/o: 10748.37/2858.57/1759.73)
lat (ms,95%): 2198.52 err/s: 0.00 reconn/s: 0.00
[ 10s ] queue length: 2135, concurrency: 16
SQL statistics:
    queries performed:
        read:                            109900
        write:                           28461
        other:                           18605
        total:                           156966
    transactions:                        7840   (782.46 per sec.)
    queries:                             156966 (15665.80 per sec.)
    ignored errors:                      10     (1.00 per sec.)
    reconnects:                          0      (0.00 per sec.)

General statistics:
    total time:                          10.0180s
    total number of events:              7840

Latency (ms):
         min:                                    5.47
         avg:                                 1077.01
         max:                                 2216.02
         95th percentile:                     2082.91
         sum:                              8443758.69

Threads fairness:
    events (avg/stddev):           490.0000/5.55
    execution time (avg/stddev):   527.7349/8.64
```

Take note of the statistics.

prepare is performed only once, but **run** can be performed multiple times. Doing so will reuse the originally prepared data and generate a complex **SELECT**, **INSERT**, **UPDATE**, **DELETE** load while maintaining transactions. A transaction will have multiple statements executed within, but the total number of queries per second will be much higher.

Generally, after performing **`prepare`**, server variables such as **`buffer_pool_size`** and **`tmp_table_size`** can be tuned, and encryption can be set. Then **`sysbench`** can be rerun using the **`run`** argument multiple times to see the impact of those variable changes.

For instance, the output above says
`transactions: 7840   (782.46 per sec.)`
but then
`queries: 156966 (15665.80 per sec.)`
which is much higher since each transaction has many smaller events being generated.

Repeat the above benchmark and increase the threads to 32, 64, 128 and so on to observe when performance starts to drop. That will be the peak capacity of the server.

6. When you are finished benchmarking, run the same command with the **`cleanup`** parameter to destroy the test data generated for the benchmarking. Once clean up is done, you will have to **`prepare`** once again before rerunning any benchmarks.

```
# sysbench oltp_common --db-driver=mysql --mysql-user=root --mysql-db=sbtest
cleanup
sysbench 1.0.17 (using system LuaJIT 2.0.4)

Dropping table 'sbtest1'...
Dropping table 'sbtest2'...
Dropping table 'sbtest3'...
Dropping table 'sbtest4'...
```

Lab Exercises for Lesson 6: Security

Assumptions

- You will execute all of the commands as the OS **root** user.

Lab Exercise 6-1: Setting Up Data-in-Transit Encryption

Overview

In this lab exercise you will secure Galera communication by creating a self-signed CA certificate, a CSR server certificate, and a client certificate using the **openssl** command-line tool. After the certificates are created, you will enable, configure and test data-in-transit encryption on the MariaDB server.

Duration

This lab exercise should take approximately 25 minutes to complete.

Tasks

1. On Node 1 (**mariadb-lab-instance-1**) create a working directory wherein the certificates will be created, and navigate to this directory. The rest of the commands will be done on Node 1 unless otherwise specified.

```
# mkdir /etc/my.cnf.d/certificates
# cd /etc/my.cnf.d/certificates
```

2. Create a self-signed CA certificate. The key and certificate will be created at one step.

```
# openssl req -x509 -nodes -newkey rsa -keyout ca-key.pem -out ca-cert.pem
-subj "/C=US/ST=Massachusetts/L=Boston/O=MariaDB/OU=Training/CN=mycluster"
```

3. When a MariaDB client validates the server certificate, the CN field from its subject is matched against the host to which it is connecting. No DNS resolution is performed at this point so, in order for validation to be successful, there must be an exact match. It is a good idea to initially decide whether to use DNS names or IP addresses, and maintain the same policy. This

is also why a multi-node cluster would need a separate certificate for each node. For this training class, IP addresses will be used except when generating a client side certificate.

4. Create a new Certificate Signing Request (CSR) key for the MariaDB server certificate and sign it with the CA key created above. Before running the fist command, replace the IP address in the CN field with your Node 1's internal IP (10.0.2.184, in this example).

```
# openssl req -newkey rsa -nodes -keyout server-key.pem -out server-req.pem
-subj "/C=US/ST=Massachusetts/L=Boston/O=MariaDB/OU=Training/CN=10.0.2.184"

# openssl x509 -req -in server-req.pem -CA ca-cert.pem -CAkey ca-key.pem
-set_serial 02 -out server-cert.pem
Signature ok
subject=/C=US/ST=Massachusetts/L=Boston/O=MariaDB/OU=Training/CN=10.0.2.184
Getting CA Private Key
```

5. Repeat the above steps to generate a certificate for the client side.

