PERCONA

www.percona.com

Percona XtraDB Cluster Documentation

Release 8.0.20-11.3

Percona LLC and/or its affiliates 2009-2020

Oct 22, 2020

CONTENTS

I Introduction 3

II Getting Started 9

III Features 35

IV PXC Security 47

V User's Manual 61

VI How-tos 99

VII Reference 175

Percona XtraDB Cluster is a database clustering solution for MySQL. It ensures high availability, prevents downtime and data loss, and provides linear scalability for a growing environment.

Features of Percona XtraDB Cluster

Feature	Details
Synchronous replication**	Data is written to all nodes simultaneously, or not written at all in case of a failure even on a single node
Multi-source replication	Any node can trigger a data update.
True parallel replication	Multiple threads on replica performing replication on row level
Automatic node provisioning	You simply add a node and it automatically syncs.
Data consistency	No more unsynchronized nodes.
PXC Strict Mode	Avoids the use of experimental and unsupported features
Configuration script for ProxySQL	Percona XtraDB Cluster includes the `proxysql-admin` tool that automatically configures Percona XtraDB Cluster nodes using ProxySQL.
Automatic configuration of SSL encryption	Percona XtraDB Cluster includes the `pxc-encrypt-cluster-traffic` variable that enables automatic configuration of SSL encryption
Optimized Performance	Percona XtraDB Cluster performance is optimized to scale with a growing production workload

Percona XtraDB Cluster 8.0 is fully compatible with *MySQL* Server Community Edition 8.0 and Percona Server for MySQL 8.0.

See also:

Overview of changes in the most recent Percona XtraDB Cluster release *Important changes in Percona XtraDB Cluster 8.0*

***MySQL* Community Edition** https://www.mysql.com/products/community/

Percona Server for MySQL https://www.percona.com/doc/percona-server/LATEST/index.html

How We Made Percona XtraDB Cluster Scale https://www.percona.com/blog/2017/04/19/how-we-made-percona-xtradb-cluster-scale

Data compatibility You can use data created by any MySQL variant.

Application compatibility There is no or minimal application changes required.

Part I

Introduction

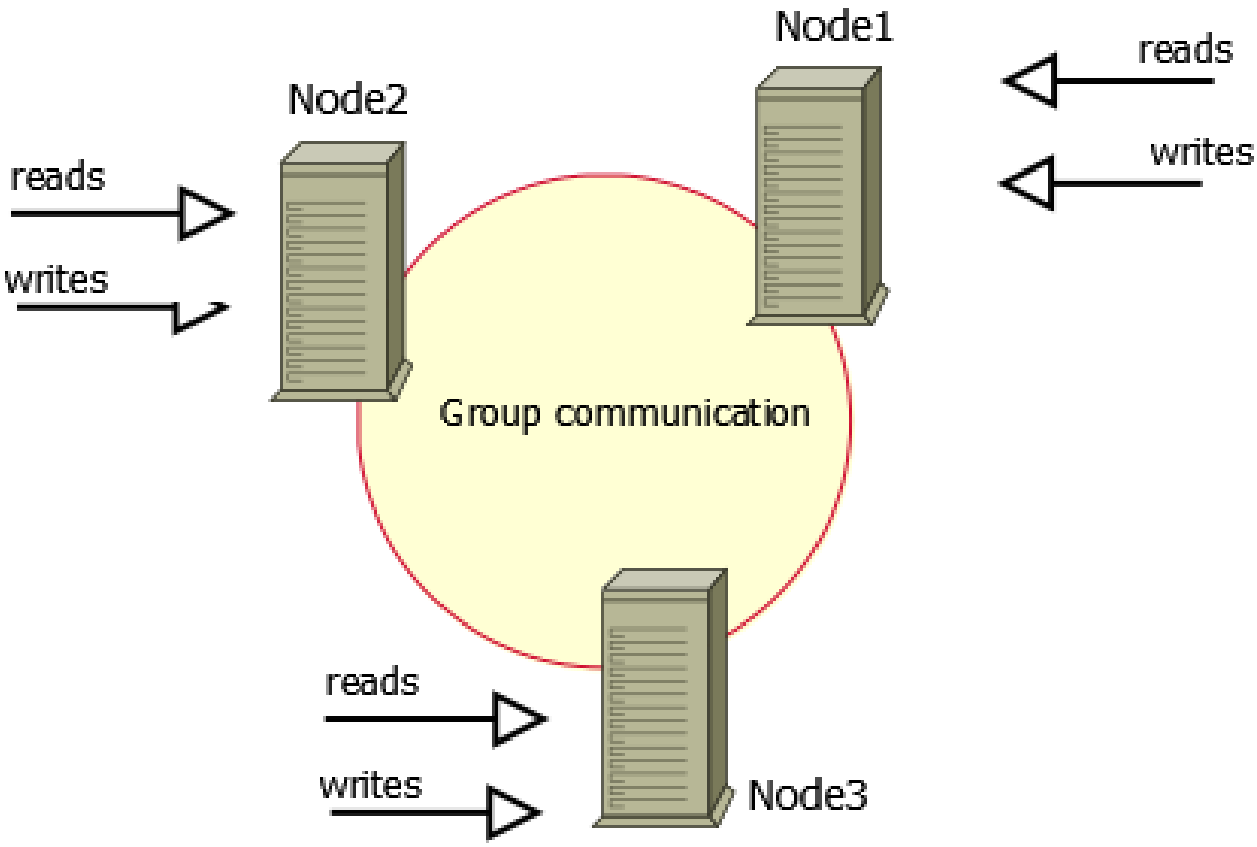
CHAPTER

ONE

ABOUT PERCONA XTRADB CLUSTER

Percona XtraDB Cluster is a fully open-source high-availability solution for MySQL. It integrates Percona Server and Percona XtraBackup with the Galera library to enable synchronous multi-source replication.

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 *100 GB*, it copies *100 GB*.
- 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 data: for 3 nodes you have 3 duplicates.

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. The default and recommended data transfer method is via Percona XtraBackup.

CHAPTER

TWO

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 are not replicated.

Unsupported queries: `LOCK TABLES` and `UNLOCK TABLES` is not supported in multi-source 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.

Transaction issuing `COMMIT` may still be aborted at that stage. Due to cluster-level optimistic concurrency control, 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.

Write throughput of the whole cluster is limited by the 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.

Minimal recommended size of cluster is 3 nodes. The 3rd node can be an arbitrator.

`enforce_storage_engine=InnoDB` is not compatible with `wsrep_replicate_myisam=OFF` *wsrep_replicate_myisam* is set to `OFF` by default.

Avoid `ALTER TABLE ... IMPORT/EXPORT` workloads when running Percona XtraDB Cluster in cluster mode. It can lead to node inconsistency if not executed in sync on all nodes.

All tables must have a primary key.

This ensures that the same rows appear in the same order on different nodes. The `DELETE` statement is not supported on tables without a primary key.

See also:

Galera Documentation: Tables without Primary Keys http://galeracluster.com/documentation-webpages/limitations.html#tables-without-primary-keys

Avoid reusing the names of persistent tables for temporary tables Although MySQL does allow having temporary tables named the same as persistent tables, this approach is not recommended.

Galera Cluster blocks the replication of those persistent tables the names of which match the names of temporary tables.

