PERCONA

www.percona.com

Percona XtraDB Cluster Documentation

Release 5.7.14-26.17

September 28, 2016

CONTENTS

Percona XtraDB Cluster is a high-availability solution for MySQL. It ensures the highest possible uptime, prevents data loss, and provides linear scalability for a growing environment.

Features of *Percona XtraDB Cluster* include:

- **Synchronous replication**: Data is written to all nodes simultaneously, or not written at all if it fails even on a single node.
- **Multi-master replication**: Any node can trigger a data update.
- **True parallel replication**: Multiple threads on slave performing replication on row level.
- **Automatic node provisioning**: You simply add a node and it automatically syncs.
- **Data consistency**: No more unsynchronized nodes.

Percona XtraDB Cluster is fully compatible with MySQL Server Community Edition, Percona Server, and MariaDB in the following sense:

- **Data compatibility**: You can use data created by any MySQL variant.
- **Application compatibility**: There is no or minimal application changes required.

CHAPTER

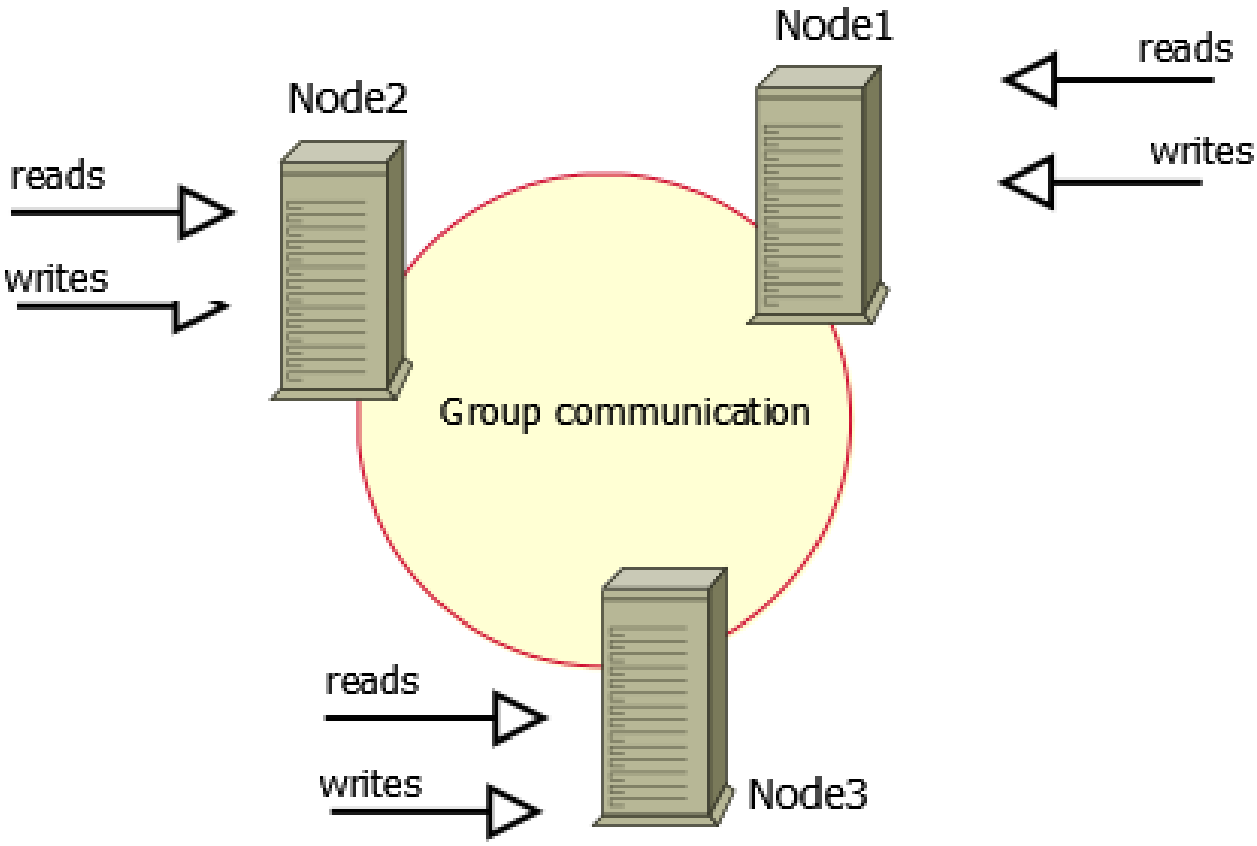
ONE

INTRODUCTION

1.1 About Percona XtraDB Cluster

Percona XtraDB Cluster is a fully open-source high-availability solution for MySQL.

A *cluster* consists of *nodes*, where each node contains the same set of data synchronized accross nodes. The recommended configuration is to have at least 3 nodes, but you can have 2 nodes as well. Each node is a regular MySQL Server instance (for example, Percona Server). You can convert an existing MySQL Server instance to a node and run the cluster using this node as a base. You can also detach any node from the cluster and use it as a regular MySQL Server instance.



Benefits:

- When you execute a query, it is executed locally on the node. All data is available locally, no need for remote access.
- No central management. You can loose any node at any point of time, and the cluster will continue to function without any data loss.
- Good solution for scaling a read workload. You can put read queries to any of the nodes.

Drawbacks:

- Overhead of provisioning new node. When you add a new node, it has to copy the full data set from one of existing nodes. If it is 100GB, it copies 100GB.
- This can't be used as an effective write scaling solution. There might be some improvements in write throughput when you run write traffic to 2 nodes versus all traffic to 1 node, but you can't expect a lot. All writes still have to go on all nodes.
- You have several duplicates of the data, for 3 nodes you have 3 duplicates.

1.1.1 Components

Percona XtraDB Cluster is based on Percona Server running with the XtraDB storage engine. It uses the Galera library, which is an implementation of the write set replication (wsrep) API developed by Codership Oy.

1.2 Percona XtraDB Cluster Limitations

The following limitations apply to Percona XtraDB Cluster:

- Replication works only with *InnoDB* storage engine. Any writes to tables of other types, including system (`mysql.*`) tables, are not replicated. However, `DDL` statements are replicated in statement level, and changes to `mysql.*` tables will get replicated that way. So you can safely issue `CREATE USER...`, but issuing `INSERT INTO mysql.user...` will not be replicated. You can enable experimental *MyISAM* replication support using the `wsrep_replicate_myisam` variable.
- Unsupported queries:
  - `LOCK TABLES` and `UNLOCK TABLES` is not supported in multi-master setups
  - Lock functions, such as `GET_LOCK()`, `RELEASE_LOCK()`, and so on
- Query log cannot be directed to table. If you enable query logging, you must forward the log to a file:

  ```
  log_output = FILE
  ```

  Use `general_log` and `general_log_file` to choose query logging and the log file name.
- Maximum allowed transaction size is defined by the `wsrep_max_ws_rows` and `wsrep_max_ws_size` variables. `LOAD DATA INFILE` processing will commit every 10 000 rows. So large transactions due to `LOAD DATA` will be split to series of small transactions.
- Due to cluster-level optimistic concurrency control, transaction issuing `COMMIT` may still be aborted at that stage. There can be two transactions writing to the same rows and committing in separate Percona XtraDB Cluster nodes, and only one of the them can successfully commit. The failing one will be aborted. For cluster-level aborts, Percona XtraDB Cluster gives back deadlock error code:

  ```
  (Error: 1213 SQLSTATE: 40001  (ER_LOCK_DEADLOCK)).
  ```

- XA transactions are not supported due to possible rollback on commit.

- The write throughput of the whole cluster is limited by weakest node. If one node becomes slow, the whole cluster slows down. If you have requirements for stable high performance, then it should be supported by corresponding hardware.
- The minimal recommended size of cluster is 3 nodes. The 3rd node can be an arbitrator.
- InnoDB fake changes feature is not supported.
- `enforce_storage_engine=InnoDB` is not compatible with `wsrep_replicate_myisam=OFF` (default).
- The `binlog_rows_query_log_events` variable is not supported.
- Backup locks used during SST or with XtraBackup can crash. On donor, either use `inno-backup-opts='--no-backup-locks'` under `[sst]` in `my.cnf` or set `FORCE_FTWRL=1` in `/etc/sysconfig/mysql` (or `/etc/sysconfig/mysql.%i` for corresponding unit/service) for CentOS/RHEL or `/etc/default/mysql` in Debian/Ubuntu. You can also use `rsync` as an alternate SST method.
- When running Percona XtraDB Cluster in cluster mode, avoid `ALTER TABLE ... IMPORT/EXPORT` workloads. It can lead to node inconsistency if not executed in sync on all nodes.

CHAPTER

TWO

INSTALLATION

2.1 Installing Percona XtraDB Cluster

Percona XtraDB Cluster supports most 64-bit Linux distributions. Percona provides packages for popular DEB-based and RPM-based distributions:

- Debian 7 (“wheezy”)
- Debian 8 (“jessie”)
- Ubuntu 12.04 LTS (Precise Pangolin)
- Ubuntu 14.04 LTS (Trusty Tahr)
- Ubuntu 15.10 (Wily Werewolf)
- Ubuntu 16.04 (Xenial Xerus)
- Red Hat Enterprise Linux / CentOS 6
- Red Hat Enterprise Linux / CentOS 7

2.1.1 Prerequisites

Percona XtraDB Cluster requires the following ports for communication: 3306, 4444, 4567, 4568. Make sure that these ports are not blocked by firewall or used by other software.

Percona XtraDB Cluster does not work with `SELinux` and `AppArmor` security modules. Make sure these modules are disabled.

There are several MySQL configuration options that are mandatory for running Percona XtraDB Cluster. These include the path to the Galera wsrep library, connection URL for other nodes in the cluster, and method used for state snapshot transfer (SST). Make sure that the binlog format is set to `ROW`, the default storage engine is InnoDB (MyISAM has only experimental support), and set the mode of InnoDB autoincrement locks to `2`. The following is an example of the `my.cnf` file:

```
[mysqld]

wsrep_provider=/usr/lib64/libgalera_smm.so
wsrep_cluster_address=gcomm://10.11.12.206
wsrep_slave_threads=8
wsrep_sst_method=rsync
binlog_format=ROW
default_storage_engine=InnoDB
innodb_autoinc_lock_mode=2
```

Note: If the SST method is not `rsync`, specify SST credentials using `wsrep_sst_auth=<user>:<password>`

Note: If you enable binary logging by setting `log-bin=mysql-bin`, you should also provide a unique `server-id` for each node.

2.1.2 Installation Guides

It is recommended to install Percona XtraDB Cluster from official Percona repositories using the corresponding tool for your system:

- *Install using apt* if you are running Debian or Ubuntu
- *Install using yum* if you are running Red Hat Enterprise Linux or CentOS

Note: You can also download packages from the Percona website and install them manually using `dpkg` or `rpm`.

If you want to build and run Percona XtraDB Cluster from source, see *Compiling and Installing from Source Code*.

Installing Percona XtraDB Cluster on Debian and Ubuntu

Percona provides `.deb` packages for 64-bit versions of the following distributions:

- Debian 7 ("wheezy")
- Debian 8 ("jessie")
- Ubuntu 12.04 LTS (Precise Pangolin)
- Ubuntu 14.04 LTS (Trusty Tahr)
- Ubuntu 15.10 (Wily Werewolf)
- Ubuntu 16.04 (Xenial Xerus)

Note: Percona XtraDB Cluster should work on other DEB-based distributions, but it is tested only on platforms listed above.

The packages are available in the official Percona software repositories and on the download page. It is recommended to intall Percona XtraDB Cluster from repositories using `apt`.

Installing from Repositories

1. Fetch the repository packages from Percona web:

   ```
   wget https://repo.percona.com/apt/percona-release_0.1-4.$(lsb_release -sc)_all.deb
   ```

2. Install the downloaded package with **dpkg**. To do that, run the following command as root or with **sudo**:

```
sudo dpkg -i percona-release_0.1-4.$(lsb_release -sc)_all.deb
```

Once you install this package, the Percona repositories should be added. You can check the repository configuration in the `/etc/apt/sources.list.d/percona-release.list` file.

3. Update the local cache:

```
sudo apt-get update
```

4. Install the server package:

```
sudo apt-get install percona-xtradb-cluster-57
```

Note: Alternatively, you can install the `percona-xtradb-cluster-full-57` meta package, which contains the following additional packages:

- `percona-xtradb-cluster-test-5.7`
- `percona-xtradb-cluster-5.7-dbg`
- `percona-xtradb-cluster-garbd-3.x`
- `percona-xtradb-cluster-galera-3.x-dbg`
- `percona-xtradb-cluster-garbd-3.x-dbg`
- `libmysqlclient18`

Testing and Experimental Repositories Percona offers pre-release builds from the testing repo, and early-stage development builds from the experimental repo. To enable them, add either `testing` or `experimental` at the end of the Percona repository definition in your repository file (by default, `/etc/apt/sources.list.d/percona-release.list`).

For example, if you are running Debian 8 ("jessie") and want to install the latest testing builds, the definitions should look like this:

```
deb http://repo.percona.com/apt jessie main testing
deb-src http://repo.percona.com/apt jessie main testing
```

If you are running Ubuntu 14.04 LTS (Trusty Tahr) and want to install the latest experimental builds, the definitions should look like this:

```
deb http://repo.percona.com/apt trusty main experimental
deb-src http://repo.percona.com/apt trusty main experimental
```

Pinning the Packages If you want to pin your packages to avoid upgrades, create a new file `/etc/apt/preferences.d/00percona.pref` and add the following lines to it:

```
Package: *
Pin: release o=Percona Development Team
Pin-Priority: 1001
```

For more information about pinning, refer to the official Debian Wiki.

Installing Percona XtraDB Cluster on Red Hat Enterprise Linux and CentOS

Percona provides `.rpm` packages for 64-bit versions of the following distributions:

- Red Hat Enterprise Linux / CentOS 6
- Red Hat Enterprise Linux / CentOS 7

Note: Percona XtraDB Cluster should work on other RPM-based distributions (for example, Amazon Linux AMI and Oracle Linux), but it is tested only on platforms listed above.

The packages are available in the official Percona software repositories and on the download page. It is recommended to intall Percona XtraDB Cluster from the official repository using `yum`.

Prerequisites

In CentOS, remove the `mysql-libs` package before installing Percona XtraDB Cluster, to avoid possible conflicts. The Percona XtraDB Cluster package will correctly resolve this dependency.

Use the EPEL repository to install the `socat` package before installing Percona XtraDB Cluster.

Installing from Percona Repository

1. Install the Percona repository package:

   ```
   sudo yum install http://www.percona.com/downloads/percona-release/redhat/0.1-3/percona-release-0
   ```

   Confirm installation and you should see the following if successful:

   ```
   Installed:
     percona-release.noarch 0:0.1-3

   Complete!
   ```

   Note: Red Hat Enterprise Linux and CentOS 5 do not support installing packages directly from the remote location. Download the Percona repository package first and install it manually using **rpm**:

   ```
   wget http://www.percona.com/downloads/percona-release/redhat/0.1-3/percona-release-0.1-3.noarch.
   rpm -ivH percona-release-0.1-3.noarch.rpm
   ```

   ---

2. Check that the packages are available:

   ```
   yum list | grep Percona-XtraDB-Cluster
   ```

   You should see output similar to the following:

   ```
   Percona-XtraDB-Cluster-57.x86_64           1:5.7.11-25.14.2.el7        percona-release-x86_64
   Percona-XtraDB-Cluster-57-debuginfo.x86_64 1:5.7.11-25.14.2.el7        percona-release-x86_64
   ```

3. Install the Percona XtraDB Cluster packages:

   ```
   sudo yum install Percona-XtraDB-Cluster-57
   ```

Note: Alternatively, you can install the `Percona-XtraDB-Cluster-full-57` meta package, which contains the following additional packages:

- `Percona-XtraDB-Cluster-devel-57`
- `Percona-XtraDB-Cluster-test-57`
- `Percona-XtraDB-Cluster-debuginfo-57`
- `Percona-XtraDB-Cluster-galera-3-debuginfo`
- `Percona-XtraDB-Cluster-shared-57`

Testing and Experimental Repositories Percona offers pre-release builds from the testing repo, and early-stage development builds from the experimental repo. You can enable either one in the Percona repository configuration file `/etc/yum.repos.d/percona-release.repo`. There are three sections in this file, for configuring corresponding repositories:

- stable release
- testing
- experimental

The latter two repositories are disabled by default.

If you want to install the latest testing builds, set `enabled=1` for the following entries:

```
[percona-testing-$basearch]
[percona-testing-noarch]
```

If you want to install the latest experimental builds, set `enabled=1` for the following entries:

```
[percona-experimental-$basearch]
[percona-experimental-noarch]
```

Compiling and Installing from Source Code

If you want to compile Percona XtraDB Cluster, you can find the source code on GitHub. Before you begin, make sure that the following packages are installed:

	apt	yum
Git	`git`	`git`
SCons	`scons`	`scons`
GCC	`gcc`	`gcc`
g++	`g++`	`gcc-c++`
OpenSSL	`openssl`	`openssl`
Check	`check`	`check`
CMake	`cmake`	`cmake`
Bison	`bison`	`bison`
Boost	`libboost-all-dev`	`boost-devel`
Asio	`libasio-dev`	`asio-devel`
Async I/O	`libaio-dev`	`libaio-devel`
ncurses	`libncurses5-dev`	`ncurses-devel`
Readline	`libreadline-dev`	`readline-devel`
PAM	`libpam-dev`	`pam-devel`
socat	`socat`	`socat`

You will likely have all or most of the packages already installed. If you are not sure, run one of the following commands to install any missing dependencies:

- For Debian or Ubuntu:

```
$ sudo apt-get install -y git scons gcc g++ openssl check cmake bison \
libboost-all-dev libasio-dev libaio-dev libncurses5-dev libreadline-dev \
libpam-dev socat
```

- For Red Hat Enterprise Linux or CentOS:

```
$ sudo yum install -y git scons gcc gcc-c++ openssl check cmake bison \
boost-devel asio-devel libaio-devel ncurses-devel readline-devel pam-devel socat
```

To compile Percona XtraDB Cluster from source code:

1. Clone the Percona XtraDB Cluster repository:

   ```
   $ git clone https://github.com/percona/percona-xtradb-cluster.git
   ```

   Note: You have to clone the latest repository or update it to the latest state. Old codebase may not be compatible with the build script.

2. Check out the `5.7` branch.
3. Initialize the submodule:

   ```
   $ git submodule init && git submodule update
   ```

4. Run the build script `./build-ps/build-binary.sh`. By default, it will build into the current directory, but you can specify another target output directory. For example, if you want to build into `./pxc-build`, run the following:

   ```
   $ mkdir ./pxc-build
   $ ./build-ps/build-binary.sh ./pxc-build
   ```

2.2 Upgrading Percona XtraDB Cluster

This guide describes the procedure for upgrading Percona XtraDB Cluster version 5.6 to the latest version 5.7 without downtime (*rolling upgrade*). This means that replication between nodes will continue during the upgrade.

Rolling upgrade from earlier versions to 5.7 is not supported. If you are running Percona XtraDB Cluster version 5.5, it is recommended to shut down all nodes, then remove and re-create the cluster from scratch. Alternatively, you can perform a rolling upgrade from PXC 5.5 to 5.6, and then follow the current procedure to upgrade from 5.6 to 5.7.

The following documents contain details about relevant changes in the 5.7 series of MySQL and Percona Server. Make sure you deal with any incompatible features and variables mentioned in these documents when upgrading to Percona XtraDB Cluster 5.7.

- Changed in Percona Server 5.7
- Upgrading MySQL
- Upgrading from MySQL 5.6 to 5.7

To upgrade the cluster, follow these steps for each node:

1. Make sure that all nodes are synchronized.
2. Stop the `mysql` service:

```
$ sudo service mysql stop
```

3. Remove existing packages and install version 5.7. For more information, see *Installing Percona XtraDB Cluster*.

   For example, if you have Percona software repositories configured, you might use the following commands:

   - On CentOS or RHEL:

     ```
     $ sudo yum remove 'Percona-XtraDB-Cluster-56*'
     $ sudo yum install Percona-XtraDB-Cluster-57
     ```

   - On Debian or Ubuntu:

     ```
     $ sudo apt-get remove percona-xtradb-cluster-56*
     $ sudo apt-get install percona-xtradb-cluster-57
     ```

Note: In case of Debian or Ubuntu, the `mysql` service starts automatically after install. Stop the service:

```
$ sudo service mysql stop
```

---

4. Back up `grastate.dat`, so that you can restore it if it is corrupted or zeroed out due to network issue.
5. Start the node outside the cluster by setting the `wsrep_provider` variable to `none`. For example:

   ```
   sudo mysqld --skip-grant-tables --user=mysql --wsrep-provider='none'
   ```

6. Open another session and run `mysql_upgrade`.
7. When the upgrade is done, stop `mysqld`.

Note: On CentOS, the `my.cnf` configuration file is renamed to `my.cnf.rpmsave`. Make sure to rename it back before joining the upgraded node back to the cluster.

---

8. Start the node with `pxc_strict_mode` variable set to `PERMISSIVE`. By default, *PXC Strict Mode* is set to `ENFORCING`, which will deny any unsupported operations and may halt the server upon encountering a failed validation. In `PERMISSIVE` mode it will log warnings and continue running as normal.

   ```
   $ sudo mysqld --pxc-strict-mode=PERMISSIVE
   ```

9. Check the log for any experimental or unsupported features that might have been encountered.
10. If you fixed all incompatibilities releaved by *PXC Strict Mode* validations, you can set the `pxc_strict_mode` variable to `ENFORCING`:

    ```
    mysql> SET pxc_strict_mode=ENFORCING;
    ```

Note: It is highly recommended to run with the default `ENFORCING` mode and ensure that the workload passes all validations concerning experimental and unsupported features.

---

11. Repeat this procedure for the next node in the cluster until you upgrade all nodes.

CHAPTER

THREE

FEATURES

3.1 High Availability

In a basic setup with 3 nodes, Percona XtraDB Cluster will continue to function if you take any of the nodes down. At any point in time, you can shut down any node to perform maintenance or make configuration changes. Even in unplanned situations (like a node crashing or if it becomes unavailable over the network), the Percona XtraDB Cluster will continue to work and you'll be able to run queries on working nodes.

If there were changes to data while a node was down, there are two options that the node may use when it joins the cluster again:

- **State Snapshot Transfer** (SST) is when all data is copied from one node to another.

  SST is usually used when a new node joins the cluster and receives all data from an existing node. There are three methods of SST available in Percona XtraDB Cluster:

  - **mysqldump**
  - **rsync**
  - **xtrabackup**.

  The downside of `mysqldump` and `rsync` is that your cluster becomes **READ-ONLY** while data is being copied (SST applies the `FLUSH TABLES WITH READ LOCK` command).

  SST using `xtrabackup` does not require the `READ LOCK` command for the entire syncing process, only for syncing *.frm* files (the same as with a regular backup).

- **Incremental State Transfer** (IST) is when only incremental changes are copied from one node to another.

  Even without locking your cluster in read-only state, SST may be intrusive and disrupt normal operation of your services. IST lets you avoid that. If a node goes down for a short period of time, it can fetch only those changes that happened while it was down. IST is implemeted using a caching mechanism on nodes. Each node contains a cache, ring-buffer (the size is configurable) of last N changes, and the node is able to transfer part of this cache. Obviously, IST can be done only if the amount of changes needed to transfer is less than N. If it exceeds N, then the joining node has to perform SST.

You can monitor the current state of a node using the following command:

```
SHOW STATUS LIKE 'wsrep_local_state_comment';
```

When a node is in `Synced (6)` state, it is part of the cluster and ready to handle traffic.

3.2 Multi-Master Replication

Multi-master replication means that you can write to any node and be sure that the write will be consistent for all nodes in the cluster. This is different from regular MySQL replication, where you have to apply writes to master to ensure that it will be synced.

With multi-master replication any write is either committed on all nodes or not committed at all. The following diagram shows how it works for two nodes, but the same logic is applied with any number of nodes in the cluster:

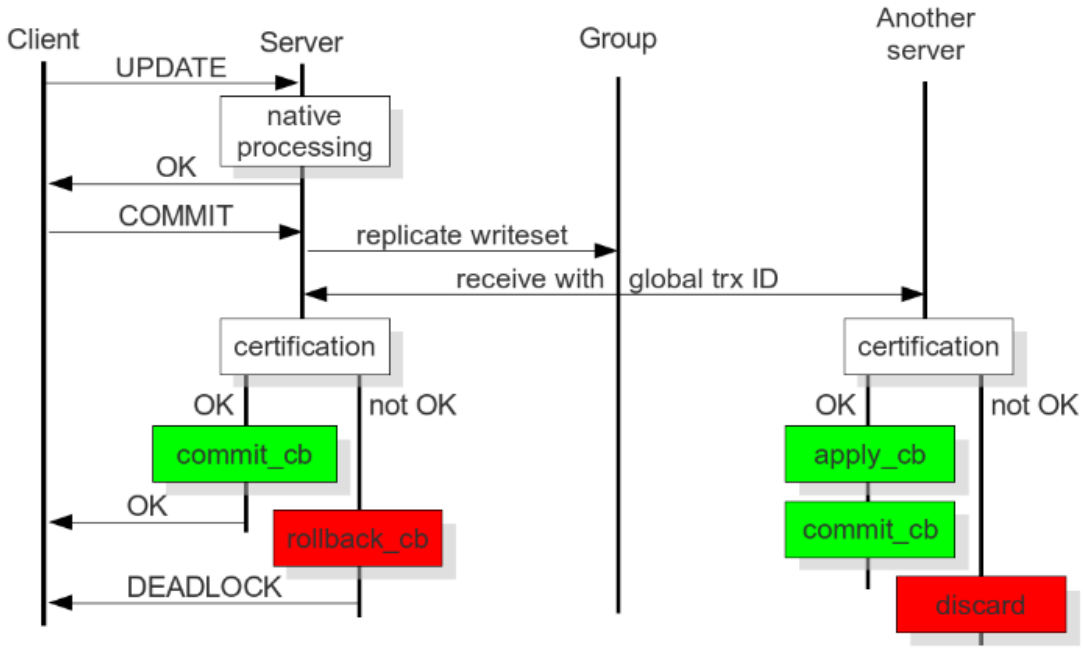


Figure 3.1: *Image source:* Galera documentation - HOW CERTIFICATION-BASED REPLICATION WORKS

All queries are executed locally on the node, and there is special handling only on `COMMIT`. When the `COMMIT` query is issued, the transaction has to pass certification on all nodes. If it does not pass, you will receive `ERROR` as the response for that query. After that, the transaction is applied on the local node.