```
# openssl req -newkey rsa -nodes -keyout client-key.pem -out client-req.pem
-subj
"/C=US/ST=Massachusetts/L=Boston/O=MariaDB/OU=Training/CN=mycluster.mydomain
.com"

# openssl x509 -req -in client-req.pem -CA ca-cert.pem -CAkey ca-key.pem
-set_serial 03 -out client-cert.pem
Signature ok
subject=/C=US/ST=Massachusetts/L=Boston/O=MariaDB/OU=Training/CN=mycluster.m
ydomain.com
Getting CA Private Key
```

6. Verify the certificates before moving on to configuring and enabling data-in-transit encryption on the MariaDB server.

```
# openssl verify -CAfile ca-cert.pem server-cert.pem client-cert.pem
server-cert.pem: OK
client-cert.pem: OK
```

7. Make sure the certificate keys are only readable by the MariaDB system user.

```
# chown mysql:mysql /etc/my.cnf.d/certificates/*
# chmod 640 /etc/my.cnf.d/certificates/*key*
```

In this lab exercise both the client and server will run on the same OS instance, so the client will be able to see this directory. If the client connection were coming from a remote server then the client would need the CA certificate file, and its own certificate and private key.

8. Edit the custom server configuration file (**`/etc/my.cnf.d/z-custom.cnf`**) and under the **`[mariadb]`** section add the following variables shown below to enable encryption. Enabling encryption on the MariaDB server includes: activating encryption, setting paths to the CA certificate, server certificate and server key, and setting the allowed TLS versions to 1.2 and 1.3 only. If the CA uses a chain of certificates (which is the usual case), then all of the certificates must be present in the CA file as a bundle.

```
# vi /etc/my.cnf.d/z-custom.cnf

[mariadb]
...
ssl=1
ssl-ca=/etc/my.cnf.d/certificates/ca-cert.pem
ssl-cert=/etc/my.cnf.d/certificates/server-cert.pem
ssl-key=/etc/my.cnf.d/certificates/server-key.pem
tls_version=TLSv1.2,TLSv1.3
```

9. Restart the MariaDB Server to enable the changes.

```
# systemctl restart mariadb
```

10. Log on to the MariaDB database server, and verify that encryption has been enabled.

```
# mariadb

MariaDB [(none)]> SHOW VARIABLES like '%ssl%';
+---------------------+---------------------------------------------+
| Variable_name       | Value                                       |
+---------------------+---------------------------------------------+
| have_openssl        | YES                                         |
| have_ssl            | YES                                         |
| ssl_ca              | /etc/my.cnf.d/certificates/ca-cert.pem      |
| ssl_capath          |                                             |
| ssl_cert            | /etc/my.cnf.d/certificates/server-cert.pem |
| ssl_cipher          |                                             |
| ssl_crl             |                                             |
| ssl_crlpath         |                                           |
| ssl_key             | /etc/my.cnf.d/certificates/server-key.pem  |
| version_ssl_library | OpenSSL 1.0.2k-fips  26 Jan 2017          |
| wsrep_ssl_mode      | PROVIDER                                  |
+---------------------+-------------------------------------------+
11 rows in set (0.001 sec)

MariaDB [(none)]> exit
Bye
```

11. Create a user named **`encrypt_user`** to test that encryption is in place and working as expected. The use of a specific certificate will be enforced by setting its parameters, **`ISSUER`** and **`SUBJECT`**. A specific cipher suite will also be required by setting the **`CIPHER`** parameter. So that the host will default to any host (i.e., **`%`**), a host will not be specified in the **`USER CREATE`** statement.

In MariaDB Server, the **`%`** will not match a localhost so a **`user@'%'`** will not be allowed to connect from localhost. This means that a **`user@'%'`** will not be allowed to connect over either the Unix socket or a TCP connection to the loopback IP address.

Unlike server and client configurations where the TLS protocol version is specified, here the **`CIPHER`** parameter sets an exact cipher suite to use. This suite should be available under the chosen TLS version and should be supported by both the server and the client. To check for supported cipher suites for a particular TLS version, connect over TLS with the same version and a user that does not have an enforced cipher suite. Then check the **`Ssl_cipher_list`** session status variable.

If the database server runs with encryption, but the client does not have encryption parameters set, the client will still be able to connect over TLS. To enforce the client to connect over TLS with any client certificate, use the **`"REQUIRE X509"`** clause instead of the **`"REQUIRE CIPHER ... ISSUER ... SUBJECT"`** clause in the **`CREATE USER`** statement.

Since we want to match on a particular CA and client certificate, we need to first obtain their Subject lines in a format MariaDB recognises. The actual output may differ depending on the generated certificates.

```
# openssl x509 -in /etc/my.cnf.d/certificates/ca-cert.pem -noout -text |
grep Issuer | head -1 | sed -e 's/,* /\//g' | awk -F ':' '{print $2}'
/C=US/ST=Massachusetts/L=Boston/O=MariaDB/OU=Training/CN=mycluster
```

```
# openssl x509 -in /etc/my.cnf.d/certificates/client-cert.pem -noout -text |
grep Subject | head -1 | sed -e 's/,* /\//g' | awk -F ':' '{print $2}'
/C=US/ST=Massachusetts/L=Boston/O=MariaDB/OU=Training/CN=mycluster.mydomain.
com
```

Copy and paste the two subject lines from the output above into the respective `ISSUE` and `SUBJECT` clauses of the `CREATE USER` statement below.