With wsrep_debug set to *1*, the error log may contain the following message:

```
... [Note] WSREP: TO BEGIN: -1, 0 : create table t (i int) engine=innodb
... [Note] WSREP: TO isolation skipped for: 1, sql: create table t (i int)␣
↪engine=innodb.Only temporary tables affected.
```

See also:

MySQL Documentation: Problems with temporary tables

Part II

Getting Started

CHAPTER

THREE

QUICK START GUIDE FOR PERCONA XTRADB CLUSTER

This guide describes the procedure for setting up Percona XtraDB Cluster.

Examples provided in this guide assume there are three Percona XtraDB Cluster nodes, as a common choice for trying out and testing:

Node	Host	IP
Node 1	pxc1	192.168.70.61
Node 2	pxc2	192.168.70.62
Node 3	pxc3	192.168.70.63

Note: Avoid creating a cluster with two or any even number of nodes, because this can lead to *split brain*. For more information, see *Cluster Failover*.

The following procedure provides an overview with links to details for every step:

1. *Install Percona XtraDB Cluster* on all nodes and set up root access for them.

   It is recommended to install from official Percona repositories:

   - On Red Hat and CentOS, *install using YUM*.
   - On Debian and Ubuntu, *install using APT*.

2. *Configure all nodes* with relevant settings required for write-set replication.

   This includes path to the Galera library, location of other nodes, etc.

3. *Bootstrap the first node* to initialize the cluster.

   This must be the node with your main database, which will be used as the data source for the cluster.

4. *Add other nodes* to the cluster.

   Data on new nodes joining the cluster is overwritten in order to synchronize it with the cluster.

5. *Verify replication*.

   Although cluster initialization and node provisioning is performed automatically, it is a good idea to ensure that changes on one node actually replicate to other nodes.

6. *Install ProxySQL*.

   To complete the deployment of the cluster, a high-availability proxy is required. We recommend installing ProxySQL on client nodes for efficient workload management across the cluster without any changes to the applications that generate queries.

Percona Monitoring and Management

Percona Monitoring and Management is the best choice for managing and monitoring Percona XtraDB Cluster performance. It provides visibility for the cluster and enables efficient troubleshooting.

CHAPTER

FOUR

INSTALLING PERCONA XTRADB CLUSTER

Install Percona XtraDB Cluster on all hosts that you are planning to use as cluster nodes and ensure that you have root access to the MySQL server on each one.

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

- *Debian or Ubuntu*
- *Red Hat or CentOS*

Important: After installing Percona XtraDB Cluster the `mysql` service is *stopped* but *enabled* so that it may start the next time the system is restarted. The service starts if the the grastate.dat file exists and the value of `seqno` is not **-1**.

See also:

More information about Galera state information in *Index of files created by PXC* *grastat.dat*

Installation Alternatives

Percona also provides a generic tarball with all required files and binaries for manual installation:

- *Installing Percona XtraDB Cluster from Binary Tarball*

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

If you want to run Percona XtraDB Cluster using Docker, see *Running Percona XtraDB Cluster in a Docker Container*.

Product version numbering

The version number in Percona XtraDB Cluster releases contains the following components:

- The version of Percona Server that the given Percona XtraDB Cluster release is based on
- The sequence number which represents the Percona XtraDB Cluster built.

For example, version number *8.0.18-9.3* means that this is the third Percona XtraDB Cluster build based on Percona Server 8.0.18-9.

Installing Percona XtraDB Cluster on Debian or Ubuntu

Specific information on the supported platforms, products, and versions is described in Percona Software and Platform Lifecycle.

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

Prerequisites

- You need to have root access on the node where you will be installing Percona XtraDB Cluster (either logged in as a user with root privileges or be able to run commands with `sudo`).
- Make sure that the following ports are not blocked by firewall or used by other software. Percona XtraDB Cluster requires them for communication.
  - 3306
  - 4444
  - 4567
  - 4568
- If you previously had MySQL installed on the server, there might be an AppArmor profile which will prevent Percona XtraDB Cluster nodes from communicating with each other. The best solution is to remove the `apparmor` package entirely:

  ```
  $ sudo apt-get remove apparmor
  ```

  If you need to have AppArmor enabled due to security policies or for other reasons, it is possible to disable or extend the MySQL profile.

Installing from Repository

1. Configure Percona repositories as described in Percona Software Repositories Documentation.
2. Install the Percona XtraDB Cluster server package:

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

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

- `libperconaserverclient21-dev`
- `libperconaserverclient21`
- `percona-xtradb-cluster-client`
- `percona-xtradb-cluster-common`
- `percona-xtradb-cluster-dbg`
- `percona-xtradb-cluster-full`
- `percona-xtradb-cluster-garbd-debug`
- `percona-xtradb-cluster-garbd`
- `percona-xtradb-cluster-server-debug`

- percona-xtradb-cluster-server
- percona-xtradb-cluster-source
- percona-xtradb-cluster-test
- percona-xtradb-cluster

During the installation, you are requested to provide a password for the root user on the database node.

Next Steps

After you install Percona XtraDB Cluster and stop the mysql service, configure the node according to the procedure described in *Configuring Nodes for Write-Set Replication*.

Installing Percona XtraDB Cluster on Red Hat Enterprise Linux and CentOS

Specific information on the supported platforms, products, and versions is described in Percona Software and Platform Lifecycle.

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

Prerequisites

Note: You need to have root access on the node where you will be installing Percona XtraDB Cluster (either logged in as a user with root privileges or be able to run commands with sudo).

Note: Make sure that the following ports are not blocked by firewall or used by other software. Percona XtraDB Cluster requires them for communication.

- 3306
- 4444
- 4567
- 4568

Note: The SELinux security module can constrain access to data for Percona XtraDB Cluster. The best solution is to change the mode from enforcing to permissive by running the following command:

```
setenforce 0
```

This only changes the mode at runtime. To run SELinux in permissive mode after a reboot, set SELINUX=permissive in the /etc/selinux/config configuration file.

Installing from Percona Repository

1. Configure Percona repositories as described in Percona Software Repositories Documentation.
2. Install the Percona XtraDB Cluster packages:

```
$ sudo yum install percona-xtradb-cluster
```

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

- `percona-xtraDB-cluster-shared`
- `percona-xtraDB-cluster-shared-compat`
- `percona-xtradb-cluster-client`
- `percona-xtradb-cluster-debuginfo`
- `percona-xtradb-cluster-devel`
- `percona-xtradb-cluster-garbd`
- `percona-xtradb-cluster-server`
- `percona-xtradb-cluster-test`

3. Start the Percona XtraDB Cluster server:

```
$ sudo service mysql start
```

4. Copy the automatically generated temporary password for the superuser account:

```
$ sudo grep 'temporary password' /var/log/mysqld.log
```

5. Use this password to log in as `root`:

```
$ mysql -u root -p
```

6. Change the password for the superuser account and log out. For example:

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'rootPass';
Query OK, 0 rows affected (0.00 sec)

mysql> exit
Bye
```

7. Stop the `mysql` service:

```
$ sudo service mysql stop
```

Next Steps

After you install Percona XtraDB Cluster and change the superuser account password, configure the node according to the procedure described in *Configuring Nodes for Write-Set Replication*.

Installing Percona XtraDB Cluster from Binary Tarball

Percona provides generic tarballs with all required files and binaries for manual installation.

You can download the appropriate tarball package from https://www.percona.com/downloads/Percona-XtraDB-Cluster-80

In *Percona XtraDB Cluster* 8.0.20-11 and later, the multiple binary tarballs available in the **Linux - Generic** section are replaced with the following:

Name	Type	Description
Percona-XtraDB-Cluster-8.0.xx-relxx-xx-Linux.x86_64.glibc2.12.tar.gz	Full	Contains binaries, libraries, test files, and debug symbols
Percona-XtraDB-Cluster-8.0.xx-relxx-xx-Linux.x86_64.glibc2.12-minimal.tar.gz	Minimal	Contains binaries, and libraries but does not include test files, or debug symbols

Both binary tarballs support all distributions.

For installations before *Percona XtraDB Cluster* 8.0.20-11, the **Linux - Generic** section contains multiple tarballs which are based on the *OpenSSL* library available in your distribution:

- `ssl100`: for Debian prior to 9, and Ubuntu prior to 14.04 versions
- `ssl101`: for CentOS 7, and 8
- `ssl102`: for Debian 9, or 10, and Ubuntu versions starting from 14.04

Note: In CentOS version 7.04 and later, the *OpenSSL* library is `ssl102`.

For example, you can use `curl` as follows:

```
$ curl -O https://www.percona.com/downloads/Percona-XtraDB-Cluster-80/Percona-XtraDB-
↪Cluster/binary/tarball/TARBALL_NAME
```

Be sure to check your system and make sure that the packages are installed that Percona XtraDB Cluster 8.0 depends on.

For Debian or Ubuntu:

```
$ sudo apt-get install -y \
socat libdbd-mysql-perl \
libaio1 libc6 libcurl3 libev4 libgcc1 libgcrypt20 \
libgpg-error0 libssl1.1 libstdc++6 zlib1g libatomic1
```

For Red Hat Enterprise Linux or CentOS:

```
$ sudo yum install -y openssl socat  \
procps-ng chkconfig procps-ng coreutils shadow-utils \
grep libaio libev libcurl perl-DBD-MySQL perl-Digest-MD5 \
libgcc libstdc++ libgcrypt libgpg-error zlib glibc openssl-libs
```

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`
curl	`libcurl-dev`	`libcurl-devel`

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 libcurl-dev
```

- 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 libcurl-devel
```

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
   ```

   Important: 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 `8.0` branch:

   ```
   $ cd percona-xtradb-cluster-galera
   $ git checkout 8.0
   ```

3. Initialize the submodule:

   ```
   $ git submodule init wsrep/src && git submodule update wsrep/src
   $ git submodule init percona-xtradb-cluster-galera && git submodule update␣
   ↪percona-xtradb-cluster-galera
   ```

```
$ cd  percona-xtradb-cluster-galera
$ git submodule init wsrep/src && git submodule update wsrep/src
& git submodule init && git submodule update
$ cd ..
```

4. Run the build script `./build-ps/build-binary.sh`. By default, it attempts building into the current directory. Specify the target output directory, such as `./pxc-build`:

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

When the compilation completes, `pxc-build` contains a tarball, such as `Percona-XtraDB-Cluster-8.0.x86_64.tar.gz`, that you can deploy on your system.

Note: The exact version and release numbers may differ.

Running Percona XtraDB Cluster in a Docker Container

Docker images of Percona XtraDB Cluster are hosted publicly on Docker Hub at https://hub.docker.com/r/percona/percona-xtradb-cluster/.

For more information about using Docker, see the Docker Docs. Make sure that you are using the latest version of Docker. The ones provided via `apt` and `yum` may be outdated and cause errors.

Note: By default, Docker pulls the image from Docker Hub if the image is not available locally.

The image contains only the most essential binaries for Percona XtraDB Cluster to run. Some utilities included in a Percona Server or *MySQL* installation might be missing from the Percona XtraDB Cluster Docker image.

The following procedure describes how to set up a simple 3-node cluster for evaluation and testing purposes. Do not use these instructions in a production environment because the MySQL certificates generated in this procedure are self-signed. For a production environment, you should generate and store the certificates to be used by Docker.

In this procedure, all of the nodes run Percona XtraDB Cluster 8.0 in separate containers on one host:

1. Create a ~/pxc-docker-test/config directory.
2. Create a custom.cnf file with the following contents, and place the file in the new directory:

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

[client]
ssl-ca = /cert/ca.pem
ssl-cert = /cert/client-cert.pem
ssl-key = /cert/client-key.pem

[sst]
encrypt = 4
ssl-ca = /cert/ca.pem
ssl-cert = /cert/server-cert.pem
ssl-key = /cert/server-key.pem
```

3. Create a cert directory and generate self-signed SSL certificates on the host node:

```
$ mkdir -m 777 -p ~/pxc-docker-test/cert
docker run --name pxc-cert --rm -v ~/pxc-docker-test/cert:/cert
percona/percona-xtradb-cluster:8.0 mysql_ssl_rsa_setup -d /cert
```

4. Create a Docker network:

```
docker network create pxc-network
```

5. Bootstrap the cluster (create the first node):

```
docker run -d \
  -e MYSQL_ROOT_PASSWORD=test1234# \
  -e CLUSTER_NAME=pxc-cluster1 \
  --name=pxc-node1 \
  --net=pxc-network \
  -v ~/pxc-docker-test/config:/etc/percona-xtradb-cluster.conf.d \
  percona/percona-xtradb-cluster:8.0
```

6. Join the second node:

```
docker run -d \
  -e MYSQL_ROOT_PASSWORD=test1234# \
  -e CLUSTER_NAME=pxc-cluster1 \
  -e CLUSTER_JOIN=pxc-node1 \
  --name=pxc-node2 \
  --net=pxc-network \
  -v ~/pxc-docker-test/cert:/cert \
  -v ~/pxc-docker-test/config:/etc/percona-xtradb-cluster.conf.d \
  percona/percona-xtradb-cluster:8.0
```

7. Join the third node:

```
docker run -d \
  -e MYSQL_ROOT_PASSWORD=test1234# \
  -e CLUSTER_NAME=pxc-cluster1 \
  -e CLUSTER_JOIN=pxc-node1 \
  --name=pxc-node3 \
  --net=pxc-network \
  -v ~/pxc-docker-test/cert:/cert \
  -v ~/pxc-docker-test/config:/etc/percona-xtradb-cluster.conf.d \
  percona/percona-xtradb-cluster:8.0
```

To verify the cluster is available, do the following:

1. Access the MySQL client. For example, on the first node:

```
$ sudo docker exec -it pxc-node1 /usr/bin/mysql -uroot -ptest1234#
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12
...
You are enforcing ssl connection via unix socket. Please consider
switching ssl off as it does not make connection via unix socket
any more secure

mysql>
```

2. View the wsrep status variables:

```
mysql> show status like 'wsrep%';
+------------------------------+-------------------------------------------------+
| Variable_name                | Value                                           |
+------------------------------+-------------------------------------------------+
| wsrep_local_state_uuid       | 625318e2-9e1c-11e7-9d07-aee70d98d8ac            |
...
| wsrep_local_state_comment    | Synced                                          |
...
| wsrep_incoming_addresses     | 172.18.0.2:3306,172.18.0.3:3306,172.18.0.4:3306 |
...
| wsrep_cluster_conf_id        | 3                                               |
| wsrep_cluster_size           | 3                                               |
| wsrep_cluster_state_uuid     | 625318e2-9e1c-11e7-9d07-aee70d98d8ac            |
| wsrep_cluster_status         | Primary                                         |
| wsrep_connected              | ON                                              |
...
| wsrep_ready                  | ON                                              |
+------------------------------+-------------------------------------------------+
59 rows in set (0.02 sec)
```

See also:

Creating SSL and RSA Certificates and Keys How to create the files required for SSL and RSA support in MySQL.

CHAPTER

FIVE

CONFIGURING NODES FOR WRITE-SET REPLICATION

After installing Percona XtraDB Cluster on each node, you need to configure the cluster. In this section, we will demonstrate how to configure a three node cluster:

Node	Host	IP
Node 1	pxc1	192.168.70.61
Node 2	pxc2	192.168.70.62
Node 3	pxc3	192.168.70.63

1. Stop the Percona XtraDB Cluster server. After the installation completes the server is not started. You need this step if you have started the server manually.

   ```
   $ sudo service mysql stop
   ```

2. Edit the configuration file of the first node to provide the cluster settings.

   If you use Debian or Ubuntu, edit `/etc/mysql/mysql.conf.d/mysqld.cnf`:

   ```
   wsrep_provider=/usr/lib/galera4/libgalera_smm.so
   wsrep_cluster_name=pxc-cluster
   wsrep_cluster_address=gcomm://192.168.70.61,192.168.70.62,192.168.70.63
   ```

   If you use Red Hat or CentOS, edit `/etc/my.cnf`. Note that on these systems you set the wsrep_provider option to a different value:

   ```
   wsrep_provider=/usr/lib64/galera4/libgalera_smm.so
   wsrep_cluster_name=pxc-cluster
   wsrep_cluster_address=gcomm://192.168.70.61,192.168.70.62,192.168.70.63
   ```

3. Configure *node 1*.

   ```
   wsrep_node_name=pxc1
   wsrep_node_address=192.168.70.61
   pxc_strict_mode=ENFORCING
   ```

4. Set up *node 2* and *node 3* in the same way: Stop the server and update the configuration file applicable to your system. All settings are the same except for *wsrep_node_name* and *wsrep_node_address*.

   For node 2 wsrep_node_name=pxc2 wsrep_node_address=192.168.70.62

   For node 3 wsrep_node_name=pxc3 wsrep_node_address=192.168.70.63

5. Set up the traffic encryption settings. Each node of the cluster must use the same SSL certificates.

   ```
   [mysqld]
   wsrep_provider_options="socket.ssl_key=server-key.pem;socket.ssl_cert=server-cert.
   ↪pem;socket.ssl_ca=ca.pem"
   ```

```
[sst]
encrypt=4
ssl-key=server-key.pem
ssl-ca=ca.pem
ssl-cert=server-cert.pem
```

Important: In Percona XtraDB Cluster 8.0, the *Encrypting Replication Traffic* is enabled by default (via the `pxc-encrypt-cluster-traffic` variable).

The replication traffic encryption cannot be enabled on a running cluster. If it was disabled before the cluster was bootstrapped, the cluster must to stopped. Then set up the encryption, and bootstrap (see *Bootstrapping the First Node*) again.

See also:

More information about the security settings in Percona XtraDB Cluster

- *Security Basics*
- *Encrypting PXC Traffic*
- *SSL Automatic Configuration*

Template of the configuration file

Here is an example of a full configuration file installed on CentOS to `/etc/my.cnf`.

```
# Template my.cnf for PXC
# Edit to your requirements.
[client]
socket=/var/lib/mysql/mysql.sock
[mysqld]
server-id=1
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
# Binary log expiration period is 604800 seconds, which equals 7 days
binlog_expire_logs_seconds=604800
######## wsrep ###############
# Path to Galera library
wsrep_provider=/usr/lib64/galera4/libgalera_smm.so
# Cluster connection URL contains IPs of nodes
#If no IP is found, this implies that a new cluster needs to be created,
#in order to do that you need to bootstrap this node
wsrep_cluster_address=gcomm://
# In order for Galera to work correctly binlog format should be ROW
binlog_format=ROW
# Slave thread to use
wsrep_slave_threads=8
wsrep_log_conflicts
# This changes how InnoDB autoincrement locks are managed and is a requirement for␣
↪Galera
innodb_autoinc_lock_mode=2
```

```
# Node IP address
#wsrep_node_address=192.168.70.63
# Cluster name
wsrep_cluster_name=pxc-cluster
#If wsrep_node_name is not specified,  then system hostname will be used
wsrep_node_name=pxc-cluster-node-1
#pxc_strict_mode allowed values: DISABLED,PERMISSIVE,ENFORCING,MASTER
pxc_strict_mode=ENFORCING
# SST method
wsrep_sst_method=xtrabackup-v2
```

Next Steps: Bootstrap the first node

After you configure all your nodes, initialize Percona XtraDB Cluster by bootstrapping the first node according to the procedure described in *Bootstrapping the First Node*.

Essential configuration variables

wsrep_provider

Specify the path to the Galera library. The location depends on the distribution:

- Debian and Ubuntu: `/usr/lib/galera4/libgalera_smm.so`
- Red Hat and CentOS: `/usr/lib64/galera4/libgalera_smm.so`

wsrep_cluster_name

Specify the logical name for your cluster. It must be the same for all nodes in your cluster.

wsrep_cluster_address

Specify the IP addresses of nodes in your cluster. At least one is required for a node to join the cluster, but it is recommended to list addresses of all nodes. This way if the first node in the list is not available, the joining node can use other addresses.

Note: No addresses are required for the initial node in the cluster. However, it is recommended to specify them and *properly bootstrap the first node*. This will ensure that the node is able to rejoin the cluster if it goes down in the future.

---

wsrep_node_name

Specify the logical name for each individual node. If this variable is not specified, the host name will be used.

wsrep_node_address

Specify the IP address of this particular node.

wsrep_sst_method

By default, Percona XtraDB Cluster uses Percona XtraBackup for *State Snapshot Transfer*. `xtrabackup-v2` is the only supported option for this variable. This method requires a user for SST to be set up on the initial node.

pxc_strict_mode

PXC Strict Mode is enabled by default and set to `ENFORCING`, which blocks the use of experimental and unsupported features in Percona XtraDB Cluster.

`binlog_format`

Galera supports only row-level replication, so set `binlog_format=ROW`.

`default_storage_engine`

Galera fully supports only the InnoDB storage engine. It will not work correctly with MyISAM or any other non-transactional storage engines. Set this variable to `default_storage_engine=InnoDB`.

`innodb_autoinc_lock_mode`

Galera supports only interleaved (`2`) lock mode for InnoDB. Setting the traditional (`0`) or consecutive (`1`) lock mode can cause replication to fail due to unresolved deadlocks. Set this variable to `innodb_autoinc_lock_mode=2`.

CHAPTER

SIX

BOOTSTRAPPING THE FIRST NODE

After you *configure all PXC nodes*, initialize the cluster by bootstrapping the first node. The initial node must contain all the data that you want to be replicated to other nodes.

Bootstrapping implies starting the first node without any known cluster addresses: if the *wsrep_cluster_address* variable is empty, Percona XtraDB Cluster assumes that this is the first node and initializes the cluster.

Instead of changing the configuration, start the first node using the following command:

```
[root@pxc1 ~]# systemctl start mysql@bootstrap.service
```

When you start the node using the previous command, it runs in bootstrap mode with `wsrep_cluster_address=gcomm://`. This tells the node to initialize the cluster with *wsrep_cluster_conf_id* set to `1`. After you *add other nodes* to the cluster, you can then restart this node as normal, and it will use standard configuration again.

To make sure that the cluster has been initialized, run the following:

```
mysql@pxc1> 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)
```

The previous output shows that the cluster size is 1 node, it is the primary component, the node is in the `Synced` state, it is fully connected and ready for write-set replication.

Different from previous version

The variable `wsrep_sst_auth` has been removed. Percona XtraDB Cluster 8.0 automatically creates the system user *mysql.pxc.internal.session*. During *SST*, the user *mysql.pxc.sst.user* and the role *mysql.pxc.sst.role* are created on the donor node.

See also:

***Percona* Blog Post: Percona XtraBackup 8.0 New Feature: wsrep_sst_auth removal** https://www.percona.com/blog/2019/10/03/percona-xtradb-cluster-8-0-new-feature-wsrep_sst_auth-removal/

Next Steps

After initializing the cluster, you can *add other nodes*.

CHAPTER

SEVEN

ADDING NODES TO CLUSTER

New nodes that are *properly configured* are provisioned automatically. When you start a node with the address of at least one other running node in the `wsrep_cluster_address` variable, this node automatically joins and synchronizes with the cluster.

Note: Any existing data and configuration will be overwritten to match the data and configuration of the DONOR node. Do not join several nodes at the same time to avoid overhead due to large amounts of traffic when a new node joins.

Percona XtraDB Cluster uses Percona XtraBackup for *State Snapshot Transfer* and the `wsrep_sst_method` variable is always set to `xtrabackup-v2`.

Starting the Second Node

Start the second node using the following command:

```
[root@pxc2 ~]# systemctl start mysql
```

After the server starts, it receives *SST* automatically.

To check the status of the second node, run the following:

```
mysql@pxc2> show status like 'wsrep%';
+----------------------------------+----------------------------------------------------
↪+
| Variable_name                    | Value                                              ␣
↪|
+----------------------------------+----------------------------------------------------
↪+
| wsrep_local_state_uuid           | a08247c1-5807-11ea-b285-e3a50c8efb41               ␣
↪|
| ...                              | ...                                                ␣
↪|
| wsrep_local_state                | 4                                                  ␣
↪|
| wsrep_local_state_comment        | Synced                                             ␣
↪|
| ...                              |                                                    ␣
↪|
| wsrep_cluster_size               | 2                                                  ␣
↪|
| wsrep_cluster_status             | Primary                                            ␣
↪|
```

```
| wsrep_connected                  | ON                                          |
| ...                              | ...                                         |
| wsrep_provider_capabilities      | :MULTI_MASTER:CERTIFICATION: ...            |
| wsrep_provider_name              | Galera                                      |
| wsrep_provider_vendor            | Codership Oy <info@codership.com>           |
| wsrep_provider_version           | 4.3(r752664d)                               |
| wsrep_ready                      | ON                                          |
| ...                              | ...                                         |
+----------------------------------+---------------------------------------------+
75 rows in set (0.00 sec)
```

The output of `SHOW STATUS` shows that the new node has been successfully added to the cluster. The cluster size is now 2 nodes, it is the primary component, and it is fully connected and ready to receive write-set replication.

If the state of the second node is `Synced` as in the previous example, then the node received full *SST* is synchronized with the cluster, and you can proceed to add the next node.

Note: If the state of the node is `Joiner`, it means that SST hasn't finished. Do not add new nodes until all others are in `Synced` state.

Starting the Third Node

To add the third node, start it as usual:

```
[root@pxc3 ~]# systemctl start mysql
```

To check the status of the third node, run the following:

```
mysql@pxc3> 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)
```

Previous output shows that the new node has been successfully added to the cluster. Cluster size is now 3 nodes, it is the primary component, and it is fully connected and ready to receive write-set replication.

Next Steps

When you add all nodes to the cluster, you can *verify replication* by running queries and manipulating data on nodes to see if these changes are synchronized accross the cluster.

CHAPTER

EIGHT

VERIFYING REPLICATION

Use the following procedure to verify replication by creating a new database on the second node, creating a table for that database on the third node, and adding 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 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)
```

Next Steps

- Consider installing ProxySQL on client nodes for efficient workload management across the cluster without any changes to the applications that generate queries. This is the recommended high-availability solution for Percona XtraDB Cluster.

  For more information, see *Load balancing with ProxySQL*.

- Percona Monitoring and Management is the best choice for managing and monitoring Percona XtraDB Cluster performance. It provides visibility for the cluster and enables efficient troubleshooting.

Part III

Features

CHAPTER

NINE

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. Percona XtraDB Cluster uses **`xtrabackup`** for SST.

  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.

CHAPTER

TEN

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 GLOBAL 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.

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.

- *Group replication*
- *Storage engine*
- *MyISAM replication*
- *Binary log format*
- *Tables without primary keys*
- *Log output*
- *Explicit table locking*
- *Auto-increment lock mode*
- *Combining schema and data changes in a single statement*
- *Discarding and Importing Tablespaces*

Group replication

Group replication is a feature of *MySQL* that provides distributed state machine replication with strong coordination between servers. It is implemented as a plugin which, if activated, may conflict with Percona XtraDB Cluster. Group replication cannot be activated to run alongside Percona XtraDB Cluster. However, you can migrate to Percona XtraDB Cluster from the environment that uses group replication.

For the strict mode to work correctly, make sure that the group replication plugin is *not active*. In fact, if *pxc_strict_mode* is set to *ENFORCING* or *MASTER*, the server will stop with an error:

Error message with `pxc_strict_mode` set to *ENFORCING* or *MASTER*

```
Group replication cannot be used with |pxc| in strict mode.
```

If *pxc_strict_mode* is set to `DISABLED` you can use group replication at your own risk. Setting *pxc_strict_mode* to `PERMISSIVE` will result in a warning.

Warning message with `pxc_strict_mode` set to *PERMISSIVE*

```
Using group replication with |pxc| is only supported for migration. Please
make sure that group replication is turned off once all data is migrated to
|pxc|.
```

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. In 8.0, setting the binlog_format[1] variable to anything but ROW at startup or runtime is not allowed regardless of the value of the *pxc_strict_mode* variable.

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

[1] http://dev.mysql.com/doc/refman/8.0/en/replication-options-binary-log.html#sysvar_binlog_format
[2] http://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html#sysvar_log_output

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()`
- `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.

[3] http://dev.mysql.com/doc/refman/8.0/en/innodb-parameters.html#sysvar_innodb_autoinc_lock_mode

`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

With strict mode set to `ENFORCING`, Percona XtraDB Cluster does not support CTAS (CREATE TABLE ... AS SELECT) statements, because they combine both schema and data changes. Note that tables in the SELECT clause should be present on all replication nodes.

With strict mode set to `PERMISSIVE` or `DISABLED`, CTAS statements are replicated using the TOI (Total Order Isolation) method to ensure consistency. In Percona XtraDB Cluster 5.7, CTAS statements were replicated using DML write-sets when strict mode was set to `PERMISSIVE` or `DISABLED`.

Important: MyISAM tables are created and loaded even if *wsrep_replicate_myisam* equals to 1. Percona XtraDB Cluster does not recommend using the *MyISAM* storage engine. The support for *MyISAM* is experimental and may be removed in a future release.

See also:

***MySQL* Bug System: XID inconsistency on master-slave with CTAS** https://bugs.mysql.com/bug.php?id=93948

Depending on the strict mode selected, the following happens:

Mode	Behavior
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 a CTAS operation is performed.
ENFORCING	At startup, no validation is performed. At runtime, any CTAS operation is denied and an error is logged.