Response time of `COMMIT` includes the following:

- Network round-trip time
- Certification time
- Local applying

Note: Applying the transaction on remote nodes does not affect the response time of `COMMIT`, because it happens in the background after the response on certification.

There are two important consequences of this architecture:

- Several appliers can be used in parallel. This enables truely parallel replication. A slave can have many parallel threads configured using the `wsrep_slave_threads` variable.

- There might be a small period of time when a slave is out of sync. This happens because the master may apply events faster than the slave. And if you do read from the slave, you may read the data that has not changed yet. You can see that from the diagram.

  However, this behavior can be changed by setting the `wsrep_causal_reads=ON` variable. In this case, the read on the slave will wait until the event is applied (this will obviously increase the response time of the read). The gap between the slave and the master is the reason why this replication is called *virtually synchronous replication*, and not *real synchronous replication*.

The described behavior of `COMMIT` also has another serious implication. If you run write transactions to two different nodes, the cluster will use an optimistic locking model. This means a transaction will not check on possible locking conflicts during the individual queries, but rather on the `COMMIT` stage, and you may get `ERROR` response on `COMMIT`.

This is mentioned because it is one of the incompatibilities with regular *InnoDB* that you might experience. With InnoDB, `DEADLOCK` and `LOCK TIMEOUT` errors usually happen in response to a particular query, but not on `COMMIT`. It is good practice to check the error codes after a `COMMIT` query, but there are still many applications that do not do that.

If you plan to use multi-master replication and run write transactions on several nodes, you may need to make sure you handle the responses on `COMMIT` queries.

3.3 PXC Strict Mode

PXC Strict Mode is designed to avoid the use of experimental and unsupported features in Percona XtraDB Cluster. It performs a number of validations at startup and during runtime.

Depending on the actual mode you select, upon encountering a failed validation, the server will either throw an error (halting startup or denying the operation), or log a warning and continue running as normal. The following modes are available:

- `DISABLED`: Do not perform strict mode validations and run as normal.
- `PERMISSIVE`: If a vaidation fails, log a warning and continue running as normal.
- `ENFORCING`: If a validation fails during startup, halt the server and throw an error. If a validation fails during runtime, deny the operation and throw an error.
- `MASTER`: The same as `ENFORCING` except that the validation of *explicit table locking* is not performed. This mode can be used with clusters in which write operations are isolated to a single node.

By default, PXC Strict Mode is set to `ENFORCING`, except if the node is acting as a standalone server or the node is bootstrapping, then PXC Strict Mode defaults to `DISABLED`.

It is recommended to keep PXC Strict Mode set to `ENFORCING`, because in this case whenever Percona XtraDB Cluster encounters an experimental feature or an unsupported operation, the server will deny it. This will force you to re-evaluate your Percona XtraDB Cluster configuration without risking the consistency of your data.

If you are planning to set PXC Strict Mode to anything else than `ENFORCING`, you should be aware of the limitations and effects that this may have on data integrity. For more information, see *Validations*.

To set the mode, use the `pxc_strict_mode` variable in the configuration file or the `--pxc-strict-mode` option during `mysqld` startup.

Note: It is better to start the server with the necessary mode (the default `ENFORCING` is highly recommended). However, you can dynamically change it during runtime. For example, to set PXC Strict Mode to `PERMISSIVE`, run the following command:

```
mysql> SET pxc_strict_mode=PERMISSIVE;
```

Note: To further ensure data consistency, it is important to have all nodes in the cluster running with the same configuration, including the value of `pxc_strict_mode` variable.

3.3.1 Validations

PXC Strict Mode validations are designed to ensure optimal operation for common cluster setups that do not require experimental features and do not rely on operations not supported by Percona XtraDB Cluster.

Warning: If an unsupported operation is performed on a node with `pxc_strict_mode` set to `DISABLED` or `PERMISSIVE`, it will not be validated on nodes where it is replicated to, even if the destination node has `pxc_strict_mode` set to `ENFORCING`.

This section describes the purpose and consequences of each validation.

Storage engine

Percona XtraDB Cluster currently supports replication only for tables that use a transactional storage engine (XtraDB or InnoDB). To ensure data consistency, the following statements should not be allowed for tables that use a non-transactional storage engine (MyISAM, MEMORY, CSV, etc.):

- Data manipulation statements that perform writing to table (for example, `INSERT`, `UPDATE`, `DELETE`, etc.)
- The following administrative statements: `CHECK`, `OPTIMIZE`, `REPAIR`, and `ANALYZE`
- `TRUNCATE TABLE` and `ALTER TABLE`

Depending on the selected mode, the following happens:

`DISABLED`

At startup, no validation is performed.

At runtime, all operations are permitted.

`PERMISSIVE`

At startup, no validation is perfromed.

At runtime, all operations are permitted, but a warning is logged when an undesirable operation is performed on an unsupported table.

`ENFORCING` or `MASTER`

At startup, no validation is performed.

At runtime, any undesirable operation performed on an unsupported table is denied and an error is logged.

Note: Unsupported tables can be converted to use a supported storage engine.

MyISAM replication

Percona XtraDB Cluster provides experimental support for replication of tables that use the MyISAM storage engine. Due to the non-transactional nature of MyISAM, it is not likely to ever be fully supported in Percona XtraDB Cluster.

MyISAM replication is controlled using the `wsrep_replicate_myisam` variable, which is set to `OFF` by default. Due to its unreliability, MyISAM replication should not be enabled if you want to ensure data consistency.

Depending on the selected mode, the following happens:

`DISABLED`

At startup, no validation is performed.

At runtime, you can set `wsrep_replicate_myisam` to any value.

`PERMISSIVE`

At startup, if `wsrep_replicate_myisam` is set to `ON`, a warning is logged and startup continues.

At runtime, it is permitted to change `wsrep_replicate_myisam` to any value, but if you set it to `ON`, a warning is logged.

`ENFORCING` or `MASTER`

At startup, if `wsrep_replicate_myisam` is set to `ON`, an error is logged and startup is aborted.

At runtime, any attempt to change `wsrep_replicate_myisam` to `ON` fails and an error is logged.

Note: The `wsrep_replicate_myisam` variable controls *replication* for MyISAM tables, and this validation only checks whether it is allowed. Undesirable operations for MyISAM tables are restricted using the *Storage engine* validation.

Binary log format

Percona XtraDB Cluster supports only the default row-based binary logging format. Setting the `binlog_format` [1] variable to anything but `ROW` at startup is not allowed, because this changes the global scope, which must be set to ROW. Validation is performed only at runtime and against session scope.

Depending on the selected mode, the following happens:

`DISABLED`

At runtime, you can set `binlog_format` to any value.

`PERMISSIVE`

At runtime, it is permitted to change `binlog_format` to any value, but if you set it to anything other than `ROW`, a warning is logged.

`ENFORCING` or `MASTER`

At runtime, any attempt to change `binlog_format` to anything other than `ROW` fails and an error is logged.

[1] http://dev.mysql.com/doc/refman/5.7/en/replication-options-binary-log.html#sysvar_binlog_format

Tables without primary keys

Percona XtraDB Cluster cannot properly propagate certain write operations to tables that do not have primary keys defined. Undesirable operations include data manipulation statements that perform writing to table (especially `DELETE`).

Depending on the selected mode, the following happens:

`DISABLED`

At startup, no validation is performed.

At runtime, all operations are permitted.

`PERMISSIVE`

At startup, no validation is perfromed.

At runtime, all operations are permitted, but a warning is logged when an undesirable operation is performed on a table without an explicit primary key defined.

`ENFORCING` or `MASTER`

At startup, no validation is performed.

At runtime, any undesirable operation performed on a table without an explicit primary key is denied and an error is logged.

Log output

Percona XtraDB Cluster does not support tables in the MySQL database as the destination for log output. By default, log entries are written to file. This validation checks the value of the `log_output` [2] variable.

Depending on the selected mode, the following happens:

`DISABLED`

At startup, no validation is performed.

At runtime, you can set `log_output` to any value.

`PERMISSIVE`

At startup, if `log_output` is set only to `TABLE`, a warning is logged and startup continues.

At runtime, it is permitted to change `log_output` to any value, but if you set it only to `TABLE`, a warning is logged.

`ENFORCING` or `MASTER`

At startup, if `log_output` is set only to `TABLE`, an error is logged and startup is aborted.

At runtime, any attempt to change `log_output` only to `TABLE` fails and an error is logged.

Explicit table locking

Percona XtraDB Cluster has only experimental support for explicit table locking operations, The following undesirable operations lead to explicit table locking and are covered by this validation:

- `LOCK TABLES`
- `GET_LOCK()` and `RELEASE_LOCK()`

[2] http://dev.mysql.com/doc/refman/5.7/en/server-system-variables.html#sysvar_log_output

- `FLUSH TABLES <tables> WITH READ LOCK`
- Setting the `SERIALIZABLE` transaction level

Depending on the selected mode, the following happens:

`DISABLED` or `MASTER`

At startup, no validation is performed.

At runtime, all operations are permitted.

`PERMISSIVE`

At startup, no validation is performed.

At runtime, all operations are permitted, but a warning is logged when an undesirable operation is performed.

`ENFORCING`

At startup, no validation is performed.

At runtime, any undesirable operation is denied and an error is logged.

Auto-increment lock mode

The lock mode for generating auto-increment values must be *interleaved* to ensure that each node generates a unique (but non-sequential) identifier.

This validation checks the value of the `innodb_autoinc_lock_mode` [3] variable. By default, the variable is set to `1` (*consecutive* lock mode), but it should be set to `2` (*interleaved* lock mode).

Depending on the strict mode selected, the following happens:

`DISABLED`

At startup, no validation is performed.

`PERMISSIVE`

At startup, if `innodb_autoinc_lock_mode` is not set to `2`, a warning is logged and startup continues.

`ENFORCING` or `MASTER`

At startup, if `innodb_autoinc_lock_mode` is not set to `2`, an error is logged and startup is aborted.

Note: This validation is not performed during runtime, because the `innodb_autoinc_lock_mode` variable cannot be set dynamically.

Combining schema and data changes in a single statement

Percona XtraDB Cluster does not support `CREATE TABLE ... AS SELECT` (CTAS) statements, because they combine both schema and data changes.

Depending on the strict mode selected, the following happens:

`DISABLED`

At startup, no validation is performed.

At runtime, all operations are permitted.

[3] http://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html#sysvar_innodb_autoinc_lock_mode

`PERMISSIVE`

At startup, no validation is perfromed.

At runtime, all operations are permitted, but a warning is logged when a CTAS operation is performed.

`ENFORCING`

At startup, no validation is performed.

At runtime, any CTAS operation is denied and an error is logged.

References

CHAPTER

FOUR

USER'S MANUAL

4.1 Bootstrapping the cluster

Bootstrapping refers to setting up the initial node in the cluster. Whether you are creating a new cluster from scratch or restoring after a cluster-wide crash, you need to define which node contains the most relevant data. Other nodes will synchronize to this initial node using *SST*.

The *MySQL* configuration file (`my.cnf`) should contain at least the following necessary configuration options:

```
[mysqld]
# Path to Galera library
wsrep_provider=/usr/lib64/libgalera_smm.so

# Cluster connection URL
wsrep_cluster_address=gcomm://

# Binlog format (Galera requires it to be ROW)
binlog_format=ROW

# Default storage engine (MyISAM is considered experimental)
default_storage_engine=InnoDB

# InnoDB autoincrement locks mode (Galera requires it to be 2)
innodb_autoinc_lock_mode=2
```

Bootstrapping the cluster involes the following steps:

1. On the initial node, set the `wsrep_cluster_address` variable to `gcomm://`.

   This tells the node it can bootstrap without any cluster to connect to. Setting that and starting up the first node should result in a cluster with a `wsrep_cluster_conf_id` of 1.

2. After starting a single-node cluster, change the `wsrep_cluster_address` variable to list all the other nodes in the cluster. For example:

   ```
   wsrep_cluster_address=gcomm://192.168.70.2,192.168.70.3,192.168.70.4
   ```

   Cluster membership is not defined by this setting. It is defined by the nodes that join the cluster with the proper cluster name configured using the `wsrep_cluster_name` variable (defaults to `my_wsrep_cluster`). Hence, the `wsrep_cluster_address` variable does not need to be identical on all nodes. However, it is good practice, because on restart the node will try all other nodes in that list and look for any that are currently running in the cluster.

3. Once the first node is configured, start other nodes, one at a time.

   When bootstrapping, other nodes will most likely be using *SST*, so you should avoid joining multiple nodes at once.

If the cluster has been bootstrapped before, to avoid editing the `my.cnf` twice to change the `wsrep_cluster_address` to `gcomm://` and then to other node addresses, the initial node can be started with:

```
/etc/init.d/mysql bootstrap-pxc
```

Note: On CentOS/RHEL 7, the following bootstrap command should be used:

```
systemctl start mysql@bootstrap.service
```

The above commands will ensure that values in `my.cnf` remain unchanged. The next time the node is restarted, it won't require updating the configuration file. This can be useful when the cluster has been previously set up and for some reason all nodes went down and the cluster needs to be restored.

4.1.1 Other Reading

- How to start a Percona XtraDB Cluster

4.2 State Snapshot Transfer

State Snapshot Transfer (SST) is a full data copy from one node (donor) to the joining node (joiner). It's used when a new node joins the cluster. In order to be synchronized with the cluster, the new node has to receive data from a node that is already part of the cluster.

There are three methods of SST available in Percona XtraDB Cluster:

- **mysqldump**
- **rsync**
- **xtrabackup**

The downside of `mysqldump` and `rsync` is that the donor node becomes **READ-ONLY** while data is being copied. Xtrabackup SST, on the other hand, uses backup locks, which means the Galera provider is not paused at all as with FTWRL (Flush Tables with Read Lock) earlier. The SST method can be configured using the `wsrep_sst_method` variable.

Note: If the `gcs.sync_donor` variable is set to `Yes` (default is `No`), the whole cluster will get blocked if the donor is blocked by SST.

4.2.1 Choosing the SST Donor

If there are no nodes available that can safely perform incremental state transfer (*IST*), the cluster defaults to *SST*.

If there are nodes available that can perform *IST*, the cluster prefers a local node over remote nodes to serve as the donor.

If there are no local nodes available that can perform *IST*, the cluster chooses a remote node to serve as the donor.

If there are several local and remote nodes that can perform *IST*, the cluster chooses the node with the highest `seqno` to serve as the donor.

4.2.2 Using Percona Xtrabackup

The default SST method is `xtrabackup-v2` which uses *Percona XtraBackup*. This is the least blocking method that leverages backup locks. XtraBackup is run locally on the donor node, so it's important that the correct user credentials are set up on the donor node. In order for Percona XtraDB Cluster to perform SST using XtraBackup, credentials for connecting to the donor node need to be set up in the `wsrep_sst_auth` variable. Besides the credentials, the *datadir* needs to be specified in the server configuration file `my.cnf`, otherwise the transfer process will fail.

For more information about the required credentials, see the XtraBackup manual.

To test if the credentials will work, run **innobackupex** on the donor node with the username and password specified in the `wsrep_sst_auth` variable. For example, if the value of `wsrep_sst_auth` is `root:Passw0rd`, the **innobackupex** command should look like this:

```
innobackupex --user=root --password=Passw0rd /tmp/
```

Detailed information on this method is provided in *Percona XtraBackup SST Configuration* documentation.

4.2.3 Using mysqldump

This method uses the standard **mysqldump** utility to dump all the databases from the donor node and import them to the joining node. For this method to work, the `wsrep_sst_auth` variable needs to be set up with the root credentials. This method is the slowest and it performs a global lock during *SST*, which blocks writes to the donor node.

The script used for this method is `/usr/bin/wsrep_sst_mysqldump` and it is included in the Percona XtraDB Cluster binary packages.

4.2.4 Using `rsync`

This method uses **rsync** to copy files from donor to the joining node. In some cases, this can be faster than using XtraBackup, but it requires a global data lock, which will block writes to the donor node. This method doesn't require root credentials to be set up in the `wsrep_sst_auth` variable.

The script used for this method is `/usr/bin/wsrep_sst_rsync` and it is included in the Percona XtraDB Cluster binary packages.

4.2.5 Other Reading

- SST Methods for MySQL
- *Xtrabackup SST configuration*

4.3 Percona XtraBackup SST Configuration

Percona XtraBackup SST works in two stages:

- First it identifies the type of data transfer based on the presence of `xtrabackup_ist` file on the joiner node.
- Then it starts data transfer:
  - In case of *SST*, it empties the data directory except for some files (`galera.cache`, `sst_in_progress`, `grastate.dat`) and then proceeds with SST
  - In case of *IST*, it proceeds as before.

Note: As of Percona XtraDB Cluster 5.7, `xtrabackup-v2` is the only XtraBackup SST method.

4.3.1 SST Options

The following options specific to *SST* can be used in `my.cnf` under `[sst]`.

Note:

- Non-integer options which have no default value are disabled if not set.
- `:Match:  Yes` implies that option should match on donor and joiner nodes.
- SST script reads `my.cnf` when it runs on either donor or joiner node, not during `mysqld` startup.
- SST options must be specified in the main `my.cnf` file.

option `streamfmt`

Values xbstream, tar

Default xbstream

Match Yes

Used to specify the Percona XtraBackup streaming format. The recommended value is `streamfmt=xbstream`. Certain features are not available with `tar`, for instance: encryption, compression, parallel streaming, streaming incremental backups. For more information about the `xbstream` format, see The xbstream Binary.

option `transferfmt`

Values socat, nc

Default socat

Match Yes

Used to specify the data transfer format. The recommended value is `transferfmt=socat`, because it allows for socket options, such as transfer buffer sizes. For more information, see socat(1).

option `tca`

Example tca=~/etc/ssl/certs/mycert.crt

Used to specify the full path to the certificate authority (CA) file for `socat` encryption based on OpenSSL.

option `tcert`

Example tcert=~/etc/ssl/certs/mycert.pem

Used to specify the full path to the certificate file in PEM format for `socat` encryption based on OpenSSL.

Note: For more information about `tca` and `tcert`, see http://www.dest-unreach.org/socat/doc/socat-openssltunnel.html. The `tca` is essentially the self-signed certificate in that example, and `tcert` is the PEM file generated after concatenation of the key and the certificate generated earlier. The names of options were chosen to be compatible with `socat` parameter names as well as with MySQL's SSL authentication. For testing you can also download certificates from launchpad.

Note: Irrespective of what is shown in the example, you can use the same .crt and .pem files on all nodes and it will work, since there is no server-client paradigm here, but rather a cluster with homogeneous nodes.

option `encrypt`

Values 0, 1, 2, 3

Default 0

Match Yes

Used to enable and specify SST encryption mode:

- Set `encrypt=0` to disable SST encryption. This is the default value.
- Set `encrypt=1` to perform symmetric SST encryption based on XtraBackup.
- Set `encrypt=2` to perform SST encryption based on OpenSSL with `socat`. Ensure that `socat` is built with OpenSSL: `socat -V | grep OPENSSL`. This is recommended if your nodes are over WAN and security constraints are higher.
- Set `encrypt=3` to perform SST encryption based on SSL for just the key and certificate files as implemented in Galera.

  The latter has been implemented in `5.5.34-23.7.6` for compatibility with Galera. It does not provide certificate validation. In order to work correctly, paths to the key and certificate files need to be specified as well, for example:

```
[sst]
encrypt=3
tkey=/etc/mysql/key.pem
tcert=/etc/mysql/cert.pem
```

Note: The `encrypt=3` option can only be used when `wsrep_sst_method` is set to `xtrabackup-v2` (which is the default).

---

option `encrypt-algo`

Values AES128, AES192, AES256

Used to specify the SST encryption algorithm. It uses the same values as the `--encryption` option for XtraBackup (see this document). The `encrypt-algo` option is considered only if `encrypt` is set to `1`.

option `sockopt`

Used to specify key/value pairs of socket options, separated by commas. Must begin with a comma. You can use the `tcpwrap` option to blacklist or whitelist clients. For more information about socket options, see socat (1).

Note: You can also enable SSL-based compression with `sockopt`. This can be used in place of the XtraBackup `compress` option.

---

option `progress`

Values 1, path/to/file

Used to specify where to write SST progress. If set to `1`, it writes to MySQL `stderr`. Alternatively, you can specify the full path to a file. If this is a FIFO, it needs to exist and be open on reader end before itself, otherwise `wsrep_sst_xtrabackup` will block indefinitely.

Note: Value of 0 is not valid.

---

option `rebuild`

Values 0, 1

Default 0

Used to enable rebuilding of index on joiner node. This is independent of compaction, though compaction enables it. Rebuild of indexes may be used as an optimization.

Note: #1192834 affects this option.

---

option `time`

Values 0, 1

Default 0

Enabling this option instruments key stages of backup and restore in SST.

option `rlimit`

Example rlimit=128k

Used to set a a ratelimit in bytes. Add a suffix (k, m, g, t) to specify units. For example, `128k` is 128 kilobytes. For more information, see pv(1).

Note: Rate is limited on donor node. The rationale behind this is to not allow SST to saturate the donor's regular cluster operations or to limit the rate for other purposes.

---

option `incremental`

Values 0, 1

Default 0

Used to sepersede IST on joiner node. Requires manual setup and is not supported currently.

option `use_extra`

Values 0, 1

Default 0

Used to force SST to use the thread pool's extra_port. Make sure that thread pool is enabled and the `extra_port` option is set in `my.cnf` before you enable this option.

option `cpat`

Used to define the files that need to be deleted in the *datadir* before running SST, so that the state of the other node can be restored cleanly. For example:

```
[sst]
cpat='.*galera\.cache$\|.*sst_in_progress$\|.*grastate\.dat$\|.*\.err$\|.*\.log$\|.*RPM_UPGRADE_MARKI
```

Note: This option can only be used when `wsrep_sst_method` is set to `xtrabackup-v2` (which is the default value).

---

option `compressor`

Default not set (disabled)

Example compressor='gzip'

option `decompressor`

Default not set (disabled)

Example decompressor='gzip -dc'

Two previous options enable stream-based compression/decompression. When these options are set, compression/decompression is performed on stream, in contrast to performing decompression after streaming to disk, involving additional I/O. This saves a lot of I/O (up to twice less I/O on joiner node).

You can use any compression utility which works on stream: `gzip`, `pigz` (which is recommended because it is multi-threaded), etc. Compressor has to be set on donor node and decompressor on joiner node (although you can set them vice-versa for configuration homogeneity, it won't affect that particular SST). To use XtraBackup based compression as before, set `compress` under `[xtrabackup]`. Having both enabled won't cause any failure (although you will be wasting CPU cycles).

option `inno-backup-opts`

option `inno-apply-opts`

option `inno-move-opts`

Default Empty

Type Quoted String

This group of options can be used to pass innobackupex options for backup, apply, and move stages.

Note: Although these options are related to XtraBackup SST, they cannot be specified in `my.cnf`, because they are for passing innobackupex options.

---

option `sst-initial-timeout`

Default 100

Unit seconds

This option is used to configure initial timeout (in seconds) to receive the first packet via SST. This has been implemented, so that if the donor node fails somewhere in the process, the joiner node will not hang up and wait forever.

By default, the joiner node will not wait for more than 100 seconds to get a donor node. The default should be sufficient, however, it is configurable, so you can set it appropriately for your cluster. To disable initial SST timeout, set `sst-initial-timeout=0`.

Note: If you are using `wsrep_sst_donor`, and you want the joiner node to strictly wait for donors listed in the variable and not fall back (that is, without a terminating comma at the end), **and** there is a possibility of **all** nodes in that variable to be unavailable, disable initial SST timeout or set it to a higher value (maximum threshold that you want the joiner node to wait). You can also disable this option (or set it to a higher value) if you believe all other nodes in the cluster can potentially become unavailable at any point in time (mostly in small clusters) or there is a high network latency or network disturbance (which can cause donor selection to take longer than 100 seconds).

4.3.2 XtraBackup SST Dependencies

The following are optional dependencies of Percona XtraDB Cluster introduced by `wsrep_sst_xtrabackup-v2` (except for obvious and direct dependencies):

- `qpress` for decompression. It is an optional dependency of *Percona XtraBackup* 2.1.4 and it is available in our software repositories.
- `my_print_defaults` to extract values from `my.cnf`. Provided by the server package.
- `openbsd-netcat` or `socat` for transfer. `socat` is a direct dependency of Percona XtraDB Cluster and it is the default.
- `xbstream` or `tar` for streaming. `tar` is default.

- `pv` is required for `progress` and `rlimit`.
- `mkfifo` is required for `progress`. Provided by `coreutils`.
- `mktemp` is required for `incremental`. Provided by `coreutils`.

4.3.3 XtraBackup-based Encryption

This is enabled when `encrypt` is set to `1` under `[sst]` in `my.cnf`. However, due to bug #1190335, it will also be enabled when you specify any of the following options under `[xtrabackup]` in `my.cnf`:

- `encrypt`
- `encrypt-key`
- `encrypt-key-file`

There is no way to disable encryption from innobackupex if any of the above are in `my.cnf` under `[xtrabackup]`. For that reason, comsider the following scenarios:

1. If you want to use XtraBackup-based encryption for SST but not otherwise, use `encrypt=1` under `[sst]` and provide the above XtraBackup encryption options under `[sst]`. Details of those options can be found here.
2. If you want to use XtraBackup-based encryption always, use `encrypt=1` under `[sst]` and have the above XtraBackup encryption options either under `[sst]` or `[xtrabackup]`.
3. If you don't want to use XtraBackup-based encryption for SST, but want it otherwise, use `encrypt=0` or `encrypt=2` and do **NOT** provide any XtraBackup encryption options under `[xtrabackup]`. You can still have them under `[sst]` though. You will need to provide those options on innobackupex command line then.
4. If you don't want to use XtraBackup-based encryption at all (or only the OpenSSL-based for SST with `encrypt=2`), don't provide any XtraBackup encryption options in `my.cnf`.

Note: The `encrypt` option under `[sst]` is different from the one under `[xtrabackup]`. The former is for disabling/changing encryption mode, while the latter is to provide an encryption algorithm. To disambiguate, if you need to provide the latter under `[sst]` (for example, in cases 1 and 2 above), it should be specified as `encrypt-algo`.

Warning: An implication of the above is that if you specify any of the XtraBackup encryption options, and `encrypt=0` under `[sst]`, it will still be encrypted and SST will fail. Look at case 3 above for resolution.

4.4 Restarting the cluster nodes

To restart a cluster node, shut down MySQL and restarting it. The node should leave the cluster (and the total vote count for *quorum* should decrement).

When it rejoins, the node should synchronize using *IST*. If the set of changes needed for IST are not found in the `gcache` file on any other node in the entire cluster, then *SST* will be performed instead. Therefore, restarting cluster nodes for rolling configuration changes or software upgrades is rather simple from the cluster's perspective.