```
# mariadb

MariaDB [(none)]> CREATE USER 'encrypt_user' IDENTIFIED BY 'MariaDB_123'
REQUIRE CIPHER 'DHE-RSA-AES256-GCM-SHA384'
ISSUER '/C=US/ST=Massachusetts/L=Boston/O=MariaDB/OU=Training/CN=mycluster'
SUBJECT
'/C=US/ST=Massachusetts/L=Boston/O=MariaDB/OU=Training/CN=mycluster.mydomain
.com';
Query OK, 0 rows affected (0.009 sec)
```

12. Check that encryption is enforced by attempting to connect to the database without setting TLS parameters. Set the `-h` command-line parameter to the FQDN of the host (for this training class, this will be the IP address of your node) so that the connection will go through the non-loopback IP address of the server.

Since MariaDB Server runs on the same host and a Unix socket is also available, you will enforce the connection to go through the non-loopback IP address of the server and enforce TCP protocol.

```
# mariadb -u encrypt_user -pMariaDB_123 -P 3306 -h 10.0.2.184 --protocol tcp
ERROR 1045 (28000): Access denied for user
'encrypt_user'@'ip-10-0-2-184.ec2.internal' (using password: YES)
```

Replace the IP address with your own IP address.

13. Attempt to connect to the database again, but this time pass all of the TLS parameters that have been set, and request the server certificate to be validated. The MariaDB Client validates the server certificate not only by its chain of trust, but also by matching the CN part of its DN field against the hostname to which the connection is being established. This is the reason the FQDN of the server was used when generating the server certificate.

```
# mariadb -u encrypt_user -pMariaDB_123 -P 3306 -h 10.0.2.184 \
    --protocol tcp \
    --ssl-ca=/etc/my.cnf.d/certificates/ca-cert.pem \
    --ssl-cert=/etc/my.cnf.d/certificates/client-cert.pem \
    --ssl-key=/etc/my.cnf.d/certificates/client-key.pem \
    --tls-version='TLSv1.2,TLSv1.3' \
    --ssl-verify-server-cert

MariaDB [(none)]> exit
Bye
```

14. [OPTIONAL] If time allows, or after class hours, generate separate certificates for Node 2 and Node 3 by following steps 1-18 first on Node 2 and then on Node 3.

Lab Exercise 6-2: Setting Up Data-at-Rest Encryption

Overview

In this lab exercise you will configure data-at-rest encryption on the cluster nodes.

Duration

This lab exercise should take approximately 15 minutes to complete.

Tasks

1. Log into the database on Node 1 (**mariadb-lab-instance-1**) using the MariaDB Client, and create a test database called `encrypt_test`, a test table called `encrypt_test1` in the database, and insert some test data into this new table.

```
# mariadb

MariaDB [(none)]> CREATE DATABASE encrypt_test;
Query OK, 1 row affected (0.006 sec)

MariaDB [(none)]> USE encrypt_test;
Database changed
MariaDB [encrypt_test]> CREATE TABLE encrypt_test1 (id int, txt
varchar(255));
Query OK, 0 rows affected (0.013 sec)
```

```
MariaDB [encrypt_test]> INSERT INTO encrypt_test1 VALUES (1, 'example');
Query OK, 1 row affected (0.003 sec)

MariaDB [encrypt_test]> exit
Bye
```

2. Stop the MariaDB server on Node 1 (**mariadb-lab-instance-1**), verify that the InnoDB tablespace file is unencrypted (i.e., a match on the search for example is found), then start the MariaDB server again.

```
# systemctl stop mariadb

# grep example /var/lib/mysql/encrypt_test/encrypt_test1.ibd
Binary file /var/lib/mysql/encrypt_test/encrypt_test1.ibd matches

# systemctl start mariadb
```

3. On Node 1 (**mariadb-lab-instance-1**), generate two random keys in a text file named **`keys.txt`**.

```
# echo -n "1;" > /etc/my.cnf.d/keys.txt
# openssl rand -hex 32|tr /a-z/ /A-Z/ >> /etc/my.cnf.d/keys.txt

# echo -n "2;" >> /etc/my.cnf.d/keys.txt
# openssl rand -hex 32|tr /a-z/ /A-Z/ >> /etc/my.cnf.d/keys.txt
```

4. Copy the keys into the same file at the same location on both Node 2 (**mariadb-lab-instance-2**) and Node 3 (**mariadb-lab-instance-3**).

To avoid SELinux labeling issues do not copy the file over scp. Instead open a blank file in the editor and paste the keys into the file. Save and exit the file.

5. On each node (**mariadb-lab-instance-1**, **mariadb-lab-instance-2** and **mariadb-lab-instance-3**), edit the custom configuration file to enable the encryption plugin with the generated key and turn on encryption of InnoDB files. The plugin cannot be loaded unless the configuration options are set, and the server cannot start up with the encryption configuration options set unless the plugin is loaded, so set the configuration options and load the plugin at the same time.