Important: Although CTAS operations for temporary tables are permitted even in `STRICT` mode, temporary tables should not be used as *source* tables in CTAS operations due to the fact that temporary tables are not present on all nodes.

If `node-1` has a temporary and a non-temporary table with the same name, CTAS on `node-1` will use temporary and CTAS on `node-2` will use the non-temporary table resulting in a data level inconsistency.

Discarding and Importing Tablespaces

`DISCARD TABLESPACE` and `IMPORT TABLESPACE` are not replicated using TOI. This can lead to data inconsistency if executed on only one node.

Depending on the strict mode selected, 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 you discard or import a tablespace.

`ENFORCING`

At startup, no validation is performed.

At runtime, discarding or importing a tablespace is denied and an error is logged.

References

Part IV

PXC Security

CHAPTER

ELEVEN

SECURITY BASICS

By default, Percona XtraDB Cluster does not provide any protection for stored data. There are several considerations to take into account for securing Percona XtraDB Cluster:

- *Securing the Network*

  Anyone with access to your network can connect to any Percona XtraDB Cluster node either as a client or as another node joining the cluster. You should consider restricting access using VPN and filter traffic on ports used by Percona XtraDB Cluster.

- *Encrypting PXC Traffic*

  Unencrypted traffic can potentially be viewed by anyone monitoring your network. In Percona XtraDB Cluster 8.0 traffic encryption is enabled by default.

- Data-at-rest encryption

  Percona XtraDB Cluster supports tablespace encryption to provide at-rest encryption for physical tablespace data files.

  For more information, see the following blog post:

  - MySQL Data at Rest Encryption

Security Modules

Most modern disributions include special security modules that control access to resources for users and applications. By default, these modules will most likely constrain communication between Percona XtraDB Cluster nodes.

The easiest solution is to disable or remove such programs, however, this is not recommended for production environments. You should instead create necessary security policies for Percona XtraDB Cluster.

SELinux

SELinux is usually enabled by default in Red Hat Enterprise Linux and derivatives (including CentOS). During installation and configuration, you can set the mode to `permissive` by running the following command:

```
setenforce 0
```

Note: This only changes the mode at runtime. To run SELinux in permissive mode after a reboot, set `SELINUX=permissive` in the `/etc/selinux/config` configuration file.

To use SELinux with Percona XtraDB Cluster, you need to create an access policy. For more information, see SELinux and MySQL.

AppArmor

AppArmor is included in Debian and Ubuntu. During installation and configuration, you can disable AppArmor for `mysqld`:

1. Create the following symbolic link:

```
$ sudo ln -s /etc/apparmor.d/usr /etc/apparmor.d/disable/.sbin.mysqld
```

2. Restart AppArmor:

```
$ sudo service apparmor restart
```

Note: If your system uses `systemd`, run the following command instead:

```
$ sudo systemctl restart apparmor
```

To use AppArmor with Percona XtraDB Cluster, you need to create or extend the MySQL profile. For more information, see AppArmor and MySQL.

CHAPTER

TWELVE

SECURING THE NETWORK

By default, anyone with access to your network can connect to any Percona XtraDB Cluster node either as a client or as another node joining the cluster. This could potentially let them query your data or get a complete copy of it.

In general, it is a good idea to disable all remote connections to Percona XtraDB Cluster nodes. If you require clients or nodes from outside of your network to connect, you can set up a VPN (virtual private network) for this purpose.

Firewall Configuration

A firewall can let you filter Percona XtraDB Cluster traffic based on the clients and nodes that you trust.

By default, Percona XtraDB Cluster nodes use the following ports:

- 3306 is used for MySQL client connections and *SST* (State Snapshot Transfer) via `mysqldump`.
- 4444 is used for *SST* via *Percona XtraBackup*.
- 4567 is used for write-set replication traffic (over TCP) and multicast replication (over TCP and UDP).
- 4568 is used for *IST* (Incremental State Transfer).

Ideally you want to make sure that these ports on each node are accessed only from trusted IP addresses. You can implement packet filtering using `iptables`, `firewalld`, `pf`, or any other firewall of your choice.

Using iptables

To restrict access to Percona XtraDB Cluster ports using `iptables`, you need to append new rules to the `INPUT` chain on the filter table. In the following example, the trusted range of IP addresses is 192.168.0.1/24. It is assumed that only Percona XtraDB Cluster nodes and clients will connect from these IPs. To enable packet filtering, run the commands as root on each Percona XtraDB Cluster node.

```
# iptables --append INPUT --in-interface eth0 \
   --protocol tcp --match tcp --dport 3306 \
   --source 192.168.0.1/24 --jump ACCEPT
# iptables --append INPUT --in-interface eth0 \
   --protocol tcp --match tcp --dport 4444 \
   --source 192.168.0.1/24 --jump ACCEPT
# iptables --append INPUT --in-interface eth0 \
   --protocol tcp --match tcp --dport 4567 \
   --source 192.168.0.1/24 --jump ACCEPT
# iptables --append INPUT --in-interface eth0 \
   --protocol tcp --match tcp --dport 4568 \
   --source 192.168.0.1/24 --jump ACCEPT
# iptables --append INPUT --in-interface eth0 \
```

```
   --protocol udp --match udp --dport 4567 \
   --source 192.168.0.1/24 --jump ACCEPT
```

Note: The last one opens port 4567 for multicast replication over UDP.

If the trusted IPs are not in sequence, you will need to run these commands for each address on each node. In this case, you can consider to open all ports between trusted hosts. This is a little bit less secure, but reduces the amount of commands. For example, if you have three Percona XtraDB Cluster nodes, you can run the following commands on each one:

```
# iptables --append INPUT --protocol tcp \
    --source 64.57.102.34 --jump ACCEPT
# iptables --append INPUT --protocol tcp \
    --source 193.166.3.20  --jump ACCEPT
# iptables --append INPUT --protocol tcp \
    --source 193.125.4.10  --jump ACCEPT
```

Running the previous commands will allow TCP connections from the IP addresses of the other Percona XtraDB Cluster nodes.

Note: The changes that you make in `iptables` are not persistent unless you save the packet filtering state:

```
# service save iptables
```

For distributions that use `systemd`, you need to save the current packet filtering rules to the path where `iptables` reads from when it starts. This path can vary by distribution, but it is usually in the `/etc` directory. For example:

- `/etc/sysconfig/iptables`
- `/etc/iptables/iptables.rules`

Use `iptables-save` to update the file:

```
# iptables-save > /etc/sysconfig/iptables
```

CHAPTER

THIRTEEN

ENCRYPTING PXC TRAFFIC

There are two kinds of traffic in Percona XtraDB Cluster:

1. Client-Server traffic (the one between client applications and cluster nodes),
2. Replication traffic, that includes *SST*, *IST*, write-set replication, and various service messages.

Percona XtraDB Cluster supports encryption for all types of traffic. Replication traffic encryption can be configured either automatically or manually.

- *Encrypting Client-Server Communication*
- *Encrypting Replication Traffic*
- *SSL Automatic Configuration*
- *SSL Manual Configuration*
- *Generating Keys and Certificates Manually*

Encrypting Client-Server Communication

Percona XtraDB Cluster uses the underlying MySQL encryption mechanism to secure communication between client applications and cluster nodes.