Note: If you restart a node with an invalid configuration change that prevents MySQL from loading, Galera will drop the node's state and force an SST for that node.

4.5 Cluster Failover

Cluster membership is determined simply by which nodes are connected to the rest of the cluster; there is no configuration setting explicitly defining the list of all possible cluster nodes. Therefore, every time a node joins the cluster, the total size of the cluster is increased and when a node leaves (gracefully) the size is decreased.

The size of the cluster is used to determine the required votes to achieve *quorum*. A quorum vote is done when a node or nodes are suspected to no longer be part of the cluster (they do not respond). This no response timeout is the `evs.suspect_timeout` setting in the `wsrep_provider_options` (default 5 sec), and when a node goes down ungracefully, write operations will be blocked on the cluster for slightly longer than that timeout.

Once a node (or nodes) is determined to be disconnected, then the remaining nodes cast a quorum vote, and if the majority of nodes from before the disconnect are still still connected, then that partition remains up. In the case of a network partition, some nodes will be alive and active on each side of the network disconnect. In this case, only the quorum will continue. The partition(s) without quorum will change to non-primary state.

As a consequence, it's not possible to have safe automatic failover in a 2 node cluster, because failure of one node will cause the remaining node to become non-primary. Moreover, any cluster with an even number of nodes (say two nodes in two different switches) have some possibility of a *split brain* situation, when neither partition is able to retain quorum if connection between them is lost, and so they both become non-primary.

Therefore, for automatic failover, the *rule of 3s* is recommended. It applies at various levels of your infrastructure, depending on how far the cluster is spread out to avoid single points of failure. For example:

- A cluster on a single switch should have 3 nodes
- A cluster spanning switches should be spread evenly across at least 3 switches
- A cluster spanning networks should span at least 3 networks
- A cluster spanning data centers should span at least 3 data centers

These rules will prevent split brain situations and ensure automatic failover works correctly.

4.5.1 Using an arbitrator

If it is too expensive to add a third node, switch, network, or datacenter, you should use an arbitrator. An arbitrator is a voting member of the cluster that can receive and relay replication, but it does not persist any data, and runs its own daemon instead of `mysqld`. Placing even a single arbitrator in a 3rd location can add split brain protection to a cluster that is spread across only two nodes/locations.

4.5.2 Recovering a Non-Primary cluster

It is important to note that the *rule of 3s* applies only to automatic failover. In the event of a 2-node cluster (or in the event of some other outage that leaves a minority of nodes active), the failure of one node will cause the other to become non-primary and refuse operations. However, you can recover the node from non-primary state using the following command:

```
SET GLOBAL wsrep_provider_options='pc.bootstrap=true';
```

This will tell the node (and all nodes still connected to its partition) that it can become a primary cluster. However, this is only safe to do when you are sure there is no other partition operating in primary as well, or else Percona XtraDB Cluster will allow those two partitions to diverge (and you will end up with two databases that are impossible to re-merge automatically).

For example, assume there are two data centers, where one is primary and one is for disaster recovery, with an even number of nodes in each. When an extra arbitrator node is run only in the primary data center, the following high availability features will be available:

- Auto-failover of any single node or nodes within the primary or secondary data center
- Failure of the secondary data center would not cause the primary to go down (because of the arbitrator)
- Failure of the primary data center would leave the secondary in a non-primary state.
- If a disaster-recovery failover has been executed, you can tell the secondary data center to bootstrap itself with a single command, but disaster-recovery failover remains in your control.

4.5.3 Other Reading

- PXC - Failure Scenarios with only 2 nodes

4.6 Monitoring the cluster

Each node can have a different view of the cluster. There is no centralized node to monitor. To track down the source of issues, you have to monitor each node independently.

Values of many variables depend on the node from which you are querying. For example, replication sent from a node and writes received by all other nodes.

Having data from all nodes can help you understand where flow messages are coming from, which node sends excessively large transactions, and so on.

4.6.1 Manual Monitoring

Manual cluster monitoring can be performed using myq_tools.

4.6.2 Alerting

Besides standard MySQL alerting, you should use at least the following triggers specific to Percona XtraDB Cluster:

- Cluster state of each node
  - `wsrep_cluster_status` != Primary
- Node state
  - `wsrep_connected` != `ON`
  - `wsrep_ready` != `ON`

For additional alerting, consider the following:

- Excessive replication conflicts can be identtified using the `wsrep_local_cert_failures` and `wsrep_local_bf_aborts` variables
- Excessive flow control messages can be identified using the `wsrep_flow_control_sent` and `wsrep_flow_control_recv` variables
- Large replication queues can be identified using the `wsrep_local_recv_queue`.

4.6.3 Metrics

Cluster metrics collection for long-term graphing should be done at least for the following:

- Queue sizes: `wsrep_local_recv_queue` and `wsrep_local_send_queue`
- Flow control: `wsrep_flow_control_sent` and `wsrep_flow_control_recv`
- Number of transactions for a node: `wsrep_replicated` and `wsrep_received`
- Number of transactions in bytes: `wsrep_replicated_bytes` and `wsrep_received_bytes`
- Replication conflicts: `wsrep_local_cert_failures` and `wsrep_local_bf_aborts`

4.6.4 Other Reading

- Realtime stats to pay attention to in PXC and Galera

4.7 Certification in Percona XtraDB Cluster

Percona XtraDB Cluster replicates actions executed on one node to all other nodes in the cluster, and makes it fast enough to appear as if it is synchronous (*virtually synchronous*).

The following types of actions exist:

- DDL actions are executed using *Total Order Isolation* (TOI). We can ignore *Rolling Schema Upgrades* (ROI).
- DML actions are executed using normal Galera replication protocol.

Note: This manual page assumes the reader is aware of TOI and MySQL replication protocol.

DML (`INSERT`, `UPDATE`, and `DELETE`) operations effectively change the state of the database, and all such operations are recorded in *XtraDB* by registering a unique object identifier (key) for each change (an update or a new addition).

- A transaction can change an arbitrary number of different data objects. Each such object change is recorded in *XtraDB* using an `append_key` operation. An `append_key` operation registers the key of the data object that has undergone change by the transaction. The key for rows can be represented in three parts as `db_name`, `table_name`, and `pk_columns_for_table` (if `pk` is absent, a hash of the complete row is calculated).

  This ensures that there is quick and short meta information about the rows that this transaction has touched or modified. This information is passed on as part of the write-set for certification to all the nodes in the cluster while the transaction is in the commit phase.

- For a transaction to commit, it has to pass XtraDB/Galera certification, ensuring that transactions don't conflict with any other changes posted on the cluster group/channel. Certification will add the keys modified by a given transaction to its own central certification vector (CCV), represented by `cert_index_ng`. If the said key is already part of the vector, then conflict resolution checks are triggered.

- Conflict resolution traces the reference transaction (that last modified this item in the cluster group). If this reference transaction is from some other node, that suggests the same data was modified by the other node, and changes of that node have been certified by the local node that is executing the check. In such cases, the transaction that arrived later fails to certify.

Changes made to database objects are bin-logged. This is similar to how *MySQL* does it for replication with its Master-Slave ecosystem, except that a packet of changes from a given transaction is created and named as a write-set.

Once the client/user issues a `COMMIT`, Percona XtraDB Cluster will run a commit hook. Commit hooks ensure the following:

- Flush the binary logs.
- Check if the transaction needs replication (not needed for read-only transactions like `SELECT`).
- If a transaction needs replication, then it invokes a pre-commit hook in the Galera ecosystem. During this pre-commit hook, a write-set is written in the group channel by a *replicate* operation. All nodes (including the one that executed the transaction) subscribe to this group-channel and read the write-set.
- `gcs_recv_thread` is the first to receive the packet, which is then processed through different action handlers.
- Each packet read from the group-channel is assigned an `id`, which is a locally maintained counter by each node in sync with the group. When any new node joins the group/cluster, a seed-id for it is initialized to the current active id from group/cluster.

  There is an inherent assumption/protocol enforcement that all nodes read the packet from a channel in the same order, and that way even though each packet doesn't carry `id` information, it is inherently established using the locally maintained `id` value.

4.7.1 Common Situation

The following example shows what happens in a common situation. `act_id` is incremented and assigned only for totally ordered actions, and only in primary state (skip messages while in state exchange).

```
rcvd->id = ++group->act_id_;
```

Note: This is an amazing way to solve the problem of the id coordination in multi-master systems. Otherwise a node will have to first get an id from central system or through a separate agreed protocol, and then use it for the packet, thereby doubling the round-trip time.

4.7.2 Conflicts

The following happens if two nodes get ready with their packet at same time:

- Both nodes will be allowed to put the packet on the channel. That means the channel will see packets from different nodes queued one behind another.
- The following example shows what happens if two nodes modify same set of rows. Nodes are in sync until this point:

  ```
  create -> insert (1,2,3,4)
  ```

  - Node 1: `update i = i + 10;`
  - Node 2: `update i = i + 100;`

  Let's associate transaction ID (`trx-id`) for an update transaction that is executed on Node 1 and Node 2 in parallel. Although the real algorithm is more involved (with `uuid` + `seqno`), it is conceptually the same, so we are using `trx_id`.

  - Node 1: `update action:  trx-id=n1x`
  - Node 2: `update action:  trx-id=n2x`

  Both node packets are added to the channel, but the transactions are conflicting. The protocol says: FIRST WRITE WINS.

So in this case, whoever is first to write to the channel will get certified. Let's say Node 2 is first to write the packet, and then Node 1 makes changes immediately after it.

Note: Each node subscribes to all packages, including its own package.

- Node 2 will see its own packet and will process it. Then it will see the packet from Node 1, try to certify it, and fail.
- Node 1 will see the packet from Node 2 and will process it.

  Note: InnoDB allows isolation, so Node 1 can process packets from Node 2 independent of Node 1 transaction changes

  Then Node 1 will see its own packet, try to certify it, and fail.

  Note: Even though the packet originated from Node 1, it will undergo certification to catch cases like these.

4.7.3 Resolving Certification Conflicts

The certification protocol can be described using the previous example. The central certification vector (CCV) is updated to reflect reference transaction.

- Node 2 sees its own packet for certification, adds it to its local CCV and performs certification checks. Once these checks pass, it updates the reference transaction by setting it to `n2x`.

  Node 2 then gets the packet from Node 1 for certification. The packet key is already present in CCV, with the reference transaction set it to `n2x`, whereas write-set proposes setting it to `n1x`. This causes a conflict, which in turn causes the transaction from Node 1 to fail the certification test.

- Node 1 sees the packet from Node 2 for certification, which is then processed, the local CCV is updated, and the reference transaction is set to `n2x`.

  Using the same case as explained above, Node 1 certification also rejects the packet from Node 1.

This suggests that the node doesn't need to wait for certification to complete, but just needs to ensure that the packet is written to the channel. The applier transaction will always win and the local conflicting transaction will be rolled back.

The following example shows what happens if one of the nodes has local changes that are not synced with the group:

```
create (id primary key) -> insert (1), (2), (3), (4);
node-1: wsrep_on=0; insert (5); wsrep_on=1
node-2: insert(5).
```

The `insert(5)` statement will generate a write-set that will then be replicated to Node 1. Node 1 will try to apply it but will fail with `duplicate-key-error`, because 5 already exist.

XtraDB will flag this as an error, which would eventually cause Node 1 to shutdown.

4.7.4 Incrementing GTID

GTID is incremented only when the transaction passes certification, and is ready for commit. That way errant packets don't cause GTID to increment.

Also, group packet `id` is not confused with GTID. Without errant packets, it may seem that these two counters are the same, but they are not related.

4.8 Percona XtraDB Cluster threading model

Percona XtraDB Cluster creates a set of threads to service its operations, which are not related to existing *MySQL* threads. There are three main groups of threads:

Contents

4.8.1 Applier threads

Applier threads apply write-sets that the node receives from other nodes. Write messages are directed through `gcv_recv_thread`.

The number of applier threads is controlled using the `wsrep_slave_threads` variable. The default value is `1`, which means at least one wsrep applier thread exists to process the request.

Applier threads wait for an event, and once it gets the event, it applies it using normal slave apply routine path, and relays the log info apply path with wsrep-customization. These threads are similar to slave worker threads (but not exactly the same).

Coordination is achieved using *Apply and Commit Monitor*. A transaction passes through two important states: `APPLY` and `COMMIT`. Every transaction registers itself with an apply monitor, where its apply order is defined. So all transactions with apply order sequence number (`seqno`) of less than this transaction's sequence number, are applied before applying this transaction. The same is done for commit as well (`last_left >= trx_.depends_seqno()`).

4.8.2 Rollback thread

There is only one rollback thread to perform rollbacks in case of conflicts.

Transactions executed in parallel can conflict and may need to roll back. Applier transactions always take priority over local transactions. This is natural, as applier transactions have been accepted by the cluster, and some of the nodes may have already applied them. Local conflicting transactions still have a window to rollback.

All the transactions that need to be rolled back are added to the rollback queue, and the rollback thread is notified. The rollback thread then iterates over the queue and performs rollback operations.

If a transaction is active on a node, and a node receives a transaction write-set from the cluster group that conflicts with the local active transaction, then such local transactions are always treated as a victim transaction to roll back.

Transactions can be in a commit state or an execution stage when the conflict arises. Local transactions in the execution stage are forcibly killed so that the waiting applier transaction is allowed to proceed. Local transactions in the commit stage fail with a certification error.

4.8.3 Other threads

Service thread

This thread is created during boot-up and used to perform auxiliary services. It has two main functions:

- It releases the GCache buffer after the cached write-set is purged up to the said level.

- It notifies the cluster group that the respective node has committed a transaction up to this level. Each node maintains some basic status info about other nodes in the cluster. On receiving the message, the information is updated in this local metadata.

Receiving thread

The `gcs_recv_thread` thread is the first one to see all the messages received in a group.

It will try to assign actions against each message it receives. It adds these messages to a central FIFO queue, which are then processed by the Applier threads. Messages can include different operations like state change, configuration update, flow-control, and so on.

One important action is processing a write-set, which actually is applying transactions to database objects.

Gcomm connection thread

The gcomm connection thread `GCommConn::run_fn` is used to co-ordinate the low-level group communication activity. Think of it as a black box meant for communication.

Action-based threads

Besides the above, some threads are created on a needed basis. SST creates threads for donor and joiner (which eventually forks out a child process to host the needed SST script), IST creates receiver and async sender threads, PageStore creates a background thread for removing the files that were created.

If the checksum is enabled and the replicated write-set is big enough, the checksum is done as part of a separate thread.

4.9 Understanding GCache and Record-Set Cache

In Percona XtraDB Cluster, there is a concept of GCache and Record-Set cache (which can also be called transaction write-set cache). The use of these two caches is often confusing if you are running long transactions, because both of them result in the creation of disk-level files. This manual describes what their main differences are.

4.9.1 Record-Set Cache

When you run a long-running transaction on any particular node, it will try to append a key for each row that it tries to modify (the key is a unique identifier for the row `{db,table,pk.columns}`). This information is cached in out-write-set, which is then sent to the group for certification.

Keys are cached in HeapStore (which has `page-size=65K` and `total-size=4MB`). If the transaction data-size outgrows this limit, then the storage is switched from Heap to Page (which has `page-size=64MB` and `total-limit=free-space-on-disk`).

All these limits are non-configurable, but having a memory-page size greater than 4MB per transaction can cause things to stall due to memory pressure, so this limit is reasonable. This is another limitation to address when Galera supports large transaction.

The same long-running transaction will also generate binlog data that also appends to out-write-set on commit (`HeapStore->FileStore`). This data can be significant, as it is a binlog image of rows inserted/updated/deleted by the transaction. The `wsrep_max_ws_size` variable controls the size of this part of the write-set. The threshold doesn't consider size allocated for caching-keys and the header.

If `FileStore` is used, it creates a file on the disk (with names like `xxxx_keys` and `xxxx_data`) to store the cache data. These files are kept until a transaction is committed, so the lifetime of the transaction is linked.

When the node is done with the transaction and is about to commit, it will generate the final-write-set using the two files (if the data size grew enough to use `FileStore`) plus `HEADER`, and will publish it for certification to cluster.

The native node executing the transaction will also act as subscription node, and will receive its own write-set through the cluster publish mechanism. This time, the native node will try to cache write-set into its GCache. How much data GCache retains is controlled by the GCache configuration.

4.9.2 GCache

GCache holds the write-set published on the cluster for replication. The lifetime of write-set in GCache is not transaction-linked.

When a `JOINER` node needs an IST, it will be serviced through this GCache (if possible).

GCache will also create the files to disk. You can read more about it here.

At any given point in time, the native node has two copies of the write-set: one in GCache and another in Record-Set Cache.

For example, lets say you `INSERT/UPDATE` 2 million rows in a table with the following schema.

```
(int, char(100), char(100) with pk (int, char(100))
```

It will create write-set key/data files in the background similar to the following:

```
-rw------- 1 xxx xxx 67108864 Apr 11 12:26 0x00000707_data.000000
-rw------- 1 xxx xxx 67108864 Apr 11 12:26 0x00000707_data.000001
-rw------- 1 xxx xxx 67108864 Apr 11 12:26 0x00000707_data.000002
-rw------- 1 xxx xxx 67108864 Apr 11 12:26 0x00000707_keys.000000
```

CHAPTER

FIVE

HOW-TOS

5.1 Installing Percona XtraDB Cluster on CentOS

This tutorial describes how to install and configure three Percona XtraDB Cluster nodes on CentOS 6.3 servers, using the packages from Percona repositories.

- Node 1
  - Host name: `percona1`
  - IP address: `192.168.70.71`
- Node 2
  - Host name: `percona2`
  - IP address: `192.168.70.72`
- Node 3
  - Host name: `percona3`
  - IP address: `192.168.70.73`

5.1.1 Prerequisites

The procedure described in this tutorial requires the following:

- All three nodes have CentOS 6.3 installed.
- The firewall on all nodes is configured to allow connecting to ports 3306, 4444, 4567 and 4568.
- SELinux on all nodes is disabled.

5.1.2 Step 1. Installing PXC

Install Percona XtraDB Cluster on all three nodes as described in *Installing Percona XtraDB Cluster on Red Hat Enterprise Linux and CentOS*.

5.1.3 Step 2. Configuring the first node

Individual nodes should be configured to be able to bootstrap the cluster. For more information about bootstrapping the cluster, see *Bootstrapping the cluster*.

1. Make sure that the configuration file `/etc/my.cnf` on the first node (`percona1`) contains the following:

```
[mysqld]

datadir=/var/lib/mysql
user=mysql

# Path to Galera library
wsrep_provider=/usr/lib64/libgalera_smm.so

# Cluster connection URL contains the IPs of node#1, node#2 and node#3
wsrep_cluster_address=gcomm://192.168.70.71,192.168.70.72,192.168.70.73

# In order for Galera to work correctly binlog format should be ROW
binlog_format=ROW

# MyISAM storage engine has only experimental support
default_storage_engine=InnoDB

# This InnoDB autoincrement locking mode is a requirement for Galera
innodb_autoinc_lock_mode=2

# Node 1 address
wsrep_node_address=192.168.70.71

# SST method
wsrep_sst_method=xtrabackup-v2

# Cluster name
wsrep_cluster_name=my_centos_cluster

# Authentication for SST method
wsrep_sst_auth="sstuser:s3cret"
```

2. Start the first node with the following command:

```
[root@percona1 ~]# /etc/init.d/mysql bootstrap-pxc
```

Note: In case you're running CentOS 7, the bootstrap service should be used instead:

```
[root@percona1 ~]#  systemctl start mysql@bootstrap.service
```

The previous command will start the cluster with initial `wsrep_cluster_address` variable set to `gcomm://`. If the node or *MySQL* are restarted later, there will be no need to change the configuration file.

3. After the first node has been started, cluster status can be checked with the following command:

```
mysql> show status like 'wsrep%';
+----------------------------+--------------------------------------+
| Variable_name              | Value                                |
+----------------------------+--------------------------------------+
| wsrep_local_state_uuid     | c2883338-834d-11e2-0800-03c9c68e41ec |
...
| wsrep_local_state          | 4                                    |
| wsrep_local_state_comment  | Synced                               |
...
| wsrep_cluster_size         | 1                                    |
| wsrep_cluster_status       | Primary                              |
| wsrep_connected            | ON                                   |
...
```

```
| wsrep_ready                | ON                                    |
+----------------------------+---------------------------------------+
40 rows in set (0.01 sec)
```

This output shows that the cluster has been successfully bootstrapped.

Note: It is not recommended to leave an empty password for the root account. Password can be changed as follows:

```
mysql@percona1> UPDATE mysql.user SET password=PASSWORD("Passw0rd") where user='root';
mysql@percona1> FLUSH PRIVILEGES;
```

To perform *State Snapshot Transfer* using *XtraBackup*, set up a new user with proper privileges:

```
mysql@percona1> CREATE USER 'sstuser'@'localhost' IDENTIFIED BY 's3cret';
mysql@percona1> GRANT RELOAD, LOCK TABLES, REPLICATION CLIENT ON *.* TO 'sstuser'@'localhost';
mysql@percona1> FLUSH PRIVILEGES;
```

Note: MySQL root account can also be used for performing SST, but it is more secure to use a different (non-root) user for this.

5.1.4 Step 3. Configuring the second node

1. Make sure that the onfiguration file `/etc/my.cnf` on the second node (`percona2`) contains the following:

```
[mysqld]

datadir=/var/lib/mysql
user=mysql

# Path to Galera library
wsrep_provider=/usr/lib64/libgalera_smm.so

# Cluster connection URL contains IPs of node#1, node#2 and node#3
wsrep_cluster_address=gcomm://192.168.70.71,192.168.70.72,192.168.70.73

# In order for Galera to work correctly binlog format should be ROW
binlog_format=ROW

# MyISAM storage engine has only experimental support
default_storage_engine=InnoDB

# This InnoDB autoincrement locking mode is a requirement for Galera
innodb_autoinc_lock_mode=2

# Node 2 address
wsrep_node_address=192.168.70.72

# Cluster name
wsrep_cluster_name=my_centos_cluster

# SST method
wsrep_sst_method=xtrabackup-v2

#Authentication for SST method
wsrep_sst_auth="sstuser:s3cret"
```

2. Start the second node with the following command:

```
[root@percona2 ~]# /etc/init.d/mysql start
```

1. After the server has been started, it should receive *SST* automatically. This means that the second node won't have empty root password anymore. In order to connect to the cluster and check the status, the root password from the first node should be used. Cluster status can be checked on both nodes. The following is an example of status from the second node (`percona2`):

```
mysql> show status like 'wsrep%';
+----------------------------+--------------------------------------+
| Variable_name              | Value                                |
+----------------------------+--------------------------------------+
| wsrep_local_state_uuid     | c2883338-834d-11e2-0800-03c9c68e41ec |
...
| wsrep_local_state          | 4                                    |
| wsrep_local_state_comment  | Synced                               |
...
| wsrep_cluster_size         | 2                                    |
| wsrep_cluster_status       | Primary                              |
| wsrep_connected            | ON                                   |
...
| wsrep_ready                | ON                                   |
+----------------------------+--------------------------------------+
40 rows in set (0.01 sec)
```

This output shows that the new node has been successfully added to the cluster.

5.1.5 Step 4. Configuring the third node

1. Make sure that the MySQL configuration file `/etc/my.cnf` on the third node (`percona3`) contains the following:

```
[mysqld]

datadir=/var/lib/mysql
user=mysql

# Path to Galera library
wsrep_provider=/usr/lib64/libgalera_smm.so

# Cluster connection URL contains IPs of node#1, node#2 and node#3
wsrep_cluster_address=gcomm://192.168.70.71,192.168.70.72,192.168.70.73

# In order for Galera to work correctly binlog format should be ROW
binlog_format=ROW

# MyISAM storage engine has only experimental support
default_storage_engine=InnoDB

# This InnoDB autoincrement locking mode is a requirement for Galera
innodb_autoinc_lock_mode=2

# Node #3 address
wsrep_node_address=192.168.70.73

# Cluster name
wsrep_cluster_name=my_centos_cluster
```

```
# SST method
wsrep_sst_method=xtrabackup-v2

#Authentication for SST method
wsrep_sst_auth="sstuser:s3cret"
```

2. Start the third node with the following command:

```
[root@percona3 ~]# /etc/init.d/mysql start
```

1. After the server has been started, it should receive SST automatically. Cluster status can be checked on all three nodes. The following is an example of status from the third node (`percona3`):

```
mysql> show status like 'wsrep%';
+----------------------------+--------------------------------------+
| Variable_name              | Value                                |
+----------------------------+--------------------------------------+
| wsrep_local_state_uuid     | c2883338-834d-11e2-0800-03c9c68e41ec |
...
| wsrep_local_state          | 4                                    |
| wsrep_local_state_comment  | Synced                               |
...
| wsrep_cluster_size         | 3                                    |
| wsrep_cluster_status       | Primary                              |
| wsrep_connected            | ON                                   |
...
| wsrep_ready                | ON                                   |
+----------------------------+--------------------------------------+
40 rows in set (0.01 sec)
```

This output confirms that the third node has joined the cluster.

5.1.6 Testing replication

To test replication, lets create a new database on second node, create a table for that database on the third node, and add some records to the table on the first node.

1. Create a new database on the second node:

```
mysql@percona2> CREATE DATABASE percona;
Query OK, 1 row affected (0.01 sec)
```

2. Create a table on the third node:

```
mysql@percona3> USE percona;
Database changed

mysql@percona3> CREATE TABLE example (node_id INT PRIMARY KEY, node_name VARCHAR(30));
Query OK, 0 rows affected (0.05 sec)
```

3. Insert records on the first node:

```
mysql@percona1> INSERT INTO percona.example VALUES (1, 'percona1');
Query OK, 1 row affected (0.02 sec)
```

4. Retrieve all the rows from that table on the second node:

```
mysql@percona2> SELECT * FROM percona.example;
+---------+-----------+
| node_id | node_name |
+---------+-----------+
|       1 | percona1  |
+---------+-----------+
1 row in set (0.00 sec)
```

This simple procedure should ensure that all nodes in the cluster are synchronized and working as intended.

5.2 Installing Percona XtraDB Cluster on Ubuntu

This tutorial describes how to install and configure three Percona XtraDB Cluster nodes on Ubuntu 12.04.2 LTS servers, using the packages from Percona repositories.