```
# vi /etc/my.cnf.d/z-custom.cnf

[mariadb]
...
plugin_load_add = file_key_management
file_key_management_filename = /etc/my.cnf.d/keys.txt
file_key_management = on
innodb_encrypt_tables = on
innodb_encrypt_log = on
innodb_encrypt_temporary_tables = on
innodb_encryption_threads = 4
```

6. Restart MariaDB Server on Node 1 (**mariadb-lab-instance-1**), then on Node 2 (**mariadb-lab-instance-2**), then on Node 3 (**mariadb-lab-instance-3**) (rolling restart).

```
# systemctl restart mariadb
```

7. On Node 1 (**mariadb-lab-instance-1**) use the MariaDB Client to connect to the database, create another test table called **`encrypt_test2`**, and load some test data into the **`encrypt_test2`** table.

```
# mariadb

MariaDB [(none)]> USE encrypt_test;

MariaDB [encrypt_test]> CREATE TABLE encrypt_test2 (id int, txt
varchar(255)) ENCRYPTED=YES;
Query OK, 0 rows affected (0.018 sec)

MariaDB [encrypt_test]> INSERT INTO encrypt_test2 VALUES (1, 'example');
Query OK, 1 row affected (0.004 sec)

MariaDB [encrypt_test]> exit
Bye
```

8. Checking that the file is encrypted requires the MariaDB server to be stopped. Once again stop the MariaDB server on Node 1 (**mariadb-lab-instance-1**), and verify that the InnoDB tablespace file is encrypted (i.e., a match on the search for **`example`** is not found). After verifying that the InnoDB tablespace file is encrypted, start the MariaDB Server again.

```
# systemctl stop mariadb
# grep example /var/lib/mysql/encrypt_test/encrypt_test2.ibd
#

# systemctl start mariadb
```

Lab Exercises for Lesson 7: Monitoring and Troubleshooting

Assumptions

- You will execute all of the commands as the OS **`root`** user.

Lab Exercise 7-1: Creating and Observing Ad-Hoc Conflict

Overview

In this lab exercise you will create an ad-hoc conflict and then observe the deadlock.

Duration

This lab exercise should take approximately 15 minutes to complete.

Tasks

1. On Node 1 (**mariadb-lab-instance-1**), create a table called **`c`** in the **`seal`** database, and insert some values into the table.

```
# mariadb seal

MariaDB [(seal)]> CREATE TABLE c (i INT PRIMARY KEY, j INT);
Query OK, 0 rows affected (0.011 sec)
```

2. On Node 1 (**mariadb-lab-instance-1**), insert records into the **`c`** table.

```
MariaDB [(seal)]> INSERT INTO c VALUES (1,0),(2,0);
Query OK, 2 rows affected (0.004 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

3. On Node 1 (**mariadb-lab-instance-1**), start a transaction using the **`BEGIN`** statement and then update table **`c`** so that **`j=1`** where **`i=1`**.

```
MariaDB [seal]> BEGIN;
```

```
Query OK, 0 rows affected (0.000 sec)

MariaDB [seal]> UPDATE c SET j=1 WHERE i=1;
Query OK, 1 row affected (0.000 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

4. On Node 2 (**mariadb-lab-instance-2**), update table **c** in the **seal** database so that **j=2** where **i=1**.

```
MariaDB [(none)]> USE seal;

MariaDB [(seal)]> UPDATE c SET j=2 WHERE i=1;
Query OK, 1 row affected (0.003 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

5. On Node 1 (**mariadb-lab-instance-1**), commit the transaction that you started in step 3.

```
MariaDB [seal]> COMMIT;
ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting
transaction
```

Deadlock! Any statement issued after the conflict will result in a deadlock error.

6. On Node 1 (**mariadb-lab-instance-1**), start a new transaction to update table **c** in the **seal** database so that **j=1** where **i=1**.

```
MariaDB [seal]> BEGIN;
Query OK, 0 rows affected (0.000 sec)

MariaDB [seal]> UPDATE c SET j=1 WHERE i=1;
Query OK, 1 row affected (0.000 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

7. On Node 2 (**mariadb-lab-instance-2**), update table **c** in the **seal** database so that **j=2** where **i=1**.

```
MariaDB [seal]> UPDATE c SET j=2 WHERE i=1;
Query OK, 0 rows affected (0.000 sec)
Rows matched: 1  Changed: 0  Warnings: 0
```

Note: This transaction does not update the data ("**Changed: 0**") so a workset will not be sent to the other nodes, and no conflict will be created with the ongoing transaction on Node 1.

8. On Node 2 (**mariadb-lab-instance-2**), query the records from table **c** in the **seal** database.