MySQL generates default key and certificate files and places them in the data directory. You can override auto-generated files with manually created ones, as described in the section *Generating Keys and Certificates Manually*.

The auto-generated files are suitable for automatic SSL configuration, but you should use the same key and certificate files on all nodes.

Specify the following settings in the `my.cnf` configuration file for each node:

```
[mysqld]
ssl-ca=/etc/mysql/certs/ca.pem
ssl-cert=/etc/mysql/certs/server-cert.pem
ssl-key=/etc/mysql/certs/server-key.pem

[client]
ssl-ca=/etc/mysql/certs/ca.pem
ssl-cert=/etc/mysql/certs/client-cert.pem
ssl-key=/etc/mysql/certs/client-key.pem
```

After it is restarted, the node uses these files to encrypt communication with clients. MySQL clients require only the second part of the configuration to communicate with cluster nodes.

MySQL generates the default key and certificate files and places them in the data directory. You can either use them or generate new certificates. For generation of new certificate please refer to *Generating Keys and Certificates Manually* section.

Encrypting Replication Traffic

Replication traffic refers to the inter-node traffic which includes the *SST* traffic, *IST* traffic, and replication traffic.

The traffic of each type is transferred via a different channel, and so it is important to configure secure channels for all 3 variants to completely secure the replication traffic.

Percona XtraDB Cluster supports a single configuration option which helps to secure the complete replication traffic, and is often referred to as *SSL Automatic Configuration*. You can also configure the security of each channel by specifying independent parameters.

SSL Automatic Configuration

The automatic configuration of the SSL encryption needs a key and certificate files. MySQL generates a default key and certificate files and places them in the data directory.

Important: It is important that your cluster use the same SSL certificates on all nodes.

Enabling `pxc-encrypt-cluster-traffic`

Percona XtraDB Cluster includes the `pxc-encrypt-cluster-traffic` variable that enables the configuration of the SSL encryption thereby encrypting *SST*, *IST*, and replication traffic.

By default, `pxc-encrypt-cluster-traffic` is enabled thereby using a secured channel for replication. This variable is not dynamic and so it cannot be changed at runtime.

Enabled, `pxc-encrypt-cluster-traffic` has the effect of applying the following settings: *encrypt*, `ssl_key`, `ssl-ca`, `ssl-cert`.

Setting `pxc-encrypt-cluster-traffic=ON` has the effect of applying the following settings in the `my.cnf` configuration file:

```
[mysqld]
wsrep_provider_options="socket.ssl_key=server-key.pem;socket.ssl_cert=server-cert.pem;
↪socket.ssl_ca=ca.pem"

[sst]
encrypt=4
ssl-key=server-key.pem
ssl-ca=ca.pem
ssl-cert=server-cert.pem
```

For *wsrep_provider_options*, only the mentioned options are affected (`socket.ssl_key`, `socket.ssl_cert`, and `socket.ssl_ca`), the rest is not modified.

Important: Disabling `pxc-encrypt-cluster-traffic`

The default value of `pxc-encrypt-cluster-traffic` helps improve the security of your system.

When `pxc-encrypt-cluster-traffic` is not enabled, anyone with the access to your network can connect to any Percona XtraDB Cluster node either as a client or as another node joining the cluster. This potentially lets them query your data or get a complete copy of it.

If you must disable `pxc-encrypt-cluster-traffic`, you need to stop the cluster and update *[mysqld]* section of the configuration file: `pxc-encrypt-cluster-traffic=OFF` of each node. Then, restart the cluster.

The automatic configuration of the SSL encryption needs key and certificate files. *MySQL* generates default key and certificate files and places them in data directory. These auto-generated files are suitable for automatic SSL configuration, but *you should use the same key and certificate files on all nodes*. Also you can override auto-generated files with manually created ones, as covered in *Generating Keys and Certificates Manually*.

The necessary key and certificate files are first searched at the `ssl-ca`, `ssl-cert`, and `ssl-key` options under `[mysqld]`. If these options are not set, the data directory is searched for `ca.pem`, `server-cert.pem`, and `server-key.pem` files.

Note: The `[sst]` section is not searched.

If all three files are found, they are used to configure encryption. If any of the files is missing, a fatal error is generated.

SSL Manual Configuration

If user wants to enable encryption for specific channel only or use different certificates or other mix-match, then user can opt for manual configuration. This helps to provide more flexibility to end-users.

To enable encryption manually, the location of the required key and certificate files shoud be specified in the Percona XtraDB Cluster configuration. If you do not have the necessary files, see *Generating Keys and Certificates Manually*.

Note: Encryption settings are not dynamic. To enable it on a running cluster, you need to restart the entire cluster.

There are three aspects of Percona XtraDB Cluster operation, where you can enable encryption:

- *Encrypting SST Traffic*

  This refers to *SST* traffic during full data copy from one cluster node (donor) to the joining node (joiner).

- *Encrypting Replication Traffic*
- *Encrypting IST Traffic*

  This refers to all internal Percona XtraDB Cluster communication, such as, write-set replication, *IST*, and various service messages.

Encrypting 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*.

Note: If `keyring_file` plugin is used, then SST encryption is mandatory: when copying encrypted data via SST, the keyring must be sent over with the files for decryption. In this case following options are to be set in `my.cnf` on all nodes:

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

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

The only available SST method is `xtrabackup-v2` which uses *Percona XtraBackup*.

xtrabackup

This is the only available SST method (the *`wsrep_sst_method`* is always set to `xtrabackup-v2`), 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:

- `encrypt=0` is the default value, meaning that encryption is disabled.
- `encrypt=4` enables encryption based on key and certificate files generated with OpenSSL. For more information, see *Generating Keys and Certificates Manually*.

  To enable encryption for SST using XtraBackup, specify the location of the keys and certificate files in the each node's configuration under `[sst]`:

  ```
  [sst]
  encrypt=4
  ssl-ca=/etc/mysql/certs/ca.pem
  ssl-cert=/etc/mysql/certs/server-cert.pem
  ssl-key=/etc/mysql/certs/server-key.pem
  ```

Note: SSL clients require DH parameters to be at least 1024 bits, due to the logjam vulnerability. However, versions of `socat` earlier than 1.7.3 use 512-bit parameters. If a `dhparams.pem` file of required length is not found during SST in the data directory, it is generated with 2048 bits, which can take several minutes. To avoid this delay, create the `dhparams.pem` file manually and place it in the data directory before joining the node to the cluster:

```
openssl dhparam -out /path/to/datadir/dhparams.pem 2048
```

For more information, see this blog post.

Encrypting Replication/IST Traffic

Replication traffic refers to the following:

- Write-set replication which is the main workload of Percona XtraDB Cluster (replicating transactions that execute on one node to all other nodes).
- Incremental State Transfer (*IST*) which is copying only missing transactions from DONOR to JOINER node.
- Service messages which ensure that all nodes are synchronized.

All this traffic is transferred via the same underlying communication channel (`gcomm`). Securing this channel will ensure that *IST* traffic, write-set replication, and service messages are encrypted. (For IST, a separate channel is configured using the same configuration parameters, so 2 sections are described together).

To enable encryption for all these processes, define the paths to the key, certificate and certificate authority files using the following *wsrep provider options*:

- *socket.ssl_ca*
- *socket.ssl_cert*
- *socket.ssl_key*

To set these options, use the *wsrep_provider_options* variable in the configuration file:

```
wsrep_provider_options="socket.ssl=yes;socket.ssl_ca=/etc/mysql/certs/ca.pem;socket.
↪ssl_cert=/etc/mysql/certs/server-cert.pem;socket.ssl_key=/etc/mysql/certs/server-
↪key.pem"
```

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

Check :upgrade-certificate: section on how to upgrade existing certificates.