- Node 1
  - Host name: `pxc1`
  - IP address: `192.168.70.61`
- Node 2
  - Host name: `pxc2`
  - IP address: `192.168.70.62`
- Node 3
  - Host name: `pxc3`
  - IP address: `192.168.70.63`

5.2.1 Prerequisites

The procedure described in this tutorial requires he following:

- All three nodes have Ubuntu 12.04.2 LTS installed.
- Firewall on all nodes is configured to allow connecting to ports 3306, 4444, 4567 and 4568.
- AppArmor profile for *MySQL* is disabled.

5.2.2 Step 1. Installing PXC

Install Percona XtraDB Cluster on all three nodes as described in *Installing Percona XtraDB Cluster on Debian and Ubuntu*.

Note: Debian/Ubuntu installation prompts for root password. For this tutorial, set it to `Passw0rd`. After the packages have been installed, `mysqld` will start automatically. Stop `mysqld` on all three nodes using `/etc/init.d/mysql stop`.

5.2.3 Step 2. Configuring the first node

Individual nodes should be configured to be able to bootstrap the cluster. For more information about bootstrapping the cluster, see *Bootstrapping the cluster*.

1. Make sure that the configuration file `/etc/mysql/my.cnf` for the first node (`pxc1`) contains the following:

```
[mysqld]

datadir=/var/lib/mysql
user=mysql

# Path to Galera library
wsrep_provider=/usr/lib/libgalera_smm.so

# Cluster connection URL contains the IPs of node#1, node#2 and node#3
wsrep_cluster_address=gcomm://192.168.70.61,192.168.70.62,192.168.70.63

# In order for Galera to work correctly binlog format should be ROW
binlog_format=ROW

# MyISAM storage engine has only experimental support
default_storage_engine=InnoDB

# This InnoDB autoincrement locking mode is a requirement for Galera
innodb_autoinc_lock_mode=2

# Node #1 address
wsrep_node_address=192.168.70.61

# SST method
wsrep_sst_method=xtrabackup-v2

# Cluster name
wsrep_cluster_name=my_ubuntu_cluster

# Authentication for SST method
wsrep_sst_auth="sstuser:s3cretPass"
```

2. Start the first node with the following command:

```
[root@pxc1 ~]# /etc/init.d/mysql bootstrap-pxc
```

   This command will start the first node and bootstrap the cluster.

3. After the first node has been started, cluster status can be checked with the following command:

```
mysql> show status like 'wsrep%';
+----------------------------+--------------------------------------+
| Variable_name              | Value                                |
+----------------------------+--------------------------------------+
| wsrep_local_state_uuid     | b598af3e-ace3-11e2-0800-3e90eb9cd5d3 |
...
| wsrep_local_state          | 4                                    |
| wsrep_local_state_comment  | Synced                               |
...
| wsrep_cluster_size         | 1                                    |
| wsrep_cluster_status       | Primary                              |
| wsrep_connected            | ON                                   |
...
```

```
| wsrep_ready                | ON                                    |
+----------------------------+---------------------------------------+
40 rows in set (0.01 sec)
```

This output shows that the cluster has been successfully bootstrapped.

To perform *State Snapshot Transfer* using *XtraBackup*, set up a new user with proper privileges:

```
mysql@pxc1> CREATE USER 'sstuser'@'localhost' IDENTIFIED BY 's3cretPass';
mysql@pxc1> GRANT RELOAD, LOCK TABLES, REPLICATION CLIENT ON *.* TO 'sstuser'@'localhost';
mysql@pxc1> FLUSH PRIVILEGES;
```

Note: MySQL root account can also be used for performing SST, but it is more secure to use a different (non-root) user for this.

5.2.4 Step 3. Configuring the second node

1. Make sure that the configuration file `/etc/mysql/my.cnf` on the second node (`pxc2`) contains the following:

```
[mysqld]

datadir=/var/lib/mysql
user=mysql

# Path to Galera library
wsrep_provider=/usr/lib/libgalera_smm.so

# Cluster connection URL contains IPs of node#1, node#2 and node#3
wsrep_cluster_address=gcomm://192.168.70.61,192.168.70.62,192.168.70.63

# In order for Galera to work correctly binlog format should be ROW
binlog_format=ROW

# MyISAM storage engine has only experimental support
default_storage_engine=InnoDB

# This InnoDB autoincrement locking mode is a requirement for Galera
innodb_autoinc_lock_mode=2

# Node #2 address
wsrep_node_address=192.168.70.62

# Cluster name
wsrep_cluster_name=my_ubuntu_cluster

# SST method
wsrep_sst_method=xtrabackup-v2

#Authentication for SST method
wsrep_sst_auth="sstuser:s3cretPass"
```

2. Start the second node with the following command:

```
[root@pxc2 ~]# /etc/init.d/mysql start
```

3. After the server has been started, it should receive *SST* automatically. Cluster status can now be checked on both nodes. The following is an example of status from the second node (`pxc2`):

```
mysql> show status like 'wsrep%';
+----------------------------+--------------------------------------+
| Variable_name              | Value                                |
+----------------------------+--------------------------------------+
| wsrep_local_state_uuid     | b598af3e-ace3-11e2-0800-3e90eb9cd5d3 |
...
| wsrep_local_state          | 4                                    |
| wsrep_local_state_comment  | Synced                               |
...
| wsrep_cluster_size         | 2                                    |
| wsrep_cluster_status       | Primary                              |
| wsrep_connected            | ON                                   |
...
| wsrep_ready                | ON                                   |
+----------------------------+--------------------------------------+
40 rows in set (0.01 sec)
```

This output shows that the new node has been successfully added to the cluster.

5.2.5 Step 4. Configuring the third node

1. Make sure that the MySQL configuration file `/etc/mysql/my.cnf` on the third node (`pxc3`) contains the following:

```
[mysqld]

datadir=/var/lib/mysql
user=mysql

# Path to Galera library
wsrep_provider=/usr/lib/libgalera_smm.so

# Cluster connection URL contains IPs of node#1, node#2 and node#3
wsrep_cluster_address=gcomm://192.168.70.61,192.168.70.62,192.168.70.63

# In order for Galera to work correctly binlog format should be ROW
binlog_format=ROW

# MyISAM storage engine has only experimental support
default_storage_engine=InnoDB

# This InnoDB autoincrement locking mode is a requirement for Galera
innodb_autoinc_lock_mode=2

# Node #3 address
wsrep_node_address=192.168.70.63

# Cluster name
wsrep_cluster_name=my_ubuntu_cluster

# SST method
wsrep_sst_method=xtrabackup-v2

#Authentication for SST method
wsrep_sst_auth="sstuser:s3cretPass"
```

2. Start the third node with the following command:

```
[root@pxc3 ~]# /etc/init.d/mysql start
```

3. After the server has been started, it should receive SST automatically. Cluster status can be checked on all nodes. The following is an example of status from the third node (`pxc3`):

```
mysql> show status like 'wsrep%';
+----------------------------+--------------------------------------+
| Variable_name              | Value                                |
+----------------------------+--------------------------------------+
| wsrep_local_state_uuid     | b598af3e-ace3-11e2-0800-3e90eb9cd5d3 |
...
| wsrep_local_state          | 4                                    |
| wsrep_local_state_comment  | Synced                               |
...
| wsrep_cluster_size         | 3                                    |
| wsrep_cluster_status       | Primary                              |
| wsrep_connected            | ON                                   |
...
| wsrep_ready                | ON                                   |
+----------------------------+--------------------------------------+
40 rows in set (0.01 sec)
```

   This output confirms that the third node has joined the cluster.

5.2.6 Testing replication

To test replication, lets create a new database on the second node, create a table for that database on the third node, and add some records to the table on the first node.

1. Create a new database on the second node:

```
mysql@pxc2> CREATE DATABASE percona;
Query OK, 1 row affected (0.01 sec)
```

2. Create a table on the third node:

```
mysql@pxc3> USE percona;
Database changed

mysql@pxc3> CREATE TABLE example (node_id INT PRIMARY KEY, node_name VARCHAR(30));
Query OK, 0 rows affected (0.05 sec)
```

3. Insert records on the first node:

```
mysql@pxc1> INSERT INTO percona.example VALUES (1, 'percona1');
Query OK, 1 row affected (0.02 sec)
```

4. Retrieve all the rows from that table on the second node:

```
mysql@pxc2> SELECT * FROM percona.example;
+---------+-----------+
| node_id | node_name |
+---------+-----------+
|       1 | percona1  |
+---------+-----------+
1 row in set (0.00 sec)
```

This simple procedure should ensure that all nodes in the cluster are synchronized and working as intended.

5.3 Setting up Galera Arbitrator

Galera Arbitrator <http://galeracluster.com/documentation-webpages/arbitrator.html> is a member of the *Percona XtraDB Cluster* which is used for voting in case you have a small number of servers (usually two) and don't want to add any more resources. For example arbitrator can be set up on existing application server or on load balancer.

Galera Arbitrator is a member of the cluster which participates in the voting, but not in the actual replication (although it receives the same data as other nodes).

This document will show how to add Galera Arbitrator node to a already set up cluster.

Note: For more information on how to set up a cluster you can read in the *Installing Percona XtraDB Cluster on Ubuntu* or *Installing Percona XtraDB Cluster on CentOS* manuals.

5.3.1 Installation

Galera Arbitrator can be installed from Percona's repository by running:

```
root@ubuntu:~# apt-get install percona-xtradb-cluster-garbd-5.7
```

on Debian/Ubuntu distributions, or:

```
[root@centos ~]# yum install Percona-XtraDB-Cluster-garbd-57
```

on CentOS/RHEL distributions.

5.3.2 Configuration

To configure *Galera Arbitrator* on *Ubuntu/Debian* you need to edit the `/etc/default/garbd` file. On *CentOS/RHEL* configuration can be found in `/etc/sysconfig/garb` file.

Configuration file should look like this after installation:

```
# Copyright (C) 2012 Codership Oy
# This config file is to be sourced by garb service script.

# REMOVE THIS AFTER CONFIGURATION

# A comma-separated list of node addresses (address[:port]) in the cluster
# GALERA_NODES=""

# Galera cluster name, should be the same as on the rest of the nodes.
# GALERA_GROUP=""

# Optional Galera internal options string (e.g. SSL settings)
# see http://galeracluster.com/documentation-webpages/galeraparameters.html
# GALERA_OPTIONS=""

# Log file for garbd. Optional, by default logs to syslog
# Deprecated for CentOS7, use journalctl to query the log for garbd
# LOG_FILE=""
```

To set it up you'll need to add the information about the cluster you've set up. This example is using cluster information from the *Installing Percona XtraDB Cluster on Ubuntu*.

```
# Copyright (C) 2012 Codership Oy
# This config file is to be sourced by garb service script.

# A comma-separated list of node addresses (address[:port]) in the cluster
GALERA_NODES="192.168.70.61:4567, 192.168.70.62:4567, 192.168.70.63:4567"

# Galera cluster name, should be the same as on the rest of the nodes.
GALERA_GROUP="my_ubuntu_cluster"

# Optional Galera internal options string (e.g. SSL settings)
# see http://galeracluster.com/documentation-webpages/galeraparameters.html
# GALERA_OPTIONS=""

# Log file for garbd. Optional, by default logs to syslog
# Deprecated for CentOS7, use journalctl to query the log for garbd
# LOG_FILE=""
```

Note: Please note that you need to remove the `# REMOVE THIS AFTER CONFIGURATION` line before you can start the service.

You can now start the *Galera Arbitrator* daemon (`garbd`) by running:

```
root@server:~# service garbd start
[ ok ] Starting /usr/bin/garbd: :.
```

You can additionally check the `garbd` status by running:

```
root@server:~# service garbd status
[ ok ] garb is running.
```

5.4 How to set up a three-node cluster on a single box

This tutorial describes how to set up a 3-node cluster on a single physical box.

For the purposes of this tutorial, assume the following:

- The local IP address is `192.168.2.21`.
- Percona XtraDB Cluster is extracted from binary tarball into `/usr/local/Percona-XtraDB-Cluster-5.7.11-rel4be`

To set up the cluster:

1. Create three MySQL configuration files for the corresponding nodes:
   - `/etc/my.4000.cnf`

     ```
     [mysqld]
     port = 4000
     socket=/tmp/mysql.4000.sock
     datadir=/data/bench/d1
     basedir=/usr/local/Percona-XtraDB-Cluster-5.7.11-rel4beta-25.14.2.beta.Linux.x86_64
     user=mysql
     log_error=error.log
     binlog_format=ROW
     wsrep_cluster_address='gcomm://192.168.2.21:5030,192.168.2.21:6030'
     wsrep_provider=/usr/local/Percona-XtraDB-Cluster-5.7.11-rel4beta-25.14.2.beta.Linux.x86_64/l
     wsrep_sst_receive_address=192.168.2.21:4020
     wsrep_node_incoming_address=192.168.2.21
     ```

```
wsrep_slave_threads=2
wsrep_cluster_name=trimethylxanthine
wsrep_provider_options = "gmcast.listen_addr=tcp://192.168.2.21:4030;"
wsrep_sst_method=rsync
wsrep_node_name=node4000
innodb_autoinc_lock_mode=2
```

- `/etc/my.5000.cnf`

```
[mysqld]
port = 5000
socket=/tmp/mysql.5000.sock
datadir=/data/bench/d2
basedir=/usr/local/Percona-XtraDB-Cluster-5.7.11-rel4beta-25.14.2.beta.Linux.x86_64
user=mysql
log_error=error.log
binlog_format=ROW
wsrep_cluster_address='gcomm://192.168.2.21:4030,192.168.2.21:6030'
wsrep_provider=/usr/local/Percona-XtraDB-Cluster-5.7.11-rel4beta-25.14.2.beta.Linux.x86_64/l
wsrep_sst_receive_address=192.168.2.21:5020
wsrep_node_incoming_address=192.168.2.21
wsrep_slave_threads=2
wsrep_cluster_name=trimethylxanthine
wsrep_provider_options = "gmcast.listen_addr=tcp://192.168.2.21:5030;"
wsrep_sst_method=rsync
wsrep_node_name=node5000
innodb_autoinc_lock_mode=2
```

- `/etc/my.6000.cnf`

```
[mysqld]
port = 6000
socket=/tmp/mysql.6000.sock
datadir=/data/bench/d3
basedir=/usr/local/Percona-XtraDB-Cluster-5.7.11-rel4beta-25.14.2.beta.Linux.x86_64
user=mysql
log_error=error.log
binlog_format=ROW
wsrep_cluster_address='gcomm://192.168.2.21:4030,192.168.2.21:5030'
wsrep_provider=/usr/local/Percona-XtraDB-Cluster-5.7.11-rel4beta-25.14.2.beta.Linux.x86_64/l
wsrep_sst_receive_address=192.168.2.21:6020
wsrep_node_incoming_address=192.168.2.21
wsrep_slave_threads=2
wsrep_cluster_name=trimethylxanthine
wsrep_provider_options = "gmcast.listen_addr=tcp://192.168.2.21:6030;"
wsrep_sst_method=rsync
wsrep_node_name=node6000
innodb_autoinc_lock_mode=2
```

2. Create three data directories for the nodes:
   - `/data/bench/d1`
   - `/data/bench/d2`
   - `/data/bench/d3`
3. Start the first node using the following command (from the Percona XtraDB Cluster install directory):

```
bin/mysqld_safe --defaults-file=/etc/my.4000.cnf --wsrep-new-cluster
```

If the node starts correctly, you should see the following output:

```
111215 19:01:49 [Note] WSREP: Shifting JOINED -> SYNCED (TO: 0)
111215 19:01:49 [Note] WSREP: New cluster view: global state: 4c286ccc-2792-11e1-0800-94bd91e32e
```

To check the ports, run the following command:

```
$ netstat -anp | grep mysqld
tcp        0      0 192.168.2.21:4030           0.0.0.0:*                   LISTEN      21895/my
tcp        0      0 0.0.0.0:4000                0.0.0.0:*                   LISTEN      21895/my
```

4. Start the second and third nodes:

```
bin/mysqld_safe --defaults-file=/etc/my.5000.cnf
bin/mysqld_safe --defaults-file=/etc/my.6000.cnf
```

If the nodes start and join the cluster successful, you should see the following output:

```
111215 19:22:26 [Note] WSREP: Shifting JOINER -> JOINED (TO: 2)
111215 19:22:26 [Note] WSREP: Shifting JOINED -> SYNCED (TO: 2)
111215 19:22:26 [Note] WSREP: Synchronized with group, ready for connections
```

To check the cluster size, run the following command:

```
mysql -h127.0.0.1 -P6000 -e "show global status like 'wsrep_cluster_size';"
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| wsrep_cluster_size | 3     |
+--------------------+-------+
```

After that you can connect to any node and perform queries, which will be automatically synchronized with other nodes. For example, to create a database on the second node, you can run the following command:

5.5 How to set up a three-node cluster in EC2 enviroment

This manual assumes you are running *m1.xlarge* instances with Red Hat Enterprise Linux 6.1 64-bit.

- node1: 10.93.46.58
- node2: 10.93.46.59
- node3: 10.93.46.60

To set up Percona XtraDB Cluster:

1. Remove any Percona XtraDB Cluster 5.5, *Percona Server* 5.5, and *Percona Server* 5.6 packages.
2. nstall Percona XtraDB Cluster as described in *Installing Percona XtraDB Cluster on Red Hat Enterprise Linux and CentOS*.
3. Create data directories:

```
mkdir -p /mnt/data
mysql_install_db --datadir=/mnt/data --user=mysql
```

4. Stop the firewall service:

```
service iptables stop
```

Note: Alternatively, you can keep the firewall running, but open ports 3306, 4444, 4567, 4568. For example to open port 4567 on 192.168.0.1:

```
iptables -A INPUT -i eth0 -p tcp -m tcp --source 192.168.0.1/24 --dport 4567 -j ACCEPT
```

---

5. Create `/etc/my.cnf` files:

   Contents of the configuration file on the first node:

```
[mysqld]
datadir=/mnt/data
user=mysql

binlog_format=ROW

wsrep_provider=/usr/lib64/libgalera_smm.so
wsrep_cluster_address=gcomm://10.93.46.58,10.93.46.59,10.93.46.60

wsrep_slave_threads=2
wsrep_cluster_name=trimethylxanthine
wsrep_sst_method=rsync
wsrep_node_name=node1

innodb_autoinc_lock_mode=2
```

   For the second and third nodes change the following lines:

```
wsrep_node_name=node2

wsrep_node_name=node3
```

6. Start and bootstrap Percona XtraDB Cluster on the first node:

```
[root@node1 ~]# /etc/init.d/mysql bootstrap-pxc
```

   You should see the following output:

```
2014-01-30 11:52:35 23280 [Note] /usr/sbin/mysqld: ready for connections.
Version: '5.6.15-56'  socket: '/var/lib/mysql/mysql.sock'  port: 3306  Percona XtraDB Cluster (G
```

7. Start the second and third nodes:

```
[root@node2 ~]# /etc/init.d/mysql start
```

   You should see the following output:

```
2014-01-30 09:52:42 26104 [Note] WSREP: Flow-control interval: [28, 28]
2014-01-30 09:52:42 26104 [Note] WSREP: Restored state OPEN -> JOINED (2)
2014-01-30 09:52:42 26104 [Note] WSREP: Member 2 (percona1) synced with group.
2014-01-30 09:52:42 26104 [Note] WSREP: Shifting JOINED -> SYNCED (TO: 2)
2014-01-30 09:52:42 26104 [Note] WSREP: New cluster view: global state: 4827a206-876b-11e3-911c-
2014-01-30 09:52:42 26104 [Note] WSREP: SST complete, seqno: 2
2014-01-30 09:52:42 26104 [Note] Plugin 'FEDERATED' is disabled.
2014-01-30 09:52:42 26104 [Note] InnoDB: The InnoDB memory heap is disabled
2014-01-30 09:52:42 26104 [Note] InnoDB: Mutexes and rw_locks use GCC atomic builtins
2014-01-30 09:52:42 26104 [Note] InnoDB: Compressed tables use zlib 1.2.3
```

```
2014-01-30 09:52:42 26104 [Note] InnoDB: Using Linux native AIO
2014-01-30 09:52:42 26104 [Note] InnoDB: Not using CPU crc32 instructions
2014-01-30 09:52:42 26104 [Note] InnoDB: Initializing buffer pool, size = 128.0M
2014-01-30 09:52:42 26104 [Note] InnoDB: Completed initialization of buffer pool
2014-01-30 09:52:43 26104 [Note] InnoDB: Highest supported file format is Barracuda.
2014-01-30 09:52:43 26104 [Note] InnoDB: 128 rollback segment(s) are active.
2014-01-30 09:52:43 26104 [Note] InnoDB: Waiting for purge to start
2014-01-30 09:52:43 26104 [Note] InnoDB:  Percona XtraDB (http://www.percona.com) 5.6.15-rel62.0
2014-01-30 09:52:43 26104 [Note] RSA private key file not found: /var/lib/mysql//private_key.pem
2014-01-30 09:52:43 26104 [Note] RSA public key file not found: /var/lib/mysql//public_key.pem.
2014-01-30 09:52:43 26104 [Note] Server hostname (bind-address): '*'; port: 3306
2014-01-30 09:52:43 26104 [Note] IPv6 is available.
2014-01-30 09:52:43 26104 [Note]   - '::' resolves to '::';
2014-01-30 09:52:43 26104 [Note] Server socket created on IP: '::'.
2014-01-30 09:52:43 26104 [Note] Event Scheduler: Loaded 0 events
2014-01-30 09:52:43 26104 [Note] /usr/sbin/mysqld: ready for connections.
Version: '5.6.15-56'  socket: '/var/lib/mysql/mysql.sock'  port: 3306  Percona XtraDB Cluster (G
2014-01-30 09:52:43 26104 [Note] WSREP: inited wsrep sidno 1
2014-01-30 09:52:43 26104 [Note] WSREP: wsrep_notify_cmd is not defined, skipping notification.
2014-01-30 09:52:43 26104 [Note] WSREP: REPL Protocols: 5 (3, 1)
2014-01-30 09:52:43 26104 [Note] WSREP: Assign initial position for certification: 2, protocol v
2014-01-30 09:52:43 26104 [Note] WSREP: Service thread queue flushed.
2014-01-30 09:52:43 26104 [Note] WSREP: Synchronized with group, ready for connections
```

When all nodes are in SYNCED state, your cluster is ready.

8. You can try connecting to MySQL on any node and create a database:

```
$ mysql -uroot
> CREATE DATABASE hello_tom;
```

The new database will be propagated to all nodes.

5.6 Encrypting PXC Traffic

Percona XtraDB Cluster supports encryption for all traffic involved in cluster operation.

5.6.1 Client-Server Communication

This refers to communication between client applications and cluster nodes. To secure client connections, you need to generate and use SSL keys and certificates.

The following example shows `my.cnf` configuration for server nodes and client instances to use SSL:

```
[mysqld]
ssl-ca=ca.pem
ssl-cert=server-cert.pem
ssl-key=server-key.pem

[client]
```

```
ssl-ca=ca.pem
ssl-cert=client-cert.pem
ssl-key=client-key.pem
```

For more information, see the relevant sections about SSL certificates in Galera Cluster Documentation. and MySQL Server Documentation.

5.6.2 SST Traffic

This refers to full data transfer that usually occurs when a new node (JOINER) joins the cluster and receives data from an existing node (DONOR).

For more information, see *State Snapshot Transfer*.

When copying encrypted data via SST, the keyring must be sent over with the files for decryption. Make sure that the following options are set in `my.cnf` on all nodes:

```
early-plugin-load=keyring_file.so
keyring-file-data=/path/to/keyring/file
```

Warning: The cluster will not work if keyring configuration across nodes is different.

The following SST methods are available: `rsync`, `mysqldump`, and `xtrabackup`.

rsync

This SST method does not support encryption. Avoid using this method if you need to secure traffic between DONOR and JOINER nodes.

mysqldump

This SST method dumps data from DONOR and imports it to JOINER. Encryption in this case is performed using the SSL certificates configured for *secure MySQL client-server communication*.

Here is how to perform secure SST using `mysqldump`:

1. Ensure that the DONOR node is configured for SSL encryption (both `[mysqld]` and `[client]` sections). For more information, see *Client-Server Communication*.
2. Create an SSL user on the DONOR node.

   ```
   mysql> GRANT USAGE ON *.* TO 'sslsst' REQUIRE SSL;
   ```

3. Specify the DONOR superuser credentials in the `wsrep_sst_auth` variable.
4. Start the JOINER node without Galera library (`--wsrep_provider=none`) and create an SSL user with the same name and grants as on the DONOR node.
5. Configure SSL encryption on JOINER node with the same parameters as DONOR (both `[mysqld]` and `[client]` sections).
6. Restart JOINER node with Galera library.

If you do everything correctly, `mysqldump` will connect to DONOR using SSL user, generate a dump file, and import it to JOINER node.

For more information, see the relevant section in Galera Cluster documentation.

xtrabackup

This is the default SST method, which uses Percona XtraBackup to perform non-blocking transfer of files. For more information, see *Percona XtraBackup SST Configuration*.

Encryption mode for this method is selected using the `encrypt` option. Depending on the mode you select, other options will be required.

- To enable built-in XtraBackup encryption, use the following configuration:

  ```
  [sst]
  encrypt=1
  encrypt-algo=AES256
  encrypt-key=A1EDC73815467C083B0869508406637E
  ```

  In this example, you can set `encrypt-key-file` instead of `encrypt-key`.

  For more information, see Encrypted Backups.

- To enable SST encryption based on OpenSSL with the certificate authority (`tca`) and certificate (`tcert`) files:

  ```
  [sst]
  encrypt=2
  tcert=/path/to/server.pem
  tca=/path/to/server.crt
  ```

  For more information, see *Securing Traffic Between two Socat Instances Using SSL <http://www.dest-unreach.org/socat/doc/socat-openssltunnel.html>*.

- To enable SST encryption based on OpenSSL with the key (`tkey`) and certificate (`tcert`) files:

  ```
  [sst]
  encrypt=2
  tcert=/path/to/server.pem
  tkey=/path/to/server.key
  ```

Note: Percona XtraBackup supports keyring transfer in version 2.4.4 and later.

5.6.3 IST Traffic, Write-Set Replication, and Service Messages

IST refers to transferring only missing transactions from DONOR to JOINER node. Write-set replication is the main workload in Percona XtraDB Cluster whenever a transaction is performed on one node, it is replicated to all other nodes. Service messages ensure that all nodes are synchronized.

All of this traffic is transferred via the same underlying communication channel used by Galera (`gcomm`). Securing this channel will ensure that IST traffic, write-set replication, and service messages are encypted.

To enable SSL for all internal node processes, define the paths to the key, certificate and certificate authority files using the following parameters.