```
MariaDB [seal]> SELECT * FROM c;
+---+------+
| i | j    |
+---+------+
| 1 |    2 |
| 2 |    0 |
+---+------+
2 rows in set (0.000 sec)
```

9. On Node 1 (**mariadb-lab-instance-1**), query the records from table **c** in the **seal** database.

```
MariaDB [seal]> SELECT * FROM c;
+---+------+
| i | j    |
+---+------+
| 1 |    1 |
| 2 |    0 |
+---+------+
2 rows in set (0.000 sec)
```

10. On Node 1 (**mariadb-lab-instance-1**), commit the transaction that you started in Step 6, and then re-query the records from table **c** in the **seal** database.

```
MariaDB [seal]> COMMIT;
Query OK, 0 rows affected (0.004 sec)

MariaDB [seal]> SELECT * FROM c;
+---+------+
| i | j    |
+---+------+
| 1 |    1 |
| 2 |    0 |
+---+------+
2 rows in set (0.000 sec)
```

11. On Node 2 (**mariadb-lab-instance-2**), query the records from table **c** in the **seal** database.

```
MariaDB [seal]> SELECT * FROM c;
+---+------+
| i | j    |
+---+------+
| 1 |    1 |
| 2 |    0 |
+---+------+
2 rows in set (0.000 sec)
```

Lab Exercise 7-2: Monitoring the Galera Cache

Overview

In this exercise you will use **`mariadb-slap`** to generate artificial load on the cluster and observe the process of fill-up of the Galera Cache, which is set to the default size.

Duration

This lab exercise should take approximately 10 minutes to complete.

Tasks

1. For this lab exercise, you will use the **`gcache.sql`** file that was downloaded from the MariaDB git repository into the **`mariadb-training/sample_scripts`** directory on your class virtual machine(s).

On Node 1 (**mariadb-lab-instance-1**), copy the **`gcache.sql`** file from**`/mariadb-training/sample_scripts`** into the home directory.

```
# cd /home/centos
# cp /root/mariadb-training/sample_scripts/gcache.sql .
```

2. View the contents of the **gcache.sql** file. This script takes two readings of the size of the GCache one minute apart and then calculates how long the cache will last if the influx of new data remains at the same level. This is useful to calculate the allowed time that a cluster node may remain out of the cluster (e.g., for an upgrade or for a schema upgrade) without needing an SST upon rejoining the cluster.

```
# cat gcache.sql
SET @start := (SELECT SUM(VARIABLE_VALUE/1024/1024) FROM
information_schema.global_status WHERE VARIABLE_NAME LIKE 'WSREP%bytes');
DO sleep(60);
SET @end := (SELECT SUM(VARIABLE_VALUE/1024/1024) FROM
information_schema.global_status WHERE VARIABLE_NAME like 'WSREP%bytes');
SET @gcache := (SELECT
SUBSTRING_INDEX(SUBSTRING_INDEX(@@GLOBAL.wsrep_provider_options, 'gcache.size =
',-1), ';', 1));
SET @gcache_size := @gcache * 1;
SET @gcache_size_factor := if(right(@gcache,1)="M",1,1024);
SET @difference := round((@end - @start),2);
SELECT
       @difference as `MB/min`,
       @difference * 60 as `MB/hour`,
       @gcache as `gcache Size`,
       if(@difference = 0,0, round(@gcache_size*@gcache_size_factor/@difference))
       as `Time to full(minutes)`;
```

3. Open a second SSH session to the same Node 1 and become the root user.

4. On your first console on Node 1 (**mariadb-lab-instance-1**), execute the script from the command-line, passing the **`labtest`** user's credentials.

It will take one minute to run. Do not wait for it to complete, but switch to the second SSH session you have just opened.

```
# mariadb --user=labtest --password=MariaDB_123 < gcache.sql
```

5. On your second console of Node 1 (**mariadb-lab-instance-1**) and while the SQL script is running, use **`mariadb-slap`** to generate a test load on the cluster with the default Galera cache size.

```
# mariadb-slap -a --auto-generate-sql-add-autoincrement --commit=50
--concurrency=50 --engine=InnoDB --number-char-cols=10 --number-int-cols=5
--user=labtest --password=MariaDB_123 --iterations=100
Benchmark
       Running for engine InnoDB
       Average number of seconds to run all queries: 0.084 seconds
       Minimum number of seconds to run all queries: 0.073 seconds
       Maximum number of seconds to run all queries: 0.136 seconds
       Number of clients running queries: 50
       Average number of queries per client: 0
```

6. Switch back to your first console and wait one full minute for the results to be printed out. After obtaining the two measurements (as discussed in step 2), the script will output several values for the period: the input and output of data (measured in megabytes per minute and megabytes per hour); the actual Galera Cache size; and the time that the oldest entry will spend in the cache before being evicted (i.e., the maximum time that a cluster node may remain out of the cluster without needing an SST upon rejoining the cluster).