- `socket.ssl_key`
- `socket.ssl_cert`
- `socket.ssl_ca`

To set these parameters, use the `wsrep_provider_options` variable.

```
wsrep_provider_options="socket.ssl=yes;socket.ssl_key=/path/to/server-key.pem;socket.ssl_cert=/path/t
```

For more information, see Index of wsrep provider options.

Note: You must use the same key and certificate files on all nodes, preferably those used for *Client-Server Communication*.

Upgrading Certificates

The following example shows how to upgrade certificates used for securing IST traffic, write-set replication, and service messages, assumig there are two nodes in the cluster:

1. Restart Node 1 with a `socket.ssl_ca` that includes both the new and the old certificates in a single file.

   For example, you can merge contents of `old-ca.pem` and `new-ca.pem` into `upgrade-ca.pem` as follows:

   ```
   cat old-ca.pem > upgrade-ca.pem && cat new-ca.pem >> upgrade-ca.pem
   ```

   Set the `wsrep_provider_options` variable similar to the following:

   ```
   wsrep_provider_options=socket.ssl=yes;socket.ssl_ca=/path/to/upgrade-ca.pem;socket.ssl_cert=path
   ```

2. Restart Node 2 with the new `socket.ssl_ca`, `socket.ssl_cert`, and `socket.ssl_key`.

   ```
   wsrep_provider_options=socket.ssl=yes;socket.ssl_ca=/path/to/upgrade-ca.pem;socket.ssl_cert=/pat
   ```

3. Restart Node 1 with the new `socket.ssl_ca`, `socket.ssl_cert`, and `socket.ssl_key`.

   ```
   wsrep_provider_options=socket.ssl=yes;socket.ssl_ca=/path/to/upgrade-ca.pem;socket.ssl_cert=/pat
   ```

5.7 Load balancing with HAProxy

This manual describes how to configure HAProxy to work with Percona XtraDB Cluster.

The following is an example of the configuration file for HAProxy:

```
# this config requires haproxy-1.4.20

global
        log 127.0.0.1   local0
        log 127.0.0.1   local1 notice
        maxconn 4096
        uid 99
        gid 99
        daemon
        #debug
        #quiet

defaults
        log     global
        mode    http
        option  tcplog
        option  dontlognull
        retries 3
        redispatch
        maxconn 2000
        contimeout      5000
```

```
        clitimeout      50000
        srvtimeout      50000

listen mysql-cluster 0.0.0.0:3306
    mode tcp
    balance roundrobin
    option mysql-check user root

    server db01 10.4.29.100:3306 check
    server db02 10.4.29.99:3306 check
    server db03 10.4.29.98:3306 check
```

With this configuration, HAProxy will balance the load between three nodes. In this case, it only checks if `mysqld` listens on port 3306, but it doesn't take into an account the state of the node. So it could be sending queries to the node that has `mysqld` running even if it's in `JOINING` or `DISCONNECTED` state.

To check the current status of a node we need a more complex check. This idea was taken from codership-team google groups.

To implement this setup, you will need two scripts:

- **clustercheck** (located in `/usr/local/bin`) and a config for `xinetd`
- **mysqlchk** (located in `/etc/xinetd.d`) on each node

Both scripts are available in binaries and source distributions of Percona XtraDB Cluster.

Change the `/etc/services` file by adding the following line on each node:

```
mysqlchk        9200/tcp                # mysqlchk
```

The following is an example of the HAProxy configuration file in this case:

```
# this config needs haproxy-1.4.20

global
        log 127.0.0.1   local0
        log 127.0.0.1   local1 notice
        maxconn 4096
        uid 99
        gid 99
        #daemon
        debug
        #quiet

defaults
        log     global
        mode    http
        option  tcplog
        option  dontlognull
        retries 3
        redispatch
        maxconn 2000
        contimeout      5000
        clitimeout      50000
        srvtimeout      50000

listen mysql-cluster 0.0.0.0:3306
    mode tcp
    balance roundrobin
    option  httpchk
```

```
server db01 10.4.29.100:3306 check port 9200 inter 12000 rise 3 fall 3
server db02 10.4.29.99:3306 check port 9200 inter 12000 rise 3 fall 3
server db03 10.4.29.98:3306 check port 9200 inter 12000 rise 3 fall 3
```

5.8 Load balancing with ProxySQL

ProxySQL is a high-performance SQL proxy. ProxySQL runs as a daemon watched by a monitoring process. The process monitors the daemon and restarts it in case of a crash to minimize downtime.

The daemon accepts incoming traffic from *MySQL* clients and forwards it to backend *MySQL* servers.

The proxy is designed to run continuously without needing to be restarted. Most configuration can be done at runtime using queries similar to SQL statements. These include runtime parameters, server grouping, and traffic-related settings.

ProxySQL supports Percona XtraDB Cluster node status check using scheduler.

Note: For more information about ProxySQL, see ProxySQL documentation.

5.8.1 Installing ProxySQL

ProxySQL is available from the Percona software repositories. If that is what you used to *install PXC* or any other Percona software, run the corresponding command:

- On Debian or Ubuntu:

```
$ sudo apt-get install proxysql
```

- On Red Hat Enterprise Linux or CentOS:

```
$ sudo yum install proxysql
```

Alternatively, you can download packages from https://www.percona.com/downloads/proxysql/.

To start ProxySQL, run the following command:

```
$ sudo service proxysql start
```

Warning: Do not run ProxySQL with default credentials in production.
Before starting the `proxysql` service, you can change the defaults in the `/etc/proxysql.cnf` file by changing the `admin_credentials` variable. For more information, see Global Variables.

5.8.2 Automatic Configuration

The `proxysql` package from Percona includes the `proxysql-admin` tool for configuring Percona XtraDB Cluster nodes with ProxySQL:

```
Usage: proxysql-admin [ options ]

Options:
 --config-file                  Override login credentials from command line and read login credenti
 --proxysql-user=user_name      User to use when connecting to the ProxySQL service
 --proxysql-password[=password] Password to use when connecting to the ProxySQL service
```

```
 --proxysql-port=port_num        Port to use when connecting to the ProxySQL service
 --proxysql-host=host_name       Hostname to use when connecting to the ProxySQL service
 --cluster-user=user_name        User to use when connecting to the Percona XtraDB Cluster node
 --cluster-password[=password]   Password to use when connecting to the Percona XtraDB Cluster node
 --cluster-port=port_num         Port to use when connecting to the Percona XtraDB Cluster node
 --cluster-host=host_name        Hostname to use when connecting to the Percona XtraDB Cluster node
 --monitor-user=user_name        User to use for monitoring Percona XtraDB Cluster nodes through Prox
 --monitor-password[=password]   Password to for monitoring Percona XtraDB Cluster nodes through Prox
 --pxc-app-user=user_name        Application user to use when connecting to the Percona XtraDB Cluste
 --pxc-app-password[=password]   Application password to use when connecting to the Percona XtraDB Cl
 --enable                        Auto-configure Percona XtraDB Cluster nodes into ProxySQL
 --disable                       Remove Percona XtraDB Cluster configurations from ProxySQL
 --galera-check-interval         Interval for monitoring proxysql_galera_checker script(in millisecon
 --mode                          ProxySQL read/write configuration mode, currently it only support 'l
 --adduser                       Add Percona XtraDB Cluster application user to ProxySQL database
```

Note: Before using the `proxysql-admin` tool, ensure that ProxySQL and Percona XtraDB Cluster nodes you want to add are running.

Enabling ProxySQL

Use the `--enable` option to automatically configure a Percona XtraDB Cluster node into ProxySQL. The `proxysql-admin` tool will do the following:

- Add Percona XtraDB Cluster node into the ProxySQL database
- Add the following monitoring scripts into the ProxySQL `scheduler` table, if they are not available:
  - `proxysql_node_monitor` checks cluster node membership and re-configures ProxySQL if the membership changes
  - `proxysql_galera_checker` checks for desynced nodes and temporarily deactivates them
- Create two new Percona XtraDB Cluster users with the `USAGE` privilege on the node and add them to ProxySQL configuration, if they are not already configured. One user is for monitoring cluster nodes, and the other one is for communicating with the cluster.

The following example shows how to add a Percona XtraDB Cluster node with IP address 10.101.6.1 to ProxySQL:

```
$ ./proxysql-admin --proxysql-user=admin --proxysql-password=admin \
     --proxysql-port=6032 --proxysql-host=127.0.0.1 \
     --cluster-user=root --cluster-password=root \
     --cluster-port=3306 --cluster-host=10.101.6.1 \
     --enable

Configuring ProxySQL monitoring user..
Enter ProxySQL monitoring username: monitor
Enter ProxySQL monitoring password:

User monitor@'%' has been added with USAGE privilege

Adding the Percona XtraDB Cluster server nodes to ProxySQL

Configuring Percona XtraDB Cluster user to connect through ProxySQL
Enter Percona XtraDB Cluster user name: proxysql_user
Enter Percona XtraDB Cluster user password:

User proxysql_user@'%' has been added with USAGE privilege, please make sure to grant appropriate pri
```

```
Percona XtraDB Cluster ProxySQL monitoring daemon started
ProxySQL configuration completed!
```

Note: The previous example uses default ProxySQL credentials, host name (in this case localhost) and port number. It is recommended to *change the default credentials* before running ProxySQL in production.

Note: You must provide superuser credentials for the Percona XtraDB Cluster node you are adding.

Disabling ProxySQL

Use the `--disable` option to remove a Percona XtraDB Cluster node's configuration from ProxySQL. The `proxysql-admin` tool will do the following:

- Remove Percona XtraDB Cluster node from the ProxySQL database
- Stop the ProxySQL monitoring daemon for this node

The following example shows how to disable ProxySQL and remove the Percona XtraDB Cluster node added in the previous example:

```
$ ./proxysql-admin --proxysql-user=admin --proxysql-password=admin \
     --proxysql-port=6032 --proxysql-host=127.0.0.1 \
     --cluster-user=root --cluster-password=root \
     --cluster-port=3306 --cluster-host=10.101.6.1 \
     --disable

ProxySQL configuration removed!
```

Additional Options

The following extra options can be used:

- `--adduser`

  Add Percona XtraDB Cluster application user to ProxySQL database.

- `--galera-check-interval`

  Set the interval for monitoring `proxysql_galera_checker` script (in milliseconds).

- `--mode`

  Set the read/write mode for Percona XtraDB Cluster nodes in ProxySQL database, based on the hostgroup. For now, the only supported mode is `loadbal` which will be the default for a load balanced set of evenly weighted read/write nodes.

```
$ ./proxysql-admin --proxysql-user=admin --proxysql-password=admin  \
     --proxysql-port=6032 --proxysql-host=127.0.0.1 \
     --cluster-user=root --cluster-password=root \
     --cluster-port=26000 --cluster-host=10.101.6.1 \
     --galera-check-interval=3000 --adduser

Adding Percona XtraDB Cluster application user to ProxySQL database
Enter Percona XtraDB Cluster application user name: app_read
Enter Percona XtraDB Cluster application user password:
```

```
Application app_read does not exists in Percona XtraDB Cluster. Would you like to proceed [y/n] ? y

Added Percona XtraDB Cluster application user to ProxySQL database!
```

5.8.3 Manual Configuration

This tutorial describes how to configure ProxySQL with three Percona XtraDB Cluster nodes.

Node	Host Name	IP address
Node 1	pxc1	192.168.70.61
Node 2	pxc2	192.168.70.62
Node 3	pxc3	192.168.70.63
Node 4	proxysql	192.168.70.64

ProxySQL can be configured either using the `/etc/proxysql.cnf` file or through the admin interface. Using the admin interface is preferable, because it allows you to change the configuration dynamically (without having to restart the proxy).

To connect to the ProxySQL admin interface, you need a `mysql` client. You can either connect to the admin interface from Percona XtraDB Cluster nodes that already have the `mysql` client installed (Node 1, Node 2, Node 3) or install the client on Node 4 and connect locally. For this tutorial, install Percona XtraDB Cluster on Node 4:

- On Debian or Ubuntu:

  ```
  root@proxysql:~# apt-get install percona-xtradb-cluster-client-5.7
  ```

- On Red Hat Enterprise Linux or CentOS:

  ```
  [root@proxysql ~]# yum install Percona-XtraDB-Cluster-client-57
  ```

To connect to the admin interface, use the credentials, host name and port specified in the global variables.

Warning: Do not use default credentials in production!

The following example shows how to connect to the ProxySQL admin interface with default credentials:

```
root@proxysql:~# mysql -uadmin -padmin -h127.0.0.1 -P6032

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.1.30 (ProxySQL Admin Module)

Copyright (c) 2009-2016 Percona LLC and/or its affiliates
Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql@proxysql>
```

To see the ProxySQL databases and tables use the following commands:

```
mysql@proxysql> SHOW DATABASES;
+-----+---------+-------------------------------+
```

```
| seq | name    | file                            |
+-----+---------+---------------------------------+
| 0   | main    |                                 |
| 2   | disk    | /var/lib/proxysql/proxysql.db |
| 3   | stats   |                                 |
| 4   | monitor |                                 |
+-----+---------+---------------------------------+
4 rows in set (0.00 sec)

mysql@proxysql> SHOW TABLES;
+--------------------------------------+
| tables                               |
+--------------------------------------+
| global_variables                     |
| mysql_collations                     |
| mysql_query_rules                    |
| mysql_replication_hostgroups         |
| mysql_servers                        |
| mysql_users                          |
| runtime_global_variables             |
| runtime_mysql_query_rules            |
| runtime_mysql_replication_hostgroups |
| runtime_mysql_servers                |
| runtime_scheduler                    |
| scheduler                            |
+--------------------------------------+
12 rows in set (0.00 sec)
```

For more information about admin databases and tables, see Admin Tables

---

Note: ProxySQL has 3 areas where the configuration can reside:

- MEMORY (your current working place)
- RUNTIME (the production settings)
- DISK (durable configuration, saved inside an SQLITE database)

When you change a parameter, you change it in MEMORY area. That is done by design to allow you to test the changes before pushing to production (RUNTIME), or save them to disk.

Adding cluster nodes to ProxySQL

To configure the backend Percona XtraDB Cluster nodes in ProxySQL, insert corresponding records into the `mysql_servers` table.

Note: ProxySQL uses the concept of *hostgroups* to group cluster nodes. This enables you to balance the load in a cluster by routing different types of traffic to different groups. There are many ways you can configure hostgroups (for example master and slaves, read and write load, etc.) and a every node can be a member of multiple hostgroups.

This example adds three Percona XtraDB Cluster nodes to the default hostgroup (`0`), which receives both write and read traffic:

```
mysql@proxysql> INSERT INTO mysql_servers(hostgroup_id, hostname, port) VALUES (0,'192.168.70.61',330
mysql@proxysql> INSERT INTO mysql_servers(hostgroup_id, hostname, port) VALUES (0,'192.168.70.62',330
mysql@proxysql> INSERT INTO mysql_servers(hostgroup_id, hostname, port) VALUES (0,'192.168.70.63',330
```

To see the nodes:

```
mysql@proxysql> SELECT * FROM mysql_servers;

+--------------+---------------+------+--------+--------+-------------+-----------------+-----------
| hostgroup_id | hostname      | port | status | weight | compression | max_connections | max_replica
+--------------+---------------+------+--------+--------+-------------+-----------------+-----------
| 0            | 192.168.70.61 | 3306 | ONLINE | 1      | 0           | 1000            | 0
| 0            | 192.168.70.62 | 3306 | ONLINE | 1      | 0           | 1000            | 0
| 0            | 192.168.70.63 | 3306 | ONLINE | 1      | 0           | 1000            | 0
+--------------+---------------+------+--------+--------+-------------+-----------------+-----------
3 rows in set (0.00 sec)
```

Creating ProxySQL Monitoring User

To enable monitoring of Percona XtraDB Cluster nodes in ProxySQL, create a user with `USAGE` privilege on any node in the cluster and configure the user in ProxySQL.

The following example shows how to add a monitoring user on Node 2:

```
mysql@pxc2> CREATE USER 'proxysql'@'%' IDENTIFIED BY 'ProxySQLPa55';
mysql@pxc2> GRANT USAGE ON *.* TO 'proxysql'@'%';
```

The following example shows how to configure this user on the ProxySQL node:

```
mysql@proxysql> UPDATE global_variables SET variable_value='proxysql'
             WHERE variable_name='mysql-monitor_username';
mysql@proxysql> UPDATE global_variables SET variable_value='ProxySQLPa55'
             WHERE variable_name='mysql-monitor_password';
```

To load this configuration at runtime, issue a `LOAD` command. To save these changes to disk (ensuring that they persist after ProxySQL shuts down), issue a `SAVE` command.

```
mysql@proxysql> LOAD MYSQL VARIABLES TO RUNTIME;
mysql@proxysql> SAVE MYSQL VARIABLES TO DISK;
```

To ensure that monitoring is enabled, check the monitoring logs:

```
mysql@proxysql> SELECT * FROM monitor.mysql_server_connect_log ORDER BY time_start DESC LIMIT 6;
+---------------+------+------------------+----------------------+---------------+
| hostname      | port | time_start       | connect_success_time | connect_error |
+---------------+------+------------------+----------------------+---------------+
| 192.168.70.61 | 3306 | 1469635762434625 | 1695                 | NULL          |
| 192.168.70.62 | 3306 | 1469635762434625 | 1779                 | NULL          |
| 192.168.70.63 | 3306 | 1469635762434625 | 1627                 | NULL          |
| 192.168.70.61 | 3306 | 1469635642434517 | 1557                 | NULL          |
| 192.168.70.62 | 3306 | 1469635642434517 | 2737                 | NULL          |
| 192.168.70.63 | 3306 | 1469635642434517 | 1447                 | NULL          |
+---------------+------+------------------+----------------------+---------------+
6 rows in set (0.00 sec)

mysql> SELECT * FROM monitor.mysql_server_ping_log ORDER BY time_start DESC LIMIT 6;
+---------------+------+------------------+-------------------+------------+
| hostname      | port | time_start       | ping_success_time | ping_error |
+---------------+------+------------------+-------------------+------------+
| 192.168.70.61 | 3306 | 1469635762416190 | 948               | NULL       |
| 192.168.70.62 | 3306 | 1469635762416190 | 803               | NULL       |
| 192.168.70.63 | 3306 | 1469635762416190 | 711               | NULL       |
| 192.168.70.61 | 3306 | 1469635702416062 | 783               | NULL       |
```

```
| 192.168.70.62 | 3306 | 1469635702416062 | 631                | NULL        |
| 192.168.70.63 | 3306 | 1469635702416062 | 542                | NULL        |
+---------------+------+------------------+--------------------+-------------+
6 rows in set (0.00 sec)
```

The previous examples show that ProxySQL is able to connect and ping the nodes you added.

To enable monitoring of these nodes, load them at runtime:

```
mysql@proxysql> LOAD MYSQL SERVERS TO RUNTIME;
```

Creating ProxySQL Client User

ProxySQL must have users that can access backend nodes to manage connections.

To add a user, insert credentials into `mysql_users` table:

```
mysql@proxysql> INSERT INTO mysql_users (username,password) VALUES ('sbuser','sbpass');
Query OK, 1 row affected (0.00 sec)
```

Note: ProxySQL currently doesn't encrypt passwords.

Load the user into runtime space:

```
mysql@proxysql> LOAD MYSQL USERS TO RUNTIME;
```

To confirm that the user has been set up correctly, you can try to log in:

```
root@proxysql:~# mysql -u sbuser -p sbpass -h 127.0.0.1 -P 6033

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1491
Server version: 5.1.30 (ProxySQL)

Copyright (c) 2009-2016 Percona LLC and/or its affiliates
Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

To provide read/write access to the cluster for ProxySQL, add this user on one of the Percona XtraDB Cluster nodes:

```
mysql@pxc3> CREATE USER 'sbuser'@'192.168.70.64' IDENTIFIED BY 'sbpass';
Query OK, 0 rows affected (0.01 sec)

mysql@pxc3> GRANT ALL ON *.* TO 'sbuser'@'192.168.70.64';
Query OK, 0 rows affected (0.00 sec)
```

Adding Galera Support

Default ProxySQL cannot detect a node which is not in `Synced` state. To monitor status of Percona XtraDB Cluster nodes, use the `proxysql_galera_checker` script. The script is located here: `/usr/bin/proxysql_galera_checker`.

To use this script, load it into ProxySQL Scheduler.

The following example shows how you can load the script for default ProxySQL configuration:

```
mysql@proxysql> INSERT INTO scheduler(id,interval_ms,filename,arg1,arg2,arg3,arg4)
  VALUES
  (1,'10000','/usr/bin/proxysql_galera_checker','127.0.0.1','6032','0',
  '/var/lib/proxysql/proxysql_galera_checker.log');
```

To load the scheduler changes into the runtime space:

```
mysql@proxysql> LOAD SCHEDULER TO RUNTIME;
```

To make sure that the script has been loaded, check the `runtime_scheduler` table:

```
mysql@proxysql> SELECT * FROM runtime_scheduler\G
*************************** 1. row ***************************
         id: 1
interval_ms: 10000
   filename: /usr/bin/proxysql/proxysql_galera_checker
       arg1: 127.0.0.1
       arg2: 6032
       arg3: 0
       arg4: /var/lib/proxysql/proxysql_galera_checker.log
       arg5: NULL
 1 row in set (0.00 sec)
```

To check the status of available nodes, run the following command:

```
mysql@proxysql> SELECT hostgroup_id,hostname,port,status FROM mysql_servers;
+--------------+---------------+------+--------+
| hostgroup_id | hostname      | port | status |
+--------------+---------------+------+--------+
| 0            | 192.168.70.61 | 3306 | ONLINE |
| 0            | 192.168.70.62 | 3306 | ONLINE |
| 0            | 192.168.70.63 | 3306 | ONLINE |
+--------------+---------------+------+--------+
3 rows in set (0.00 sec)
```

Note: Each node can have the following status:

- `ONLINE`: backend node is fully operational.
- `SHUNNED`: backend node is temporarily taken out of use, because either too many connection errors hapenned in a short time, or replication lag exceeded the allowed threshold.
- `OFFLINE_SOFT`: new incoming connections aren't accepted, while existing connections are kept until they become inactive. In other words, connections are kept in use until the current transaction is completed. This allows to gracefully detach a backend node.
- `OFFLINE_HARD`: existing connections are dropped, and new incoming connections aren't accepted. This is equivalent to deleting the node from a hostgroup, or temporarily taking it out of the hostgroup for maintenance.

Testing Cluster with sysbench

You can install `sysbench` from Percona software repositories:

- For Debian or Ubuntu:

```
root@proxysql:~# apt-get install sysbench
```

- For Red Hat Enterprise Linux or CentOS

```
[root@proxysql ~]# yum install sysbench
```

Note: `sysbench` requires ProxySQL client user credentials that you creted in *Creating ProxySQL Client User*.

1. Create the database that will be used for testing on one of the Percona XtraDB Cluster nodes:

   ```
   mysql@pxc1> CREATE DATABASE sbtest;
   ```

2. Populate the table with data for the benchmark on the ProxySQL node:

   ```
   root@proxysql:~# sysbench --report-interval=5 --num-threads=4 \
     --num-requests=0 --max-time=20 \
     --test=/usr/share/doc/sysbench/tests/db/oltp.lua \
     --mysql-user='sbuser' --mysql-password='sbpass' \
     --oltp-table-size=10000 --mysql-host=127.0.0.1 --mysql-port=6033 \
     prepare
   ```

3. Run the benchmark on the ProxySQL node:

   ```
   root@proxysql:~# sysbench --report-interval=5 --num-threads=4 \
     --num-requests=0 --max-time=20 \
     --test=/usr/share/doc/sysbench/tests/db/oltp.lua \
     --mysql-user='sbuser' --mysql-password='sbpass' \
     --oltp-table-size=10000 --mysql-host=127.0.0.1 --mysql-port=6033 \
     run
   ```

ProxySQL stores collected data in the `stats` schema:

```
mysql@proxysql> SHOW TABLES FROM stats;
+---------------------------------+
| tables                          |
+---------------------------------+
| stats_mysql_query_rules         |
| stats_mysql_commands_counters   |
| stats_mysql_processlist         |
| stats_mysql_connection_pool     |
| stats_mysql_query_digest        |
| stats_mysql_query_digest_reset  |
| stats_mysql_global              |
+---------------------------------+
```

For example, to see the number of commands that run on the cluster:

```
mysql@proxysql> SELECT * FROM stats_mysql_commands_counters;
+------------------+---------------+-----------+-----------+-----------+----------+----------+--------
| Command          | Total_Time_us | Total_cnt | cnt_100us | cnt_500us | cnt_1ms  | cnt_5ms  | cnt_10r
+------------------+---------------+-----------+-----------+-----------+----------+----------+--------
| ALTER_TABLE      | 0             | 0         | 0         | 0         | 0        | 0        | 0
| ANALYZE_TABLE    | 0             | 0         | 0         | 0         | 0        | 0        | 0
| BEGIN            | 2212625       | 3686      | 55        | 2162      | 899      | 569      | 1
| CHANGE_MASTER    | 0             | 0         | 0         | 0         | 0        | 0        | 0
| COMMIT           | 21522591      | 3628      | 0         | 0         | 0        | 1765     | 1590
| CREATE_DATABASE  | 0             | 0         | 0         | 0         | 0        | 0        | 0
| CREATE_INDEX     | 0             | 0         | 0         | 0         | 0        | 0        | 0
...
```

```
| DELETE             | 2904130       | 3670      | 35        | 1546      | 1346     | 723      | 19
| DESCRIBE           | 0             | 0         | 0         | 0         | 0        | 0        | 0
...
| INSERT             | 19531649      | 3660      | 39        | 1588      | 1292     | 723      | 12
...
| SELECT             | 35049794      | 51605     | 501       | 26180     | 16606    | 8241     | 70
| SELECT_FOR_UPDATE  | 0             | 0         | 0         | 0         | 0        | 0        | 0
...
| UPDATE             | 6402302       | 7367      | 75        | 2503      | 3020     | 1743     | 23
| USE                | 0             | 0         | 0         | 0         | 0        | 0        | 0
| SHOW               | 19691         | 2         | 0         | 0         | 0        | 0        | 1
| UNKNOWN            | 0             | 0         | 0         | 0         | 0        | 0        | 0
+--------------------+---------------+-----------+-----------+-----------+----------+----------+--------
45 rows in set (0.00 sec)
```

Automatic Fail-over

ProxySQL will automatically detect if a node is not available or not synced with the cluster.

You can check the status of all available nodes by running:

```
mysql@proxysql> SELECT hostgroup_id,hostname,port,status FROM mysql_servers;
+--------------+---------------+------+--------+
| hostgroup_id | hostname      | port | status |
+--------------+---------------+------+--------+
| 0            | 192.168.70.61 | 3306 | ONLINE |
| 0            | 192.168.70.62 | 3306 | ONLINE |
| 0            | 192.168.70.63 | 3306 | ONLINE |
+--------------+---------------+------+--------+
3 rows in set (0.00 sec)
```

To test problem detection and fail-over mechanism, shut down Node 3:

```
root@pxc3:~# service mysql stop
```

ProxySQL will detect that the node is down and update its status to `OFFLINE_SOFT`:

```
mysql@proxysql> SELECT hostgroup_id,hostname,port,status FROM mysql_servers;
+--------------+---------------+------+--------------+
| hostgroup_id | hostname      | port | status       |
+--------------+---------------+------+--------------+
| 0            | 192.168.70.61 | 3306 | ONLINE       |
| 0            | 192.168.70.62 | 3306 | ONLINE       |
| 0            | 192.168.70.63 | 3306 | OFFLINE_SOFT |
+--------------+---------------+------+--------------+
3 rows in set (0.00 sec)
```

Now start Node 3 again:

```
root@pxc3:~# service mysql start
```

The script will detect the change and mark the node as `ONLINE`:

```
mysql@proxysql> SELECT hostgroup_id,hostname,port,status FROM mysql_servers;
+--------------+---------------+------+--------+
| hostgroup_id | hostname      | port | status |
+--------------+---------------+------+--------+
| 0            | 192.168.70.61 | 3306 | ONLINE |
| 0            | 192.168.70.62 | 3306 | ONLINE |
```

```
| 0            | 192.168.70.63 | 3306 | ONLINE |
+--------------+---------------+------+--------+
3 rows in set (0.00 sec)
```

5.9 Setting up PXC reference architecture with HAProxy

This manual describes how to set up Percona XtraDB Cluster in a virtualized test sandbox.

The procedure assumes Amazon EC2 micro instances running CentOS 6. However, it should apply to any virtualization technology (for example, VirtualBox) with any Linux distribution.

This manual requires three virtual machines for Percona XtraDB Cluster nodes, and one for HAProxy client, which redirects requests to the nodes. Running HAProxy on an application server, instead of having it as a dedicated entity, removes the unnecessary extra network roundtrip, because the load balancing layer in Percona XtraDB Cluster scales well with application servers.

1. Install Percona XtraDB Cluster on the three cluster nodes, as described in *Installing Percona XtraDB Cluster on Red Hat Enterprise Linux and CentOS*.
2. Install HAProxy and `sysbench` on the client node:

   ```
   yum -y install haproxy sysbench
   ```

3. Make sure that the `my.cnf` configuration file on the first node contains the following:

   ```
   [mysqld]
   server_id=1
   binlog_format=ROW
   log_bin=mysql-bin
   wsrep_cluster_address=gcomm://
   wsrep_provider=/usr/lib/libgalera_smm.so
   datadir=/var/lib/mysql

   wsrep_slave_threads=2
   wsrep_cluster_name=pxctest
   wsrep_sst_method=xtrabackup
   wsrep_node_name=ip-10-112-39-98

   log_slave_updates

   innodb_autoinc_lock_mode=2
   innodb_buffer_pool_size=400M
   innodb_log_file_size=64M
   ```

4. Start the first node
5. Adjust the `my.cnf` configuration files on the second and third nodes to contain the same configuration settings, except the following:

   - Second node:

     ```
     server_id=2
     wsrep_cluster_address=gcomm://10.116.39.76
     wsrep_node_name=ip-10-244-33-92
     ```

   - Third node:

```
server_id=3
wsrep_cluster_address=gcomm://10.116.39.76
wsrep_node_name=ip-10-194-10-179
```

Note:

- `server_id` can be any unique number
- `wsrep_cluster_address` is the IP address of the first node
- `wsrep_node_name` can be any unique name, for example, the output of the `hostname` command

6. Start the second and third nodes.

   When a new node joins the cluster, *SST* is performed by taking a backup using XtraBackup, then copying it to the new node with `netcat`. After a successful *SST*, you should see the following in the error log:

```
120619 13:20:17 [Note] WSREP: State transfer required:
     Group state: 77c9da88-b965-11e1-0800-ea53b7b12451:97
     Local state: 00000000-0000-0000-0000-000000000000:-1
120619 13:20:17 [Note] WSREP: New cluster view: global state: 77c9da88-b965-11e1-0800-ea53b7b124
120619 13:20:17 [Warning] WSREP: Gap in state sequence. Need state transfer.
120619 13:20:19 [Note] WSREP: Running: 'wsrep_sst_xtrabackup 'joiner' '10.195.206.117' '' '/var/
120619 13:20:19 [Note] WSREP: Prepared |SST| request: xtrabackup|10.195.206.117:4444/xtrabackup_
120619 13:20:19 [Note] WSREP: wsrep_notify_cmd is not defined, skipping notification.
120619 13:20:19 [Note] WSREP: Assign initial position for certification: 97, protocol version: 2
120619 13:20:19 [Warning] WSREP: Failed to prepare for incremental state transfer: Local state U
      at galera/src/replicator_str.cpp:prepare_for_IST():439. IST will be unavailable.
120619 13:20:19 [Note] WSREP: Node 0 (ip-10-244-33-92) requested state transfer from '*any*'. Se
120619 13:20:19 [Note] WSREP: Shifting PRIMARY -> JOINER (TO: 102)
120619 13:20:19 [Note] WSREP: Requesting state transfer: success, donor: 1
120619 13:20:59 [Note] WSREP: 1 (ip-10-112-39-98): State transfer to 0 (ip-10-244-33-92) complet
120619 13:20:59 [Note] WSREP: Member 1 (ip-10-112-39-98) synced with group.
120619 13:21:17 [Note] WSREP: |SST| complete, seqno: 105
120619 13:21:17 [Note] Plugin 'FEDERATED' is disabled.
120619 13:21:17 InnoDB: The InnoDB memory heap is disabled
120619 13:21:17 InnoDB: Mutexes and rw_locks use GCC atomic builtins
120619 13:21:17 InnoDB: Compressed tables use zlib 1.2.3
120619 13:21:17 InnoDB: Using Linux native AIO
120619 13:21:17 InnoDB: Initializing buffer pool, size = 400.0M
120619 13:21:17 InnoDB: Completed initialization of buffer pool
120619 13:21:18 InnoDB: highest supported file format is Barracuda.
120619 13:21:18  InnoDB: Waiting for the background threads to start
120619 13:21:19 Percona XtraDB (http://www.percona.com) 1.1.8-rel25.3 started; log sequence numb
120619 13:21:19 [Note] Recovering after a crash using mysql-bin
120619 13:21:19 [Note] Starting crash recovery...
120619 13:21:19 [Note] Crash recovery finished.
120619 13:21:19 [Note] Server hostname (bind-address): '(null)'; port: 3306
120619 13:21:19 [Note]   - '(null)' resolves to '0.0.0.0';
120619 13:21:19 [Note]   - '(null)' resolves to '::';
120619 13:21:19 [Note] Server socket created on IP: '0.0.0.0'.
120619 13:21:19 [Note] Event Scheduler: Loaded 0 events
120619 13:21:19 [Note] WSREP: Signalling provider to continue.
120619 13:21:19 [Note] WSREP: Received |SST|: 77c9da88-b965-11e1-0800-ea53b7b12451:105
120619 13:21:19 [Note] WSREP: |SST| received: 77c9da88-b965-11e1-0800-ea53b7b12451:105
120619 13:21:19 [Note] WSREP: 0 (ip-10-244-33-92): State transfer from 1 (ip-10-112-39-98) compl
120619 13:21:19 [Note] WSREP: Shifting JOINER -> JOINED (TO: 105)
120619 13:21:19 [Note] /usr/sbin/mysqld: ready for connections.
Version: '5.5.24-log'  socket: '/var/lib/mysql/mysql.sock'  port: 3306  Percona XtraDB Cluster (
```

```
120619 13:21:19 [Note] WSREP: Member 0 (ip-10-244-33-92) synced with group.
120619 13:21:19 [Note] WSREP: Shifting JOINED -> SYNCED (TO: 105)
120619 13:21:20 [Note] WSREP: Synchronized with group, ready for connections
```

For debugging information about the *SST*, you can check the `sst.err` file and the error log.

After *SST* finishes, you can check the cluster size as follows:

```
mysql> show global status like 'wsrep_cluster_size';
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| wsrep_cluster_size | 3     |
+--------------------+-------+
1 row in set (0.00 sec)
```

7. When all cluster nodes are started, configure HAProxy on the client node. This will enable the application to connect to localhost as if it were a single MySQL server, instead of a Percona XtraDB Cluster node.

   You can configure HAProxy to connect and write to all cluster nodes or to one node at a time. The former method can lead to rollbacks due to conflicting writes when optimistic locking at commit time is triggered, while the latter method avoids rollbacks.

   However, most good applications should be able to handle rollbacks, so either method is fine in this case.

   To configure HAProxy, add the following to `/etc/haproxy/haproxy.cfg`:

```
global
log 127.0.0.1 local0
log 127.0.0.1 local1 notice
maxconn 4096
chroot /usr/share/haproxy
user haproxy
group haproxy
daemon

defaults
log global
mode http
option tcplog
option dontlognull
retries 3
option redispatch
maxconn 2000
contimeout 5000
clitimeout 50000
srvtimeout 50000

frontend pxc-front
bind *:3307
mode tcp
default_backend pxc-back

frontend stats-front
bind *:80
mode http
default_backend stats-back

frontend pxc-onenode-front
bind *:3306
```

```
mode tcp
default_backend pxc-onenode-back

backend pxc-back
mode tcp
balance leastconn
option httpchk
server c1 10.116.39.76:3306 check port 9200 inter 12000 rise 3 fall 3
server c2 10.195.206.117:3306 check port 9200 inter 12000 rise 3 fall 3
server c3 10.202.23.92:3306 check port 9200 inter 12000 rise 3 fall 3

backend stats-back
mode http
balance roundrobin
stats uri /haproxy/stats
stats auth pxcstats:secret

backend pxc-onenode-back
mode tcp
balance leastconn
option httpchk
server c1 10.116.39.76:3306 check port 9200 inter 12000 rise 3 fall 3
server c2 10.195.206.117:3306 check port 9200 inter 12000 rise 3 fall 3 backup
server c3 10.202.23.92:3306 check port 9200 inter 12000 rise 3 fall 3 backup
```

In this configuration, three frontend-backend pairs are defined:

- The `stats` pair is for HAProxy statistics page (port 80).

  You can access it at `/haproxy/stats` using the credential specified in the `stats auth` parameter.

- The `pxc` pair is for connecting to all three nodes (port 3307).

  In this case, the *leastconn* load balancing method is used, instead of round-robin, which means connection is made to the backend with the least connections established.

- The `pxc-onenode` pair is for connecting to one node at a time (port 3306) to avoid rollbacks because of optimistic locking.

  If the node goes offline, HAProxy will connect to another one.

Note: MySQL is checked via `httpchk`. MySQL will not serve these requests by default. You have to set up the `clustercheck` utility, which is distributed with Percona XtraDB Cluster. This will enable HAProxy to check MySQL via HTTP.

The `clustercheck` script is a simple shell script that accepts HTTP requests and checks the node via the `wsrep_local_state` variable. If the node's status is fine, it will send a response with HTTP code `200 OK`. Otherwise, it sends `503`.

To create the `clustercheck` user, run the following:

```
mysql> grant process on *.* to 'clustercheckuser'@'localhost' identified by 'clustercheckpasswor
Query OK, 0 rows affected (0.00 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)
```

If you want to use a different user name or password, you have to modify them in the `clustercheck` script.

If you run the script on a running node, you should see the following:

```
# clustercheck
HTTP/1.1 200 OK

Content-Type: Content-Type: text/plain
```

You can use `xinetd` to daemonize the script. If *xinetd* is not installed, you can install it with `yum`:

```
# yum -y install xinetd
```

The service is configured in `/etc/xinetd.d/mysqlchk`:

```
# default: on
# description: mysqlchk
service mysqlchk
{
# this is a config for xinetd, place it in /etc/xinetd.d/
disable = no
flags = REUSE
socket_type = stream
port = 9200
wait = no
user = nobody
server = /usr/bin/clustercheck
log_on_failure += USERID
only_from = 0.0.0.0/0
# recommended to put the IPs that need
# to connect exclusively (security purposes)
per_source = UNLIMITED
}
```

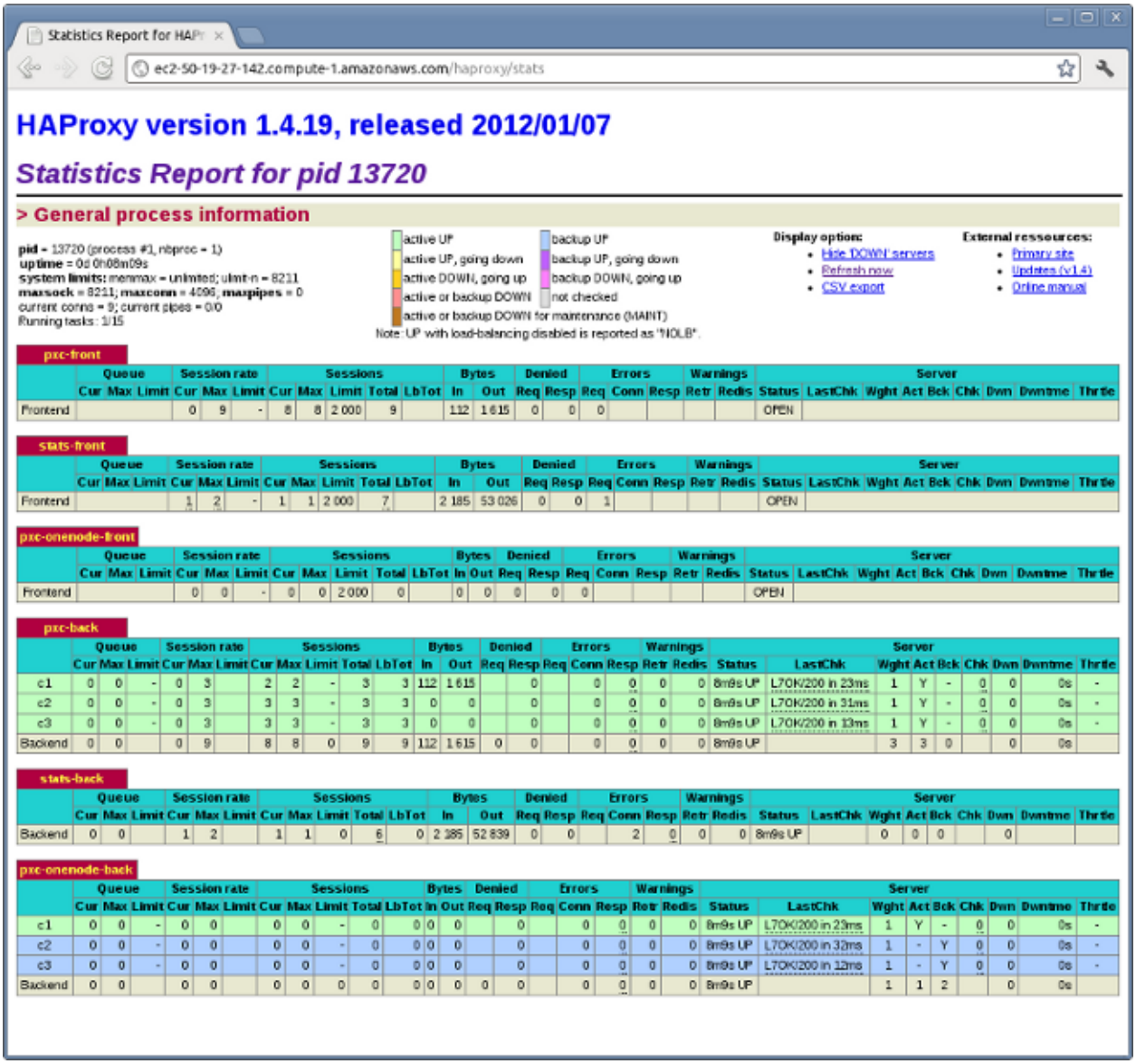
Add the new service to `/etc/services`:

```
mysqlchk 9200/tcp # mysqlchk
```

Clustercheck will now listen on port 9200 after `xinetd` restarts and HAProxy is ready to check MySQL via HTTP:

```
# service xinetd restart
```

If you did everything correctly, the statistics page for HAProxy should look like this:



5.9.1 Testing the cluster with sysbench

After you set up Percona XtraDB Cluster in a sand box, you can test it using sysbench. This example shows how to do it with `sysbench` from the EPEL repository.

1. Create a database and a user for `sysbench`:

```
mysql> create database sbtest;
Query OK, 1 row affected (0.01 sec)

mysql> grant all on sbtest.* to 'sbtest'@'%' identified by 'sbpass';
Query OK, 0 rows affected (0.00 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)
```

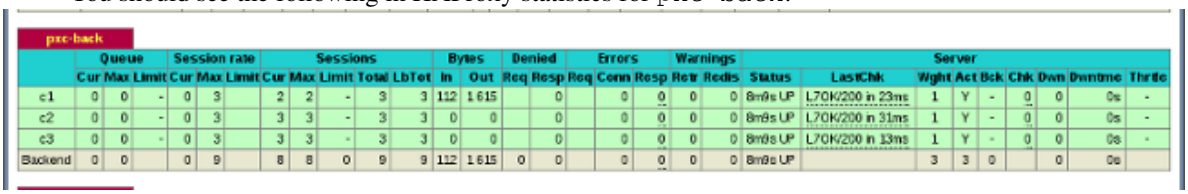
2. Populate the table with data for the benchmark:

```
sysbench --test=oltp --db-driver=mysql --mysql-engine-trx=yes --mysql-table-engine=innodb --mysq
```

3. Run the benchmark on port 3307:

   ```
   sysbench --test=oltp --db-driver=mysql --mysql-engine-trx=yes --mysql-table-engine=innodb --mysq
   ```

   You should see the following in HAProxy statistics for `pxc-back`:



pxc-back

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
c1	0	0	-	0	3		2	2	-	3	3	112	1 615		0		0	0	0	0	8m9s UP	L7OK/200 in 23ms	1	Y	-	0	0	0s	-
c2	0	0	-	0	3		3	3	-	3	3	0	0		0		0	0	0	0	8m9s UP	L7OK/200 in 31ms	1	Y	-	0	0	0s	-
c3	0	0	-	0	3		3	3	-	3	3	0	0		0		0	0	0	0	8m9s UP	L7OK/200 in 13ms	1	Y	-	0	0	0s	-
Backend	0	0		0	9		8	8	0	9	9	112	1 615	0	0		0	0	0	0	8m9s UP		3	3	0		0	0s	

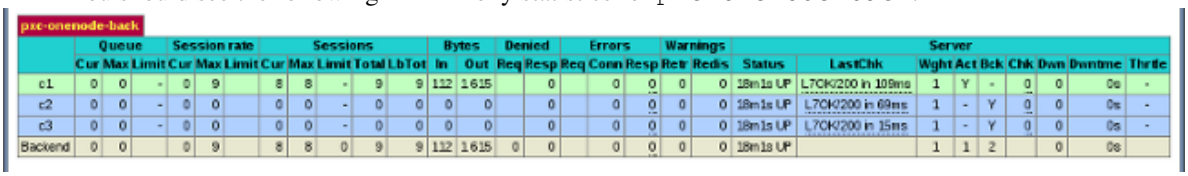
   Note the `Cur` column under `Session`:

   - `c1` has 2 threads connected
   - `c2` and `c3` have 3 threads connected

4. Run the same benchmark on port 3306:

   ```
   sysbench --test=oltp --db-driver=mysql --mysql-engine-trx=yes --mysql-table-engine=innodb --mysq
   ```

   You should see the following in HAProxy statistics for `pxc-onenode-back`:



pxc-onenode-back

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
c1	0	0	-	0	9		8	8	-	9	9	112	1 615		0		0	0	0	0	18m1s UP	L7OK/200 in 109ms	1	Y	-	0	0	0s	-
c2	0	0	-	0	0		0	0	-	0	0	0	0		0		0	0	0	0	18m1s UP	L7OK/200 in 69ms	1	-	Y	0	0	0s	-
c3	0	0	-	0	0		0	0	-	0	0	0	0		0		0	0	0	0	18m1s UP	L7OK/200 in 15ms	1	-	Y	0	0	0s	-
Backend	0	0		0	9		8	8	0	9	9	112	1 615	0	0		0	0	0	0	18m1s UP		1	1	2		0	0s	

   All 8 threads are connected to the `c1` server. `c2` and `c3` are acting as backup nodes.

If you are using *HAProxy* for *MySQL* you can break the privilege system's host part, because *MySQL* will think that the connections are always coming from the load balancer. You can work this around using T-Proxy patches and some *iptables* magic for the backwards connections. However in the setup described in this how-to this is not an issue, since each application server has it's own *HAProxy* instance, each application server connects to 127.0.0.1, so MySQL will see that connections are coming from the application servers. Just like in the normal case.

CHAPTER

SIX

REFERENCE

6.1 *Percona XtraDB Cluster* 5.7 Release notes

6.1.1 *Percona XtraDB Cluster* 5.7.14-26.17

Note: This release is dedicated to the memory of Federico Goncalvez, our colleague with Percona's Uruguayan team until his tragic death on September 6, 2016.

Fede, you are missed.

Percona is glad to announce the release of Percona XtraDB Cluster 5.7.14-26.17 on September 29, 2016. Binaries are available from the downloads area or from our *software repositories*.

Percona XtraDB Cluster 5.7.14-26.17 is the first GA release in the 5.7 series, based on the following:

- Percona Server 5.7.14-8
- Galera Replicator 3.17

For information about the changes and new features introduced in Percona Server 5.7, see Changed in Percona Server 5.7.

New Features

This is a list of the most important features introduced in Percona XtraDB Cluster 5.7 compared to version 5.6

- *PXC Strict Mode* saves your workload from experimental and unsupported features.
- Support for monitoring Galera Library instruments and other wsrep instruments as part of Performance Schema.
- Support for encrypted tablespaces in Multi-Master Topology, which enables Percona XtraDB Cluster to wire encrypted tablespace to new booting node.
- Compatibility with ProxySQL, including a quick configuration script.
- Support for monitoring Percona XtraDB Cluster nodes using Percona Monitoring and Management
- More stable and robust operation with MySQL and Percona Server version 5.7.14, as well as Galera 3.17 compatibility. Includes all upstream bug fixes, improved logging and more.
- Simplified packaging for Percona XtraDB Cluster to a single package that installs everything it needs, including the Galera library.
- Support for latest XtraBackup with enhanced security checks.

Bug Fixes

Fixed crash when a local transaction (such as `EXPLAIN` or `SHOW`) is interrupted by a replicated transaction with higher priopiry (like `ALTER` that changes table structure and can thus affect the result of the local transaction).

Fixed DONOR node getting stuck in `Joined` state after successful SST.

Fixed error message when altering non-existent table with `pxc-strict-mode` enabled.

Fixed path to directory in `percona-xtradb-cluster-shared.conf`.

Fixed setting of `seqno` in `grastate.dat` to `-1` on clean shutdown.

Fixed failure of asynchronous TOI actions (like `DROP`) for non-primary nodes.

Fixed replacing of `my.cnf` during upgrade from 5.6 to 5.7.

Security Fixes

- CVE-2016-6662
- CVE-2016-6663
- CVE-2016-6664

For more information, see https://www.percona.com/blog/2016/09/12/database-affected-cve-2016-6662/

Other Improvements

Added support of `defaults-group-suffix` for SST scripts.

6.1.2 *Percona XtraDB Cluster* 5.7.12-5rc1-26.16

Percona is glad to announce the release of Percona XtraDB Cluster 5.7.12-5rc1-26.16 on August 9, 2016. Binaries are available from the downloads area or from our *software repositories*.

Percona XtraDB Cluster 5.7.12-5rc1-26.16 is based on the following:

- Percona Server 5.7.12
- Galera Replicator 3.16

New Features

- **PXC Strict Mode**: Use the `pxc_strict_mode` variable in the configuration file or the `--pxc-strict-mode` option during `mysqld` startup. For more information, see *PXC Strict Mode*.
- **Galera instruments exposed in Performance Schema**: This includes mutexes, condition variables, file instances, and threads.

Bug Fixes

- Fixed error messages.
- Fixed the failure of SST via `mysqldump` with `gtid_mode=ON`.
- Added check for TOI that ensures node readiness to process DDL+DML before starting the execution.

- Removed protection against repeated calls of `wsrep->pause()` on the same node to allow parallel RSU operation.
- Changed `wsrep_row_upd_check_foreign_constraints` to ensure that `fk-reference-table` is open before marking it open.
- Fixed error when running `SHOW STATUS` during group state update.
- Corrected the return code of `sst_flush_tables()` function to return a non-negative error code and thus pass assertion.
- Fixed memory leak and stale pointer due to stats not freeing when toggling the `wsrep_provider` variable.
- Fixed failure of `ROLLBACK` to register `wsrep_handler`
- Fixed failure of symmetric encryption during SST.

Other Changes

- Added support for sending the keyring when performing encrypted SST. For more information, see *Encrypting PXC Traffic*.
- Changed the code of `THD_PROC_INFO` to reflect what the thread is currently doing.
- Using XtraBackup as the SST method now requires Percona XtraBackup 2.4.4 or later.
- Improved rollback process to ensure that when a transaction is rolled back, any statements open by the transaction are also rolled back.
- Removed the `sst_special_dirs` variable.
- Disabled switching of `slave_preserve_commit_order` to `ON` when running PXC in cluster mode, as it conflicts with existing multi-master commit ordering resolution algorithm in Galera.
- Changed the default `my.cnf` configuration.
- Other low-level fixes and improvements for better stability.

6.1.3 *Percona XtraDB Cluster* 5.7.12-25.16

Percona is glad to announce the release of Percona XtraDB Cluster 5.7.12-25.16 on June 30, 2016. Binaries are available from downloads area or from our *software repositories*.

Percona XtraDB Cluster 5.7.12-25.16 is based on the following:

- Percona Server 5.7.12
- Galera Replicator 3.16

6.1.4 *Percona XtraDB Cluster* 5.7.11-4beta-25.14.2

Percona is glad to announce the release of Percona XtraDB Cluster 5.7.11-4beta-25.14.2 on June 9, 2016. Binaries are available from downloads area or from our *software repositories*.

Note: This release is available only from the testing repository. It is not meant for upgrade from Percona XtraDB Cluster 5.6 and earlier versions. Only fresh installation is supported.

Percona XtraDB Cluster 5.7.11-4beta-25.14.2 is based on the following:

- Percona Server 5.7.11-4

- Galera Replicator 3.14.2

This is the first beta release in the Percona XtraDB Cluster 5.7 series. It includes all changes from upstream releases and the following changes:

- Percona XtraDB Cluster 5.7 does not include `wsrep_sst_xtrabackup`. It has been replace by `wsrep_sst_xtrabackup_v2`.
- The `wsrep_mysql_replication_bundle` variable has been removed.

6.2 Index of wsrep status variables

variable `wsrep_apply_oooe`

This variable shows parallelization efficiency, how often writests have been applied out of order.

variable `wsrep_apply_oool`

This variable shows how often a writeset with a higher sequence number was applied before one with a lower sequence number.

variable `wsrep_apply_window`

Average distance between highest and lowest concurrently applied sequence numbers.

variable `wsrep_causal_reads_`

Shows the number of writesets processed while the variable `wsrep_causal_reads` was set to `ON`.

variable `wsrep_cert_bucket_count`

This variable, shows the number of cells in the certification index hash-table.

variable `wsrep_cert_deps_distance`

Average distance between highest and lowest sequence number that can be possibly applied in parallel.

variable `wsrep_cert_index_size`

Number of entries in the certification index.

variable `wsrep_cert_interval`

Average number of write-sets received while a transaction replicates.

variable `wsrep_cluster_conf_id`

Number of cluster membership changes that have taken place.

variable `wsrep_cluster_size`

Current number of nodes in the cluster.

variable `wsrep_cluster_state_uuid`

This variable contains *UUID* state of the cluster. When this value is the same as the one in `wsrep_local_state_uuid`, node is synced with the cluster.

variable `wsrep_cluster_status`

Status of the cluster component. Possible values are:

- `Primary`
- `Non-Primary`
- `Disconnected`

variable wsrep_commit_oooe

This variable shows how often a transaction was committed out of order.

variable wsrep_commit_oool

This variable currently has no meaning.

variable wsrep_commit_window

Average distance between highest and lowest concurrently committed sequence number.

variable wsrep_connected

This variable shows if the node is connected to the cluster. If the value is `OFF`, the node has not yet connected to any of the cluster components. This may be due to misconfiguration.

variable wsrep_evs_delayed

Comma separated list of nodes that are considered delayed. The node format is `<uuid>:<address>:<count>`, where `<count>` is the number of entries on delayed list for that node.

variable wsrep_evs_evict_list

List of UUIDs of the evicted nodes.

variable wsrep_evs_repl_latency

This status variable provides information regarding group communication replication latency. This latency is measured in seconds from when a message is sent out to when a message is received.

The format of the output is `<min>/<avg>/<max>/<std_dev>/<sample_size>`.

variable wsrep_evs_state

Internal EVS protocol state.

variable wsrep_flow_control_paused

Time since the last status query that was paused due to flow control.

variable wsrep_flow_control_paused_ns

Total time spent in a paused state measured in nanoseconds.

variable wsrep_flow_control_recv

Number of `FC_PAUSE` events received since the last status query.

variable wsrep_flow_control_sent

Number of `FC_PAUSE` events sent since the last status query.

variable wsrep_gcache_pool_size

This variable shows the size of the page pool and dynamic memory allocated for GCache (in bytes).

variable wsrep_gcomm_uuid

This status variable exposes UUIDs in `gvwstate.dat`, which are Galera view IDs (thus unrelated to cluster state UUIDs). This UUID is unique for each node. You will need to know this value when using manual eviction feature.

variable wsrep_incoming_addresses

Shows the comma-separated list of incoming node addresses in the cluster.

variable wsrep_last_committed

Sequence number of the last committed transaction.

variable wsrep_local_bf_aborts

Number of local transactions that were aborted by slave transactions while being executed.

variable wsrep_local_cached_downto

The lowest sequence number in GCache. This information can be helpful with determining IST and SST. If the value is `0`, then it means there are no writesets in GCache (usual for a single node).

variable wsrep_local_cert_failures

Number of writesets that failed the certification test.

variable wsrep_local_commits

Number of writesets commited on the node.

variable wsrep_local_index

Node's index in the cluster.

variable wsrep_local_recv_queue

Current length of the receive queue (that is, the number of writesets waiting to be applied).

variable wsrep_local_recv_queue_avg

Average length of the receive queue since the last status query. When this number is bigger than `0` this means node can't apply writesets as fast as they are received. This could be a sign that the node is overloaded and it may cause replication throttling.

variable wsrep_local_replays

Number of transaction replays due to *asymmetric lock granularity*.

variable wsrep_local_send_queue

Current length of the send queue (that is, the number of writesets waiting to be sent).

variable wsrep_local_send_queue_avg

Average length of the send queue since the last status query. When cluster experiences network throughput issues or replication throttling, this value will be significantly bigger than `0`.

variable wsrep_local_state

variable wsrep_local_state_comment

Internal number and the corresponding human-readable comment of the node's state. Possible values are:

Num	Comment	Description
1	Joining	Node is joining the cluster
2	Donor/Desynced	Node is the donor to the node joining the cluster
3	Joined	Node has joined the cluster
4	Synced	Node is synced with the cluster

variable wsrep_local_state_uuid

The *UUID* of the state stored on the node.

variable wsrep_protocol_version

Version of the wsrep protocol used.

variable wsrep_provider_name

Name of the wsrep provider (usually `Galera`).

variable wsrep_provider_vendor

Name of the wsrep provider vendor (usually `Codership Oy`)

variable wsrep_provider_version

Current version of the wsrep provider.

variable wsrep_ready

This variable shows if node is ready to accept queries. If status is `OFF`, almost all queries will fail with `ERROR 1047 (08S01) Unknown Command` error (unless the `wsrep_on` variable is set to `0`).

variable wsrep_received

Total number of writesets received from other nodes.

variable wsrep_received_bytes

Total size (in bytes) of writesets received from other nodes.

variable wsrep_repl_data_bytes

Total size (in bytes) of data replicated.

variable wsrep_repl_keys

Total number of keys replicated.

variable wsrep_repl_keys_bytes

Total size (in bytes) of keys replicated.

variable wsrep_repl_other_bytes

Total size of other bits replicated.

variable wsrep_replicated

Total number of writesets sent to other nodes.

variable wsrep_replicated_bytes

Total size (in bytes) of writesets sent to other nodes.

6.3 Index of wsrep system variables

variable wsrep_auto_increment_control

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value ON

This variable manages the `auto_increment_increment` and `auto_increment_offset` variables automatically depending on the size of the cluster. This helps prevent `auto_increment` replication conflicts across the cluster by giving each node it's own range of `auto_increment` values. This may not be desirable depending on application's use and assumptions of auto-increments. It can be turned off in Master/Slave clusters.

variable wsrep_causal_reads

Command Line Yes

Config File Yes

Scope Global, Session

Dynamic Yes

Default Value OFF

In some cases, master may apply event faster than a slave, which can cause master and slave to become out of sync for a brief moment. When this variable is set to `ON`, slave will wait until that event is applied before doing any other queries. Enabling this variable will also result in larger latencies.

variable `wsrep_certify_nonPK`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value ON

When this variable is enabled, primary keys will be generated automatically for rows that don't have them. Using tables without primary keys is not recommended.

variable `wsrep_cluster_address`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

This variable needs to specify at least one other node's address that is alive and a member of the cluster. In practice, it is best (but not necessary) to provide a complete list of all possible cluster nodes. The value should be of the following format:

```
gcomm://<node1_ip>,<node2_ip>,<node3_ip>
```

Besides the IP address of the node, you can also specify port and options, for example:

```
gcomm://192.168.0.1:4567?gmcast.listen_addr=0.0.0.0:5678
```

If an empty `gcomm:/ /` is provided, the node will bootstrap itself (that is, form a new cluster). This is not recommended for production after the cluster has been bootstrapped initially. If you want to bootstrap a new cluster, you should pass the `--wsrep-new-cluster` option when starting.

variable `wsrep_cluster_name`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value my_wsrep_cluster

This is the name of the cluster and should be identical on all nodes belonging to the same cluster.

Note: It should not exceed 32 characters.

---

variable `wsrep_convert_lock_to_trx`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value OFF

This variable is used to convert `LOCK/UNLOCK TABLES` statements to `BEGIN/COMMIT`. Although this can help some older applications to work with multi-master setup it can also result in having huge writesets.

variable `wsrep_data_home_dir`

Command Line No

Config File Yes

Scope Global

Dynamic No

Default Value mysql *datadir*

This variable can be used to set up the directory where wsrep provider will store its files (like `grastate.dat`).

variable `wsrep_dbug_option`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

This variable is used to send the `DBUG` option to the wsrep provider.

variable `wsrep_debug`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value OFF

When this variable is set to `ON`, debug messages will also be logged to the `error_log`. This can be used when trying to diagnose problems or when submitting a bug.

variable `wsrep_desync`

Command Line No

Config File Yes

Scope Global

Dynamic Yes

Default Value OFF

This variable controls whether the node participates in Flow Control. Setting the `wsrep_desync` to `ON` does not automatically mean that a node will be out of sync with the cluster. It will continue to replicate in and out the writesets as usual. The only difference is that flow control will no longer take care of the `desynced` node. The result is that if `wsrep_local_recv_queue` gets higher than maximum allowed, all the other nodes will ignore the replication

lag on the node being in `desync` mode. Toggling this back will require an IST or an SST depending on how long it was desynchronized. This is similar to cluster de-synchronization, which occurs during RSU TOI. Because of this, it's not a good idea to keep desync set for a long period of time, nor should you desync several nodes at once. Also, you'll need to desync a node before it starts causing flow control for it to have any effect. Node can also be desynchronized with `/*! WSREP_DESYNC */` query comment.

variable `wsrep_dirty_reads`

Command Line Yes

Config File Yes

Scope Session, Global

Dynamic Yes

Default Value OFF

This variable is boolean and is `OFF` by default. When set to `ON`, a *Percona XtraDB Cluster* node accepts queries that only read, but not modify data even if the node is in the non-PRIM state.

variable `wsrep_drupal_282555_workaround`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value OFF

This variable was introduced as a workaround for Drupal/MySQL bug #282555. In some cases, duplicate key errors would occur when inserting the `default` value into the `auto_increment` field.

variable `wsrep_forced_binlog_format`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value NONE

This variable defines a binlog format that will always be effective regardless of session binlog format setting. Possible values for this variable are:

- `ROW`
- `STATEMENT`
- `MIXED`
- `NONE`: Resets the forced state of the binlog format (default)

variable `wsrep_load_data_splitting`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value ON

This variable controls whether `LOAD DATA` transaction splitting is wanted or not. It doesn't work as expected with `autocommit=0` when enabled.

variable `wsrep_log_conflicts`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value OFF

This variable is used to control whether sole cluster conflicts should be logged. When enabled, details of conflicting *InnoDB* lock will be logged.

variable `wsrep_max_ws_rows`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value 131072 (128K)

This variable has no effect!

By design, it was supposed to control the maximum number of rows each writeset can contain. However, it is hard to predict the number of rows because of the writeset size limit enforced by `wsrep_max_ws_size`.

Codership decided to not implement the limit by rows for now. Correct behavior may be implemented in a future release. There is a discussion open at https://github.com/codership/mysql-wsrep/issues/257

variable `wsrep_max_ws_size`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value 1073741824 (1G)

This variable is used to control maximum writeset size (in bytes). Anything bigger than the specified value will be rejected.

variable `wsrep_mysql_replication_bundle`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 0 (no grouping)

Range 0-1000

This variable controls how many replication events will be grouped together. Replication events are grouped in SQL slave thread by skipping events which may cause commit. This way the wsrep node acting in *MySQL* slave role and all other wsrep nodes in provider replication group, will see same (huge) transactions. The implementation is still experimental. This may help with the bottleneck of having only one *MySQL* slave facing commit time delay of synchronous provider.

variable `wsrep_node_address`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Format <ip address>[:port]

Default Value Usually set up as primary network interface (`eth0`)

This variable is used to specify the network address of the node. In some cases, when there are multiple NICs available, state transfer might not work if the default NIC is on different network. Setting this variable explicitly to the correct value will make SST and IST work correctly out of the box. Even in multi-network setups, IST/SST can be configured to use other interfaces/addresses.

variable `wsrep_node_incoming_address`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value <`wsrep_node_address`>:3306

This is the address at which the node accepts client connections. This information is used for status variable `wsrep_incoming_addresses` which shows all the active cluster nodes.

variable `wsrep_node_name`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value Host name

Unique name of the node. Defaults to the host name.

variable `wsrep_notify_cmd`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

This variable is used to set the notification command that the server should execute every time cluster membership or local node status changes.

variable `wsrep_on`

Command Line No

Config File No

Scope Session

Dynamic Yes

Default Value ON

This variable is used to enable/disable wsrep replication. When set to `OFF`, server will stop replication and behave like standalone *MySQL* server.

variable `wsrep_OSU_method`

Command Line Yes

Config File Yes

Scope Global and Session

Dynamic Yes

Default Value TOI

This variable can be used to select schema upgrade method. Available values are:

- `TOI`: When the *Total Order Isolation* method is selected, data definition language (DDL) statements are processed in the same order with regards to other transactions in each cluster node. This guarantees data consistency. In case of DDL statements, cluster will have parts of database locked and it will behave like a single server. In some cases (like big `ALTER TABLE`) this could have impact on cluster's performance and high availability, but it could be fine for quick changes that happen almost instantly (like fast index changes). When DDL statements are processed under TOI, the DDL statement will be replicated up front to the cluster. That is, cluster will assign global transaction ID for the DDL statement before DDL processing begins. Then every node in the cluster has the responsibility to execute the DDL statement in the given slot in the sequence of incoming transactions, and this DDL execution has to happen with high priority.
- `RSU`: When the *Rolling Schema Upgrade* method is selected, DDL statements won't be replicated across the cluster, instead it's up to the user to run them on each node separately. The node applying the changes will desynchronize from the cluster briefly, while normal work happens on all the other nodes. When a DDL statement is processed, node will apply delayed replication events. The schema changes **must** be backwards compatible for this method to work, otherwise the node that receives the change will likely break Galera replication. If replication breaks, SST will be triggered when the node tries to join again but the change will be undone.

Note: This variable's behavior is consistent with *MySQL* behavior for variables that have both global and session scope. This means if you want to change the variable in current session, you need to do it with: `SET wsrep_OSU_method` (without the `GLOBAL` keyword). Setting the variable with `SET GLOBAL wsrep_OSU_method` will change the variable globally but it won't have effect on the current session.

---

variable `wsrep_preordered`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value OFF

Use this variable to enable, transparent handling of preordered replication events (like replication from traditional master). When this variable is enabled, such events will be applied locally first before being replicated to the other

nodes of the cluster. This could increase the rate at which they can be processed, which would be otherwise limited by the latency between the nodes in the cluster.

Preordered events should not interfere with events that originate on the local node. Therefore, you should not run local update queries on a table that is also being updated through asynchronous replication.

variable `wsrep_provider`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value None

This variable should contain the path to the Galera library (like `/usr/lib64/libgalera_smm.so` on *CentOS/RHEL* and `/usr/lib/libgalera_smm.so` on *Debian/Ubuntu*).

variable `wsrep_provider_options`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

This variable contains settings currently used by Galera library.

variable `pxc_strict_mode`

Version Info

- **5.7** – Variable introduced

Command Line Yes

Config File Yes

Scope Global, Session

Dynamic Yes

Default Value ENFORCING or DISABLED

This variable is used to control PXC Strict Mode, which runs validations to avoid the use of experimental and unsupported features in Percona XtraDB Cluster.

Depending on the actual mode you select, upon encountering a failed validation, the server will either throw an error (halting startup or denying the operation), or log a warning and continue running as normal. The following modes are available:

- `DISABLED`: Do not perform strict mode validations and run as normal.
- `PERMISSIVE`: If a vaidation fails, log a warning and continue running as normal.
- `ENFORCING`: If a validation fails during startup, halt the server and throw an error. If a validation fails during runtime, deny the operation and throw an error.
- `MASTER`: The same as `ENFORCING` except that the validation of *explicit table locking* is not performed. This mode can be used with clusters in which write operations are isolated to a single node.

By default, pxc_strict_mode is set to ENFORCING, except if the node is acting as a standalone server or the node is bootstrapping, then pxc_strict_mode defaults to DISABLED.

Note: When changing the value of pxc_strict_mode from DISABLED or PERMISSIVE to ENFORCING or MASTER, ensure that the following configuration is used:

- wsrep_replicate_myisam=OFF
- binlog_format=ROW
- log_output=FILE or log_output=NONE or log_output=FILE,NONE
- tx_isolation=SERIALIZABLE

For more information, see *PXC Strict Mode*.

variable wsrep_recover

Command Line No

Config File Yes

Scope Global

Dynamic No

Default Value OFF

Location mysqld_safe

When server is started with this variable, it will parse Global Transaction ID (GTID) from log, and if the GTID is found, assign it as initial position for actual server start. This option is used to recover GTID.

variable wsrep_reject_queries

Command Line No

Config File Yes

Scope Global

Dynamic Yes

Default Value NONE

This variable can be used to reject queries for the node. This can be useful during upgrades for keeping node up (with provider enabled) without accepting queries. Using read-only is recommended here unless you want to kill existing queries. This variable accepts the following values:

- NONE: Nothing is rejected (default)
- ALL: All queries are rejected with Error 1047: Unknown command
- ALL_KILL: All queries are rejected and existing client connections are also killed without waiting.

Note: This variable doesn't affect Galera replication in any way, only the applications which connect to database are affected. If you want to desync a node, then use wsrep_desync.

---

variable wsrep_replicate_myisam

Command Line Yes

Config File Yes

Scope Session, Global

Dynamic No

Default Value OFF

This variable defines whether MyISAM should be replicated or not. It is disabled by default, because MyISAM replication is still experimental.

On the global level, `wsrep_replicate_myisam` can be set only before boot-up. On session level, you can change it during runtime as well.

For older nodes in the cluster, `wsrep_replicate_myisam` should work since the TOI decision (for MyISAM DDL) is done on origin node. Mixing of non-MyISAM and MyISAM tables in the same DDL statement is not recommended when `wsrep_replicate_myisam` is disabled, since if any table in the list is MyISAM, the whole DDL statement is not put under TOI.

Note: You should keep in mind the following when using MyISAM replication:

- DDL (CREATE/DROP/TRUNCATE) statements on MyISAM will be replicated irrespective of `wsrep_replicate_miysam` value
- DML (INSERT/UPDATE/DELETE) statements on MyISAM will be replicated only if `wsrep_replicate_myisam` is enabled
- SST will get full transfer irrespective of `wsrep_replicate_myisam` value (it will get MyISAM tables from donor)
- Difference in configuration of `pxc-cluster` node on enforce_storage_engine front may result in picking up different engine for same table on different nodes
- `CREATE TABLE AS SELECT` (CTAS) statements use non-TOI replication and are replicated only if there is involvement of InnoDB table that needs transactions (involvement of MyISAM table will cause CTAS statement to skip replication).

---

variable `wsrep_restart_slave`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value OFF

This variable controls if *MySQL* slave should be restarted automatically when node joins back to cluster, because asynchronous replication slave thread is stopped when the node tries to apply next replication event while the node is in non-primary state.

variable `wsrep_retry_autocommit`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 1

This variable sets the number of times autocommitted transactions will be tried in the cluster if it encounters certification errors. In case there is a conflict, it should be safe for the cluster node to simply retry the statement without the client's knowledge hoping that it will pass next time. This can be useful to help an application using autocommit to avoid deadlock errors that can be triggered by replication conflicts. If this variable is set to `0` transaction won't be retried and if it is set to `1`, it will be retried once.

variable `wsrep_slave_FK_checks`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value ON

This variable is used to control if Foreign Key checking is done for applier threads.

variable `wsrep_slave_threads`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value 1

This variable controls the number of threads that can apply replication transactions in parallel. Galera supports true parallel replication, replication that applies transactions in parallel only when it is safe to do so. The variable is dynamic, you can increase/decrease it at any time.

Note: When you decrease it, it won't kill the threads immediately but stop them after they are done applying current transaction (the effect with increase is immediate though).

If any replication consistency problems are encountered, it's recommended to set this back to `1` to see if that resolves the issue. The default value can be increased for better throughput.

You may want to increase it as suggested in Codership documentation, in `JOINED` state for instance to speed up the catchup process to `SYNCED`.

You can also estimate the optimal value for this from `wsrep_cert_deps_distance` as suggested on this page.

You can also refer to this for more configuration tips.

variable `wsrep_slave_UK_checks`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value OFF

This variable is used to control if Unique Key checking is done for applier threads.

variable `wsrep_sst_auth`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Format <username>:<password>

This variable should contain the authentication information needed for State Snapshot Transfer (SST). Required information depends on the method selected in the wsrep_sst_method. More information about required authentication can be found in the *State Snapshot Transfer* documentation. This variable will appear masked in the logs and in the SHOW VARIABLES query.

variable wsrep_sst_donor

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

This variable contains the name (wsrep_node_name) of the preferred donor for SST. If no node is selected as a preferred donor, it will be chosen from one of the available nodes automatically **if and only if** there is a terminating comma at the end (like 'node1,node2,'). Otherwise, if there is no terminating comma, the list of nodes in wsrep_sst_donor is considered absolute, and thus it won't fall back even if other nodes are available. Please check the note for sst-initial-timeout if you are using it without terminating comma or want joiner to wait more than default 100 seconds.

variable wsrep_sst_donor_rejects_queries

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value OFF

When this variable is enabled, SST donor node will not accept incoming queries, instead it will reject queries with UNKNOWN COMMAND error code. This can be used to signal load-balancer that the node isn't available.

variable wsrep_sst_method

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value xtrabackup-v2

Recommended xtrabackup-v2

This variable sets up the method for taking the State Snapshot Transfer (SST). Available values are:

- xtrabackup-v2: Uses *Percona XtraBackup* to perform SST. This method requires wsrep_sst_auth to be set up with <user>:<password> which it will use on donor. Privileges and perimssions needed for running *Percona XtraBackup* can be found here.

  This is the **recommended** and default option for PXC. For more details, please check *Percona XtraBackup SST Configuration*.

  Note: This method is currently recommended if you have innodb-log-group_home-dir/innodb-data-home-dir in your config. Refer to sst-special-dirs for more information.

- rsync: Uses rsync to perform the SST, this method doesn't use the wsrep_sst_auth

- `mysqldump`: Uses `mysqldump` to perform the SST, this method requires `wsrep_sst_auth` to be set up with <user>:<password>, where user has root privileges on the server.

  Note: This mehotd is not recommended unless it is required for specific reasons. Also, it is not compatible with `bind_address` set to `127.0.0.1` or `localhost`, and will cause startup to fail if set so.

- `<custom_script_name>`: Galera supports Scriptable State Snapshot Transfer. This enables users to create their own custom script for performing an SST. For example, you can create a script `/usr/bin/wsrep_MySST.sh` and specify `MySST` for this variable to run your custom SST script.
- `skip`: Use this to skip SST, it can be used when initially starting the cluster and manually restoring the same data to all nodes. It shouldn't be used as permanent setting because it could lead to data inconsistency across the nodes.

Note: Only `xtrabackup-v2` and `rsync` provide `gtid_mode async-slave` support during SST.

variable `wsrep_sst_receive_address`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value AUTO

This variable is used to configure address on which the node expects SST.

variable `wsrep_start_position`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

This variable contains the `UUID:seqno` value. By setting all the nodes to have the same value for this option, cluster can be set up without the state transfer.

variable `wsrep_sync_wait`

Command Line Yes

Config File Yes

Scope Global, Session

Dynamic Yes

This variable is used to control causality checks on some SQL statements, such as `SELECT`, `BEGIN`/`END`, `SHOW STATUS`, but not on some autocommit SQL statements `UPDATE` and `INSERT`. Causality check is determined by bitmask:

- `1` Indicates check on `READ` statements, including `SELECT`, `SHOW`, `BEGIN`/`START TRANSACTION`.
- `2` Indicates check on `UPDATE` and `DELETE` statements.
- `4` Indicates check on `INSERT` and `REPLACE` statements

This variable replaced the `wsrep_causal_reads` variable. Setting `wsrep_sync_wait` to `1` is the equivalent of setting `wsrep_causal_reads` to `ON`.

6.4 Index of wsrep_provider options

The following variables can be set and checked in the wsrep_provider_options variable. The value of the variable can be changed in the *MySQL* configuration file, my.cnf, or by setting the variable value in the *MySQL* client.

To change the value in my.cnf, the following syntax should be used:

```
wsrep_provider_options="variable1=value1;[variable2=value2]"
```

For example to set the size of the Galera buffer storage to 512 MB, specify the following in my.cnf:

```
wsrep_provider_options="gcache.size=512M"
```

Dynamic variables can be changed from the *MySQL* client using the SET GLOBAL command. For example, to change the value of the pc.ignore_sb, use the following command:

```
mysql> SET GLOBAL wsrep_provider_options="pc.ignore_sb=true";
```

6.4.1 Index

variable base_dir

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value value of datadir

This variable specifies the data directory.

variable base_host

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value value of wsrep_node_address

This variable sets the value of the node's base IP. This is an IP address on which Galera listens for connections from other nodes. Setting this value incorrectly would stop the node from communicating with other nodes.

variable base_port

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 4567

This variable sets the port on which Galera listens for connections from other nodes. Setting this value incorrectly would stop the node from communicating with other nodes.

variable `cert.log_conflicts`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value no

This variable is used to specify if the details of the certification failures should be logged.

variable `debug`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value no

When this variable is set to `yes`, it will enable debugging.

variable `evs.auto_evict`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value 0

Number of entries allowed on delayed list until auto eviction takes place. Setting value to `0` disables auto eviction protocol on the node, though node response times will still be monitored. EVS protocol version (`evs.version`) `1` is required to enable auto eviction.

variable `evs.causal_keepalive_period`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value value of `evs.keepalive_period`

This variable is used for development purposes and shouldn't be used by regular users.

variable `evs.debug_log_mask`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value 0x1

This variable is used for EVS (Extended Virtual Synchrony) debugging. It can be used only when `wsrep_debug` is set to `ON`.

variable `evs.delay_margin`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value PT1S

Time period that a node can delay its response from expected until it is added to delayed list. The value must be higher than the highest RTT between nodes.

variable `evs.delayed_keep_period`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value PT30S

Time period that node is required to remain responsive until one entry is removed from delayed list.

variable `evs.evict`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Manual eviction can be triggered by setting the `evs.evict` to a certain node value. Setting the `evs.evict` to an empty string will clear the evict list on the node where it was set.

variable `evs.inactive_check_period`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value PT0.5S

This variable defines how often to check for peer inactivity.

variable `evs.inactive_timeout`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value PT15S

This variable defines the inactivity limit, once this limit is reached the node will be considered dead.

variable `evs.info_log_mask`

Command Line No

Config File Yes

Scope Global

Dynamic No

Default Value 0

This variable is used for controlling the extra EVS info logging.

variable `evs.install_timeout`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value PT7.5S

This variable defines the timeout on waiting for install message acknowledgments.

variable `evs.join_retrans_period`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value PT1S

This variable defines how often to retransmit EVS join messages when forming cluster membership.

variable `evs.keepalive_period`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value PT1S

This variable defines how often to emit keepalive beacons (in the absence of any other traffic).

variable `evs.max_install_timeouts`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 1

This variable defines how many membership install rounds to try before giving up (total rounds will be evs.max_install_timeouts + 2).

variable evs.send_window

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 4

This variable defines the maximum number of data packets in replication at a time. For WAN setups, the variable can be set to a considerably higher value than default (for example,512). The value must not be less than evs.user_send_window.

variable evs.stats_report_period

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value PT1M

This variable defines the control period of EVS statistics reporting.

variable evs.suspect_timeout

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value PT5S

This variable defines the inactivity period after which the node is "suspected" to be dead. If all remaining nodes agree on that, the node will be dropped out of cluster even before evs.inactive_timeout is reached.

variable evs.use_aggregate

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value true

When this variable is enabled, smaller packets will be aggregated into one.

variable evs.user_send_window

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value 2

This variable defines the maximum number of data packets in replication at a time. For WAN setups, the variable can be set to a considerably higher value than default (for example, 512).

variable `evs.version`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 0

This variable defines the EVS protocol version. Auto eviction is enabled when this variable is set to `1`. Default `0` is set for backwards compatibility.

variable `evs.view_forget_timeout`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value P1D

This variable defines the timeout after which past views will be dropped from history.

variable `gcache.dir`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value *datadir*

This variable can be used to define the location of the `galera.cache` file.

variable `gcache.keep_pages_count`

Command Line Yes

Config File Yes

Scope Local, Global

Dynamic Yes

Default Value 0

This variable is used to limit the number of overflow pages rather than the total memory occupied by all overflow pages. Whenever either this or `gcache.keep_pages_size` variables are updated at runtime to a non-zero value, cleanup is called on excess overflow pages to delete them.

variable `gcache.keep_pages_size`

Command Line Yes

Config File Yes

Scope Local, Global

Dynamic No

Default Value 0

This variable is used to specify total size of the pages in storage to keep for caching purposes. If only page storage is enabled, one page is always present.

variable `gcache.mem_size`

Version Deprecated in `5.6.22-25.8`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 0

This variable was used to define how much RAM is available for the system.

Warning: This variable has been deprecated and shouldn't be used as it could cause a node to crash.

variable `gcache.name`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value /var/lib/mysql/galera.cache

This variable can be used to specify the name of the Galera cache file.

variable `gcache.page_size`

Command Line No

Config File Yes

Scope Global

Dynamic No

Default Value 128M

This variable can be used to specify the size of the page files in the page storage.

variable `gcache.size`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 128M

Size of the transaction cache for Galera replication. This defines the size of the `galera.cache` file which is used as source for *IST*. The bigger the value of this variable, the better are chances that the re-joining node will get IST instead of *SST*.

variable `gcs.fc_debug`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 0

This variable specifies after how many writesets the debug statistics about SST flow control will be posted.

variable `gcs.fc_factor`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 1

This variable is used for replication flow control. Replication is resumed when the slave queue drops below `gcs.fc_factor` * `gcs.fc_limit`.

variable `gcs.fc_limit`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 16

This variable is used for replication flow control. Replication is paused when the slave queue exceeds this limit.

variable `gcs.fc_master_slave`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value NO

This variable is used to specify if there is only one master node in the cluster.

variable `gcs.max_packet_size`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 64500

This variable is used to specify the writeset size after which they will be fragmented.

variable `gcs.max_throttle`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 0.25

This variable specifies how much the replication can be throttled during the state transfer in order to avoid running out of memory. Value can be set to `0.0` if stopping replication is acceptable in order to finish state transfer.

variable `gcs.recv_q_hard_limit`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 9223372036854775807

This variable specifies the maximum allowed size of the receive queue. This should normally be `(RAM + swap) / 2`. If this limit is exceeded, Galera will abort the server.

variable `gcs.recv_q_soft_limit`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 0.25

This variable specifies the fraction of the `gcs.recv_q_hard_limit` after which replication rate will be throttled.

variable `gcs.sync_donor`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value No

This variable controls if the rest of the cluster should be in sync with the donor node. When this variable is set to `YES`, the whole cluster will be blocked if the donor node is blocked with SST.

variable `gmcast.listen_addr`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value tcp://0.0.0.0:4567

This variable defines the address on which the node listens to connections from other nodes in the cluster.

variable `gmcast.mcast_addr`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value None

This variable should be set up if UDP multicast should be used for replication.

variable `gmcast.mcast_ttl`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 1

This variable can be used to define TTL for multicast packets.

variable `gmcast.peer_timeout`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value PT3S

This variable specifies the connection timeout to initiate message relaying.

variable `gmcast.segment`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 0

This variable specifies the group segment this member should be a part of. Same segment members are treated as equally physically close.

variable `gmcast.time_wait`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value PT5S

This variable specifies the time to wait until allowing peer declared outside of stable view to reconnect.

variable `gmcast.version`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 0

This variable shows which gmcast protocol version is being used.

variable `ist.recv_addr`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value value of `wsrep_node_address`

This variable specifies the address on which the node listens for Incremental State Transfer (*IST*).

variable `pc.announce_timeout`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value PT3S

Cluster joining announcements are sent every 1/2 second for this period of time or less if other nodes are discovered.

variable `pc.checksum`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value true

This variable controls whether replicated messages should be checksummed or not.

variable `pc.ignore_quorum`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value false

When this variable is set to `TRUE`, the node will completely ignore quorum calculations. This should be used with extreme caution even in master-slave setups, because slaves won't automatically reconnect to master in this case.

variable `pc.ignore_sb`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value false

When this variable is set to `TRUE`, the node will process updates even in the case of a split brain. This should be used with extreme caution in multi-master setup, but should simplify things in master-slave cluster (especially if only 2 nodes are used).

variable `pc.linger`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value PT20S

This variable specifies the period for which the PC protocol waits for EVS termination.

variable `pc.npvo`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value false

When this variable is set to `TRUE`, more recent primary components override older ones in case of conflicting primaries.

variable `pc.recovery`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value true

When this variable is set to `true`, the node stores the Primary Component state to disk. The Primary Component can then recover automatically when all nodes that were part of the last saved state re-establish communication with each other. This feature allows automatic recovery from full cluster crashes, such as in the case of a data center power outage. A subsequent graceful full cluster restart will require explicit bootstrapping for a new Primary Component.

variable `pc.version`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 0

This status variable is used to check which PC protocol version is used.

variable `pc.wait_prim`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value true

When set to `TRUE`, the node waits for a primary component for the period of time specified in `pc.wait_prim_timeout`. This is useful to bring up a non-primary component and make it primary with `pc.bootstrap`.

variable `pc.wait_prim_timeout`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value PT30S

This variable is used to specify the period of time to wait for a primary component.

variable `pc.weight`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value 1

This variable specifies the node weight that's going to be used for Weighted Quorum calculations.

variable `protonet.backend`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value asio

This variable is used to define which transport backend should be used. Currently only `ASIO` is supported.

variable `protonet.version`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 0

This status variable is used to check which transport backend protocol version is used.

variable `repl.causal_read_timeout`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value PT30S

This variable specifies the causal read timeout.

variable `repl.commit_order`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 3

This variable is used to specify out-of-order committing (which is used to improve parallel applying performance). The following values are available:

- `0` - BYPASS: all commit order monitoring is turned off (useful for measuring performance penalty)
- `1` - OOOC: allow out-of-order committing for all transactions
- `2` - LOCAL_OOOC: allow out-of-order committing only for local transactions
- `3` - NO_OOOC: no out-of-order committing is allowed (strict total order committing)

variable `repl.key_format`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value FLAT8

This variable is used to specify the replication key format. The following values are available:

- `FLAT8` - short key with higher probability of key match false positives
- `FLAT16` - longer key with lower probability of false positives
- `FLAT8A` - same as `FLAT8` but with annotations for debug purposes

- `FLAT16A` - same as `FLAT16` but with annotations for debug purposes

variable `repl.max_ws_size`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 2147483647

This variable is used to specify the maximum size of a write-set in bytes. This is limited to 2 gygabytes.

variable `repl.proto_max`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 7

This variable is used to specify the highest communication protocol version to accept in the cluster. Used only for debugging.

variable `socket.checksum`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value 2

This variable is used to choose the checksum algorithm for network packets. The following values are available:

- `0` - disable checksum
- `1` - plain `CRC32` (used in Galera 2.x)
- `2` - hardware accelerated `CRC32-C`

variable `socket.ssl`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value No

This variable is used to specify if SSL encryption should be used.

variable `socket.ssl_cert`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

This variable is used to specify the path (absolute or relative to working directory) to an SSL certificate (in PEM format).

variable `socket.ssl_key`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

This variable is used to specify the path (absolute or relative to working directory) to an SSL private key for the certificate (in PEM format).

variable `socket.ssl_compression`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value yes

This variable is used to specify if the SSL compression is to be used.

variable `socket.ssl_cipher`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

Default Value AES128-SHA

This variable is used to specify what cypher will be used for encryption.

6.5 Index of files created by PXC

- **`GRA_*.log`** These files contain binlog events in ROW format representing the failed transaction. That means that the slave thread was not able to apply one of the transactions. For each of those file, a corresponding warning or error message is present in the mysql error log file. Those error can also be false positives like a bad `DDL` statement (dropping a table that doesn't exists for example) and therefore nothing to worry about. However it's always recommended to check these log to understand what's is happening.

  To be able to analyze these files binlog header needs to be added to the log file. To create the `GRA_HEADER` file you need an instance running with `binlog_checksum` set to `NONE` and extract first 120 bytes from the binlog file:

  ```
  $ head -c 123 mysqld-bin.000001 > GRA_HEADER
  $ cat GRA_HEADER > /var/lib/mysql/GRA_1_2-bin.log
  $ cat /var/lib/mysql/GRA_1_2.log >> /var/lib/mysql/GRA_1_2-bin.log
  $ mysqlbinlog -vvv /var/lib/mysql/GRA_1_2-bin.log
  ```

```
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=1*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
DELIMITER /*!*/;
# at 4
#160809  16:04:05 server id 3  end_log_pos 123     Start: binlog v 4, server v 5.7.12-5rc1-1
# Warning: this binlog is either in use or was not closed properly.
ROLLBACK/*!*/;
BINLOG '
nbGpVw8DAAAAdwAAAHsAAAABAAQANS43LjEyLTVyYzEtbG9nAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAACdsalXEzgNAAgAEgAEBAQEEgAAXwAEGggAAAAICAgCAAAACgoKKioAEjQA
ALfQ8hw=
'/*!*/;
# at 123
#160809  16:05:49 server id 2  end_log_pos 75     Query    thread_id=11    exec_time=0    er
use `test`/*!*/;
SET TIMESTAMP=1470738949/*!*/;
SET @@session.pseudo_thread_id=11/*!*/;
SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=0, @@session.unique_checks=1,
SET @@session.sql_mode=1436549152/*!*/;
SET @@session.auto_increment_increment=1, @@session.auto_increment_offset=1/*!*/;
/*!\C utf8 *//*!*/;
SET @@session.character_set_client=33,@@session.collation_connection=33,@@session.collation_
SET @@session.lc_time_names=0/*!*/;
SET @@session.collation_database=DEFAULT/*!*/;
drop table t
/*!*/;
SET @@SESSION.GTID_NEXT= 'AUTOMATIC' /* added by mysqlbinlog */ /*!*/;
DELIMITER ;
# End of log file
/*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE*/;
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=0*/;
```

This information can be used for checking the *MySQL* error log for the corresponding error message.

```
160805  9:33:37 8:52:21 [ERROR] Slave SQL: Error 'Unknown table 'test'' on query. Default da
160805  9:33:37 8:52:21 [Warning] WSREP: RBR event 1 Query apply warning: 1, 3
```

In this example `DROP TABLE` statement was executed on a table that doesn't exist.

- **`galera.cache`** This file is used as a main writeset store. It's implemented as a permanent ring-buffer file that is preallocated on disk when the node is initialized. File size can be controlled with the variable `gcache.size`. If this value is bigger, more writesets are cached and chances are better that the re-joining node will get *IST* instead of *SST*. Filename can be changed with the `gcache.name` variable.
- **`grastate.dat`** This file contains the Galera state information.
  - `version` - grastate version
  - `uuid` - a unique identifier for the state and the sequence of changes it undergoes.For more information on how UUID is generated see *UUID*.
  - `seqno` - Ordinal Sequence Number, a 64-bit signed integer used to denote the position of the change in the sequence. `seqno` is `0` when no writesets have been generated or applied on that node, i.e., not applied/generated across the lifetime of a `grastate` file. `-1` is a special value for the `seqno` that is kept in the `grastate.dat` while the server is running to allow Galera to distinguish between a clean and an unclean shutdown. Upon a clean shutdown, the correct `seqno` value is written to the file. So, when the server is brought back up, if the value is still `-1` , this means that the server did not shut down cleanly. If the value is greater than `0`, this means that the shutdown was clean. `-1` is then written again to the file in order to allow the server to correctly detect if the next shutdown was clean in the same manner.

  - `cert_index` - cert index restore through grastate is not implemented yet

Examples of this file look like this:

In case server node has this state when not running it means that that node crashed during the transaction processing.

```
# GALERA saved state
version: 2.1
uuid:    1917033b-7081-11e2-0800-707f5d3b106b
seqno:   -1
cert_index:
```

In case server node has this state when not running it means that the node was gracefully shut down.

```
# GALERA saved state
version: 2.1
uuid:    1917033b-7081-11e2-0800-707f5d3b106b
seqno:   5192193423942
cert_index:
```

In case server node has this state when not running it means that the node crashed during the DDL.

```
# GALERA saved state
version: 2.1
uuid:    00000000-0000-0000-0000-000000000000
seqno:   -1
cert_index:
```

- `gvwstate.dat` This file is used for Primary Component recovery feature. This file is created once primary component is formed or changed, so you can get the latest primary component this node was in. And this file is deleted when the node is shutdown gracefully.

  First part contains the node *UUID* information. Second part contains the view information. View information is written between `#vwbeg` and `#vwend`. View information consists of:

- view_id: [view_type] [view_uuid] [view_seq]. - `view_type` is always `3` which means primary view. `view_uuid` and `view_seq` identifies a unique view, which could be perceived as identifier of this primary component.
- bootstrap: [bootstarp_or_not]. - It could be `0` or `1`, but it does not affect primary component recovery process now.
- member: [node's uuid] [node's segment]. - it represents all nodes in this primary component.

  Example of this file looks like this:

```
my_uuid: c5d5d990-30ee-11e4-aab1-46d0ed84b408
#vwbeg
view_id: 3 bc85bd53-31ac-11e4-9895-1f2ce13f2542 2
bootstrap: 0
member: bc85bd53-31ac-11e4-9895-1f2ce13f2542 0
member: c5d5d990-30ee-11e4-aab1-46d0ed84b408 0
#vwend
```

6.6 Frequently Asked Questions

6.6.1 How do I report bugs?

All bugs can be reported on Launchpad. Please submit `error.log` files from **all** the nodes.

6.6.2 How do I solve locking issues like auto-increment?

For auto-increment, Percona XtraDB Cluster changes `auto_increment_offset` for each new node. In a single-node workload, locking is handled in the same way as *InnoDB*. In case of write load on several nodes, Percona XtraDB Cluster uses optimistic locking and the application may receive lock error in response to `COMMIT` query.

6.6.3 What if a node crashes and InnoDB recovery rolls back some transactions?

When a node crashes, after restarting, it will copy the whole dataset from another node (if there were changes to data since the crash).

6.6.4 How can I check the Galera node health?

To check the health of a Galera node, use the following query:

```
SELECT 1 FROM dual;
```

The following results of the previous query are possible:

- You get the row with `id=1` (node is healthy)
- Unknown error (node is online, but Galera is not connected/synced with the cluster)
- Connection error (node is not online)

You can also check a node's health with the `clustercheck` script. First set up the `clustercheck` user:

```
GRANT USAGE ON *.* TO 'clustercheck'@'localhost' IDENTIFIED BY PASSWORD '*2470C0C06DEE42FD1618BB99005
```

You can then check a node's health by running the `clustercheck` script:

```
/usr/bin/clustercheck clustercheck password 0
```

If the node is running, you should get the following status:

```
HTTP/1.1 200 OK
Content-Type: text/plain
Connection: close
Content-Length: 40

Percona XtraDB Cluster Node is synced.
```

In case node isn't synced or if it is offline, status will look like:

```
HTTP/1.1 503 Service Unavailable
Content-Type: text/plain
Connection: close
Content-Length: 44

Percona XtraDB Cluster Node is not synced.
```

Note: The `clustercheck` script has the following syntax:

```
<user> <pass> <available_when_donor=0|1> <log_file> <available_when_readonly=0|1>
<defaults_extra_file>
```

Recommended: `server_args = user pass 1 /var/log/log-file 0 /etc/my.cnf.local`

Compatibility: `server_args = user pass 1 /var/log/log-file 1 /etc/my.cnf.local`

6.6.5 How does Percona XtraDB Cluster handle big transactions?

Percona XtraDB Cluster populates write set in memory before replication, and this sets the limit for the size of transactions that make sense. There are wsrep variables for maximum row count and maximum size of write set to make sure that the server does not run out of memory.

6.6.6 Is it possible to have different table structures on the nodes?

For example, if there are four nodes, with four tables: `sessions_a`, `sessions_b`, `sessions_c`, and `sessions_d`, and you want each table in a separate node, this is not possible for InnoDB tables. However, it will work for MEMORY tables.

6.6.7 What if a node fails or there is a network issue between nodes?

The quorum mechanism in Percona XtraDB Cluster will decide which nodes can accept traffic and will shut down the nodes that do not belong to the quorum. Later when the failure is fixed, the nodes will need to copy data from the working cluster.

The algorithm for quorum is Dynamic Linear Voting (DLV). The quorum is preserved if (and only if) the sum weight of the nodes in a new component strictly exceeds half that of the preceding Primary Component, minus the nodes which left gracefully.

The mechanism is described in detail in Galera documentation.

6.6.8 How would the quorum mechanism handle split brain?

The quorum mechanism cannot handle split brain. If there is no way to decide on the primary component, Percona XtraDB Cluster has no way to resolve a *split brain*. The minimal recommendation is to have 3 nodes. However, it is possibile to allow a node to handle traffic with the following option:

```
wsrep_provider_options="pc.ignore_sb = yes"
```

6.6.9 Why a node stops accepting commands if the other one fails in a 2-node setup?

This is expected behavior to prevent *split brain*. For more information, see previous question or Galera documentation.

6.6.10 Is it possible to set up a cluster without state transfer?

It is possible in two ways:

1. By default, Galera reads starting position from a text file `<datadir>/grastate.dat`. Make this file identical on all nodes, and there will be no state transfer after starting a node.
2. Use the `wsrep_start_position` variable to start the nodes with the same `UUID:seqno` value.

6.6.11 What TCP ports are used by Percona XtraDB Cluster?

You may need to open up to four ports if you are using a firewall:

1. Regular MySQL port (default is 3306).
2. Port for group communication (default is 4567). It can be changed using the following option:

   ```
   wsrep_provider_options ="gmcast.listen_addr=tcp://0.0.0.0:4010; "
   ```

3. Port for State Snaphot Transfer (default is 4444). It can be changed using the following option:

   ```
   wsrep_sst_receive_address=10.11.12.205:5555
   ```

4. Port for Incremental State Transfer (default is port for group communication + 1 or 4568). It can be changed using the following option:

   ```
   wsrep_provider_options = "ist.recv_addr=10.11.12.206:7777; "
   ```

6.6.12 Is there “async” mode or only “sync” commits are supported?

Percona XtraDB Cluster does not support “async” mode, all commits are synchronous on all nodes. To be precise, the commits are “virtually” synchronous, which means that the transaction should pass *certification* on nodes, not physical commit. Certification means a guarantee that the transaction does not have conflicts with other transactions on the corresponding node.

6.6.13 Does it work with regular MySQL replication?

Yes. On the node you are going to use as master, you should enable `log-bin` and `log-slave-update` options.

6.6.14 Why the init script (/etc/init.d/mysql) does not start?

Try to disable SELinux with the following command:

```
echo 0 > /selinux/enforce
```

6.6.15 What does "nc: invalid option – 'd'" in the sst.err log file mean?

This is Debian/Ubuntu specific error. Percona XtraDB Cluster uses `netcat-openbsd` package. This dependency has been fixed in recent releases. Future releases of Percona XtraDB Cluster will be compatible with any `netcat` (see bug #959970).

6.7 Glossary

LSN Each InnoDB page (usually 16kb in size) contains a log sequence number, or LSN. The LSN is the system version number for the entire database. Each page's LSN shows how recently it was changed.

InnoDB Storage engine which provides ACID-compliant transactions and foreign key support, among others improvements over *MyISAM*. It is the default engine for *MySQL* as of the 5.5 series.

MyISAM Previous default storage engine for *MySQL* for versions prior to 5.5. It doesn't fully support transactions but in some scenarios may be faster than *InnoDB*. Each table is stored on disk in 3 files: *.frm*,i `.MYD`, `.MYI`.

GTID Global Transaction ID, in *Percona XtraDB Cluster* it consists of *UUID* and an ordinal sequence number which denotes the position of the change in the sequence.

HAProxy HAProxy is a free, very fast and reliable solution offering high availability, load balancing, and proxying for TCP and HTTP-based applications. It is particularly suited for web sites crawling under very high loads while needing persistence or Layer7 processing. Supporting tens of thousands of connections is clearly realistic with todays hardware. Its mode of operation makes its integration into existing architectures very easy and riskless, while still offering the possibility not to expose fragile web servers to the net.

IST Incremental State Transfer. Functionality which instead of whole state snapshot can catch up with te group by receiving the missing writesets, but only if the writeset is still in the donor's writeset cache.

SST State Snapshot Transfer is the full copy of data from one node to another. It's used when a new node joins the cluster, it has to transfer data from existing node. There are three methods of SST available in *Percona XtraDB Cluster*: **mysqldump**, **rsync** and **xtrabackup**. The downside of *mysqldump* and *rsync* is that the node becomes *READ-ONLY* while data is being copied from one node to another (SST applies `FLUSH TABLES WITH READ LOCK` command). Xtrabackup SST does not require `READ LOCK` for the entire syncing process, only for syncing the *MySQL* system tables and writing the information about the binlog, galera and slave information (same as the regular *Percona XtraBackup* backup). State snapshot transfer method can be configured with the `wsrep_sst_method` variable.

UUID Universally Unique IDentifier which uniquely identifies the state and the sequence of changes node undergoes. 128-bit UUID is a classic DCE UUID Version 1 (based on current time and MAC address). Although in theory this UUID could be generated based on the real MAC-address, in the Galera it is always (without exception) based on the generated pseudo-random addresses ("locally administered" bit in the node address (in the UUID structure) is always equal to unity).

Complete structure of the 128-bit UUID field and explanation for its generation are as follows:

From	To	Length	Content
0	31	32	Bits 0-31 of Coordinated Universal Time (UTC) as a count of 100-nanosecond intervals since 00:00:00.00, 15 October 1582, encoded as big-endian 32-bit number.
32	47	16	Bits 32-47 of UTC as a count of 100-nanosecond intervals since 00:00:00.00, 15 October 1582, encoded as big-endian 16-bit number.
48	59	12	Bits 48-59 of UTC as a count of 100-nanosecond intervals since 00:00:00.00, 15 October 1582, encoded as big-endian 16-bit number.
60	63	4	UUID version number: always equal to 1 (DCE UUID).
64	69	6	most-significants bits of random number, which generated from the server process PID and Coordinated Universal Time (UTC) as a count of 100-nanosecond intervals since 00:00:00.00, 15 October 1582.
70	71	2	UID variant: always equal to binary 10 (DCE variant).
72	79	8	8 least-significant bits of random number, which generated from the server process PID and Coordinated Universal Time (UTC) as a count of 100-nanosecond intervals since 00:00:00.00, 15 October 1582.
80	80	1	Random bit ("unique node identifier").
81	81	1	Always equal to the one ("locally administered MAC address").
82	127	46	Random bits ("unique node identifier"): readed from the `/dev/urandom` or (if `/dev/urandom` is unavailable) generated based on the server process PID, current time and bits of the default "zero node identifier" (entropy data).

XtraBackup *Percona XtraBackup* is an open-source hot backup utility for *MySQL* - based servers that doesn't lock your database during the backup.

XtraDB *Percona XtraDB* is an enhanced version of the InnoDB storage engine, designed to better scale on modern hardware, and including a variety of other features useful in high performance environments. It is fully backwards compatible, and so can be used as a drop-in replacement for standard InnoDB. More information here .

XtraDB Cluster *Percona XtraDB Cluster* is a high availability solution for MySQL.

Percona XtraDB Cluster *Percona XtraDB Cluster* (PXC) is a high availability solution for MySQL.

my.cnf This file refers to the database server's main configuration file. Most Linux distributions place it as `/etc/mysql/my.cnf` or `/etc/my.cnf`, but the location and name depends on the particular installation. Note that this is not the only way of configuring the server, some systems does not have one even and rely on the command options to start the server and its defaults values.

cluster replication Normal replication path for cluster members. Can be encrypted (not by default) and unicast or multicast (unicast by default). Runs on tcp port 4567 by default.

datadir The directory in which the database server stores its databases. Most Linux distribution use `/var/lib/mysql` by default.

donor node The node elected to provide a state transfer (SST or IST).

ibdata Default prefix for tablespace files, e.g. `ibdata1` is a 10MB autoextendable file that *MySQL* creates for the shared tablespace by default.

joiner node The node joining the cluster, usually a state transfer target.

node A cluster node – a single mysql instance that is in the cluster.

primary cluster A cluster with *quorum*. A non-primary cluster will not allow any operations and will give `Unknown command` errors on any clients attempting to read or write from the database.

quorum A majority (> 50%) of nodes. In the event of a network partition, only the cluster partition that retains a quorum (if any) will remain Primary by default.

split brain Split brain occurs when two parts of a computer cluster are disconnected, each part believing that the other is no longer running. This problem can lead to data inconsistency.

.frm For each table, the server will create a file with the `.frm` extension containing the table definition (for all storage engines).

- *genindex*
- *search*

Symbols

B

C

D

E

G

H

I

X