```
MB/min  MB/hour   gcache Size Time to full(minutes)
38.23   2293.8    128M    3
```

Lab Exercises for Lesson 8: Maintenance

Assumptions

- You will execute all of the commands as the OS **`root`** user.

Lab Exercise 8-1: Comparing Online and Offline DDL Statements

Overview

In this lab exercise you will compare how online and offline DDL statements affect MariaDB Enterprise Cluster. An offline DDL is a DDL which is executed using the traditional method of making a full copy of the tablespace file. An online DDL is one that can update the tablespace file without the need for a complete rewrite. This allows for an online DDL statement to complete much faster than an offline DDL statement. A writeset application in MariaDB Enterprise Cluster is serialized; a long running DDL effectively locks out the cluster and should either be avoided, or carefully managed. The next exercise, Lab 8-2: Testing a Rolling Schema Update, will cover alternative ways to handle a long-running DDL.

Duration

This lab exercise should take approximately 10 minutes to complete.

Tasks

1. You will need two terminal sessions to Node 1 (**mariadb-lab-instance-1**) of the cluster. If you do not have a second session, open another terminal session to Node 1 (**mariadb-lab-instance-1**).

2. On Node 1 (**mariadb-lab-instance-1**), go to the directory with training exercises and uncompress dump files for the employees database. Once they are uncompressed, load them to the cluster with the MariaDB Client.

```
# cd
# cd mariadb-training/sample_databases/employees/
# gzip -d *.gz
# mariadb < employees.sql
```

3. On both terminal sessions to Node 1 (**mariadb-lab-instance-1**), log on to the MariaDB server and switch to the newly created **employees** database.

```
# mariadb employees

MariaDB [employees]>
```

4. On the first session on Node 1 (**mariadb-lab-instance-1**), run an online DDL statement. The nature of the change allows InnoDB to handle it without the need to rewrite the tablespace file. Take note how quick the change is applied. Verify the change by comparing the table structure before and after.

```
MariaDB [employees]> DESCRIBE salaries;
+-----------+---------+------+-----+---------+-------+
| Field     | Type    | Null | Key | Default | Extra |
+-----------+---------+------+-----+---------+-------+
| emp_no    | int(11) | NO   | PRI | NULL    |       |
| salary    | int(11) | NO   |     | NULL    |       |
| from_date | date    | NO   | PRI | NULL    |       |
| to_date   | date    | NO   |     | NULL    |       |
+-----------+---------+------+-----+---------+-------+
4 rows in set (0.001 sec)

MariaDB [employees]> ALTER TABLE salaries ADD COLUMN name varchar(32)
DEFAULT 'Unknown';
Query OK, 0 rows affected (0.288 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [employees]> DESCRIBE salaries;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| emp_no    | int(11)     | NO   | PRI | NULL    |       |
| salary    | int(11)     | NO   |     | NULL    |       |
| from_date | date        | NO   | PRI | NULL    |       |
| to_date   | date        | NO   |     | NULL    |       |
| name      | varchar(32) | YES  |     | Unknown |       |
+-----------+-------------+------+-----+---------+-------+
5 rows in set (0.078 sec)
```

5. In the second session to Node 1 (**mariadb-lab-instance-1**), run a statement to check the status of the salaries table.

```
MariaDB [employees]> SHOW OPEN TABLES LIKE 'salaries';
+-----------+----------+--------+-------------+
| Database  | Table    | In_use | Name_locked |
+-----------+----------+--------+-------------+
| employees | salaries |      0 |           0 |
+-----------+----------+--------+-------------+
1 row in set (0.000 sec)
```

6. In the first session to Node 1 (**mariadb-lab-instance-1**), start a long-running (offline) DDL. Due to the nature of the change, InnoDB will need to rewrite the tablespace file, taking a much longer time than before.

```
MariaDB [employees]> ALTER TABLE salaries MODIFY COLUMN salary
DECIMAL(20,10);
Query OK, 2844047 rows affected (17.857 sec)
Records: 2844047  Duplicates: 0  Warnings: 0
```

At the same time, in the second session on Node 1, repeat the statement to check the status of the salaries table and observe the difference.

```
MariaDB [employees]> SHOW OPEN TABLES LIKE 'salaries';
+-----------+----------+--------+-------------+
| Database  | Table    | In_use | Name_locked |
+-----------+----------+--------+-------------+
| employees | salaries |      1 |           0 |
+-----------+----------+--------+-------------+
1 row in set (0.000 sec)
```

7. In the first session to Node 1 (**mariadb-lab-instance-1**), once the DDL completes, validate the change.

```
MariaDB [employees]> DESCRIBE salaries;
+-----------+----------------+------+-----+---------+-------+
| Field     | Type           | Null | Key | Default | Extra |
+-----------+----------------+------+-----+---------+-------+
| emp_no    | int(11)        | NO   | PRI | NULL    |       |
| salary    | decimal(20,10) | YES  |     | NULL    |       |
| from_date | date           | NO   | PRI | NULL    |       |
| to_date   | date           | NO   |     | NULL    |       |
| name      | varchar(32)    | YES  |     | Unknown |       |
+-----------+---------------+------+-----+---------+-------+
5 rows in set (0.078 sec)
```

Lab Exercise 8-2: Testing a Rolling Schema Update

Overview

In this lab exercise you will test upgrading a schema using the rolling schema upgrade (RSU) method. This method applies the changes to one node at a time, allowing the rest of the cluster to bear any ongoing load while letting a long-running DDL complete without locking the cluster.

Duration

This lab exercise should take approximately 15 minutes to complete.

Tasks

1. Open a terminal session to each of the three nodes (**mariadb-lab-instance-1**, **mariadb-lab-instance-2** and **mariadb-lab-instance-3**).

2. On Node 1 (**mariadb-lab-instance-1**) create a new table called **`t_rsu`** in the **`seal`** database.

```
MariaDB [(none)]> USE seal;

MariaDB [(seal)]> CREATE TABLE t_rsu (i int, j int);
Query OK, 0 rows affected (0.016 sec)

MariaDB [(seal)]> DESCRIBE t_rsu;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| i     | int(11) | YES  |     | NULL    |       |
| j     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.001 sec)
```

3. On the other two nodes (**mariadb-lab-instance-2** and **mariadb-lab-instance-3**), confirm that the table was created.

```
MariaDB [(seal)]> DESCRIBE t_rsu;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| i     | int(11) | YES  |     | NULL    |       |
| j     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.002 sec)
```

4. On Node 1 (**mariadb-lab-instance-1**) change the **`wsrep_OSU_method`** server variable to **RSU**.

```
MariaDB [(seal)]> SHOW VARIABLES LIKE 'wsrep_OSU_method';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| wsrep_osu_method | TOI   |
+------------------+-------+
1 row in set (0.001 sec)

MariaDB [(seal)]> SET wsrep_OSU_method=RSU;
Query OK, 0 rows affected (0.000 sec)

MariaDB [(seal)]> SHOW VARIABLES LIKE 'wsrep_OSU_method';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| wsrep_osu_method | RSU   |
+------------------+-------+
1 row in set (0.001 sec)
```

5. On Node 1 (**mariadb-lab-instance-1**), execute a DDL statement while the rolling schema upgrade method is in place, and then check the table to see that the change has been applied.

```
MariaDB [(seal)]> ALTER TABLE t_rsu DROP COLUMN j;
Query OK, 0 rows affected (0.018 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
MariaDB [(seal)]> DESCRIBE t_rsu;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| i     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
1 row in set (0.001 sec)
```

6. On Node 2 (**mariadb-lab-instance-2**) and Node 3 (**mariadb-lab-instance-3**), view the layout of the same table (**`seal.t_rsu`**) and observe the fact that the table is unchanged.

```
MariaDB [(seal)]> DESCRIBE t_rsu;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| i     | int(11) | YES  |     | NULL    |       |
| j     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.001 sec)
```

7. On Node 2 (**mariadb-lab-instance-2**), change the **`wsrep_OSU_method`** server variable to **RSU**.

```
MariaDB [(seal)]> SET wsrep_OSU_method=RSU;
Query OK, 0 rows affected (0.000 sec)

MariaDB [(none)]> SHOW VARIABLES LIKE 'wsrep_OSU_method';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| wsrep_osu_method | RSU   |
+------------------+-------+
1 row in set (0.001 sec)
```

8. On Node 2 (**mariadb-lab-instance-2**), check that the table layout is unchanged then alter the **`c`** table by dropping column **`j`**. Verify the change to the table on the same node (**mariadb-lab-instance-2**).

```
MariaDB [(seal)]> DESCRIBE t_rsu;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+--------+---------+------+-----+---------+-------+
| i      | int(11) | YES  |     | NULL    |       |
| j      | int(11) | YES  |     | NULL    |       |
+--------+---------+------+-----+---------+-------+
2 rows in set (0.001 sec)

MariaDB [(seal)]> ALTER TABLE t_rsu DROP COLUMN j;
Query OK, 0 rows affected (0.015 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [(seal)]> DESCRIBE t_rsu;
+--------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+--------+---------+------+-----+---------+-------+
| i      | int(11) | YES  |     | NULL    |       |
+--------+---------+------+-----+---------+-------+
1 row in set (0.001 sec)
```

9. On Node 3 (**mariadb-lab-instance-3**), check that the table layout is unchanged then alter the **c** table by dropping column **m**. Verify the change to the table on the same node (**mariadb-lab-instance-3**).

```
MariaDB [(seal)]> DESCRIBE t_rsu;
+--------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+--------+---------+------+-----+---------+-------+
| i      | int(11) | YES  |     | NULL    |       |
| j      | int(11) | YES  |     | NULL    |       |
+--------+---------+------+-----+---------+-------+
2 rows in set (0.001 sec)

MariaDB [(seal)]> ALTER TABLE t_rsu DROP COLUMN j;
Query OK, 0 rows affected (0.029 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [(seal)]> DESCRIBE t_rsu;
+--------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+--------+---------+------+-----+---------+-------+
| i      | int(11) | YES  |     | NULL    |       |
+--------+---------+------+-----+---------+-------+
1 row in set (0.001 sec)
```

10. On each of the three three nodes (**mariadb-lab-instance-1**, **mariadb-lab-instance-2** and **mariadb-lab-instance-3**), restore the `wsrep_OSU_method` method to `TOI` and verify the change with the `SHOW VARIABLES LIKE 'wsrep_OSU_method'` statement. To restore the `wsrep_OSU_method` method to `TOI`, you can close and reopen the session since only the session variable was changed. Remember that session variables are only valid for the current session.

```
MariaDB [(seal)]> exit
Bye

# mariadb

MariaDB [(none)]> SHOW VARIABLES LIKE 'wsrep_OSU_method';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| wsrep_osu_method | TOI   |
+------------------+-------+
1 row in set (0.001 sec)
```

Lab Exercises for Lesson 10: Advanced Features

Assumptions

- You will execute all of the commands as the OS `root` user.

Lab Exercise 10-1: Setting Up Cluster Notifications

Overview

In this lab exercise you will enable the cluster notification command and then cause the default cluster notification script to be executed by changing the cluster membership. MariaDB Enterprise Cluster provides a default script called `wsrep_notify.sh` as a starting point to use in handling notifications and as you in writing your own custom notification script.

Duration

This lab exercise should take approximately 10 minutes to complete.

Tasks

1. On Node 2 (**mariadb-lab-instance-2**), open the `z-custom.cnf` configuration file.

```
# vi /etc/my.cnf.d/z-custom.cnf
```

2. On Node 2 (**mariadb-lab-instance-2**), configure the server to call the notification handler by adding the `wsrep_notify_cmd` option under the `[galera]` section of the configuration file. Set the value of the `wsrep_notify_cmd` to the path where the default cluster notification script, `wsrep_notify.sh`, is located. The typical default location for this script is in `/usr/bin/` and may be helpful as a starting point. You should use whatever script is most appropriate for your deployment.

```
[galera]
...
wsrep_notify_cmd = /usr/bin/wsrep_notify.sh
```

3. For this lab exercise, you will use the `wsrep_notify.sh` script that was downloaded from the MariaDB git repository into the `/root/mariadb-training/sample_scripts` directory on your class virtual machine(s).

On Node 2 (**mariadb-lab-instance-2**), copy the `wsrep_notify.sh` script from `/root/mariadb-training/sample_scripts` into the `/usr/bin/` directory.

```
# cd
# cp -p mariadb-training/sample_scripts/wsrep_notify.sh /usr/bin/
```

4. View the contents of the `wsrep_notify.sh` file.

```
# cat /usr/bin/wsrep_notify.sh
#!/bin/bash
# Command to call when node status or cluster membership changes.
# Some or all of the following options will be passed:
# --status  - New status of node
# --uuid    - UUID of cluster
# --primary - Whether component is Primary ("yes" or "no")
# --members - Comma-separated list of members
# --index   - Index of node in list

echo "$@" >> /var/tmp/wsrep_notifications.log
```

5. On Node 2 (**mariadb-lab-instance-2**) ensure the notification script is owned by the MariaDB system user and is executable.

```
# chown mysql:mysql /usr/bin/wsrep_notify.sh

# chmod 700 /usr/bin/wsrep_notify.sh
```

6. Restart Node 2 (**mariadb-lab-instance-2**) to let the **`wsrep_notify_cmd`** setting take effect.

```
# systemctl restart mariadb
```

7. On Node 2 (**mariadb-lab-instance-2**) tail the notifications log so that any messages written to the log can be observed.

```
# tail -f /var/tmp/wsrep_notifications.log
```

8. Now cause a change in the cluster state by shutting down MariaDB on one of the other nodes. In our lab we will do this on Node 1 (**mariadb-lab-instance-1**). So, on Node 1, stop the MariaDB server.

```
# systemctl stop mariadb
```

9. You should observe that Node 2 (**mariadb-lab-instance-2**) executes the script for each change in cluster membership and node status. These status changes can be utilized for configuring load balancers, raising alerts or scripting for any other situation where you need your infrastructure to respond to changes to the cluster.

10. On Node 1 (**mariadb-lab-instance-1**), start the MariaDB server.

```
# systemctl start mariadb
```