Generating Keys and Certificates Manually

As mentioned above, *MySQL* generates default key and certificate files and places them in the data directory. If you want to override these certificates, the following new sets of files can be generated:

- *Certificate Authority (CA) key and certificate* to sign the server and client certificates.
- *Server key and certificate* to secure database server activity and write-set replication traffic.
- *Client key and certificate* to secure client communication traffic.

These files should be generated using OpenSSL.

Note: The `Common Name` value used for the server and client keys and certificates must differ from the value used for the CA certificate.

Generating CA Key and Certificate

The Certificate Authority is used to verify the signature on certificates.

1. Generate the CA key file:

```
$ openssl genrsa 2048 > ca-key.pem
```

2. Generate the CA certificate file:

```
$ openssl req -new -x509 -nodes -days 3600
    -key ca-key.pem -out ca.pem
```

Generating Server Key and Certificate

1. Generate the server key file:

```
$ openssl req -newkey rsa:2048 -days 3600 \
    -nodes -keyout server-key.pem -out server-req.pem
```

2. Remove the passphrase:

```
$ openssl rsa -in server-key.pem -out server-key.pem
```

3. Generate the server certificate file:

```
$ openssl x509 -req -in server-req.pem -days 3600 \
    -CA ca.pem -CAkey ca-key.pem -set_serial 01 \
    -out server-cert.pem
```

Generating Client Key and Certificate

1. Generate the client key file:

```
$ openssl req -newkey rsa:2048 -days 3600 \
    -nodes -keyout client-key.pem -out client-req.pem
```

2. Remove the passphrase:

```
$ openssl rsa -in client-key.pem -out client-key.pem
```

3. Generate the client certificate file:

```
$ openssl x509 -req -in client-req.pem -days 3600 \
    -CA ca.pem -CAkey ca-key.pem -set_serial 01 \
    -out client-cert.pem
```

Verifying Certificates

To verify that the server and client certificates are correctly signed by the CA certificate, run the following command:

```
$ openssl verify -CAfile ca.pem server-cert.pem client-cert.pem
```

If the verification is successful, you should see the following output:

```
server-cert.pem: OK
client-cert.pem: OK
```

Failed validation caused by matching CN

Sometimes, an SSL configuration may fail if the certificate and the CA files contain the same CN (SSL Certificate Common Name).

To check if this is the case run `openssl` command as follows and verify that the **CN** field differs for the *Subject* and *Issuer* lines.

```
$ openssl x509 -in server-cert.pem -text -noout
```

Incorrect values

```
Certificate:
Data:
Version: 1 (0x0)
Serial Number: 1 (0x1)
Signature Algorithm: sha256WithRSAEncryption
Issuer: CN=www.percona.com, O=Database Performance., C=US
...
Subject: CN=www.percona.com, O=Database Performance., C=AU
...
```

To obtain a more compact output run `openssl` specifying *-subject* and *-issuer* parameters:

```
$ openssl x509 -in server-cert.pem -subject -issuer -noout
```

Output

```
subject= /CN=www.percona.com/O=Database Performance./C=AU
issuer= /CN=www.percona.com/O=Database Performance./C=US
```

Deploying Keys and Certificates

Use a secure method (for example, `scp` or `sftp`) to send the key and certificate files to each node. Place them under the `/etc/mysql/certs/` directory or similar location where you can find them later.

Note: Make sure that this directory is protected with proper permissions. Most likely, you only want to give read permissions to the user running `mysqld`.

The following files are required:

- Certificate Authority certificate file (`ca.pem`)

  This file is used to verify signatures.

- Server key and certificate files (`server-key.pem` and `server-cert.pem`)

  These files are used to secure database server activity and write-set replication traffic.

- Client key and certificate files (`client-key.pem` and `client-cert.pem`)

  These files are required only if the node should act as a MySQL client. For example, if you are planning to perform SST using `mysqldump`.

Note: *Upgrading Certificates* subsection covers the details on upgrading certificates, if necessary.

Upgrading Certificates

The following procedure shows how to upgrade certificates used for securing replication traffic when there are two nodes in the cluster.

1. Restart the first node with the *socket.ssl_ca* option set to a combination of the the old and new 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 as follows:

   ```
   wsrep_provider_options="socket.ssl=yes;socket.ssl_ca=/etc/mysql/certs/upgrade-ca.
   ↪pem;socket.ssl_cert=/etc/mysql/certs/old-cert.pem;socket.ssl_key=/etc/mysql/
   ↪certs/old-key.pem"
   ```

2. Restart the second node with the *socket.ssl_ca*, *socket.ssl_cert*, and *socket.ssl_key* options set to the corresponding new certificate files.

   ```
   wsrep_provider_options="socket.ssl=yes;socket.ssl_ca=/etc/mysql/certs/new-ca.pem;
   ↪socket.ssl_cert=/etc/mysql/certs/new-cert.pem;socket.ssl_key=/etc/mysql/certs/
   ↪new-key.pem"
   ```

3. Restart the first node with the new certificate files, as in the previous step.

4. You can remove the old certificate files.

Part V

User's Manual

CHAPTER

FOURTEEN

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.

Percona XtraDB Cluster enables SST (State Snapshot Transfer) via **xtrabackup**.

Xtrabackup SST 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.

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.

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.

The *datadir* needs to be specified in the server configuration file `my.cnf`, otherwise the transfer process will fail.

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

SST for tables with tablespaces that are not in the data directory

For example:

```
CREATE TABLE t1 (c1 INT PRIMARY KEY) DATA DIRECTORY = '/alternative/directory';
```

SST using Percona XtraBackup

XtraBackup will restore the table to the same location on the joiner node. If the target directory does not exist, it will be created. If the target file already exists, an error will be returned, because XtraBackup cannot clear tablespaces not in the data directory.

Other Reading

- SST Methods for MySQL
- *Xtrabackup SST configuration*

CHAPTER

FIFTEEN

PERCONA XTRABACKUP SST CONFIGURATION

Percona XtraBackup SST works in two stages:

1. First it identifies the type of data transfer based on the presence of `xtrabackup_ist` file on the joiner node.
2. 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.

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 the option must 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 the default `transferfmt=socat` because it allows for socket options, such as transfer buffer sizes. For more information, see socat(1).

Note: Using `transferfmt=nc` does not support the SSL-based encryption mode (value `4` for the *`encrypt`* option).

option `ssl-ca`

Example ssl-ca=/etc/ssl/certs/mycert.crt

Specifies the absolute path to the certificate authority (CA) file for `socat` encryption based on OpenSSL.

option `ssl-cert`

Example ssl-cert=/etc/ssl/certs/mycert.pem

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

Note: For more information about `ssl-ca` and `ssl-cert`, see http://www.dest-unreach.org/socat/doc/socat-openssltunnel.html. The `ssl-ca` is essentially a self-signed certificate in that example, and `ssl-cert` 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 `ssl-key`

Example ssl-key=/etc/ssl/keys/key.pem

Used to specify the full path to the private key in PEM format for socat encryption based on OpenSSL.

option `encrypt`

Values 0, 4

Default 4

Match Yes

Enables SST encryption mode in Percona XtraBackup:

- Set `encrypt=0` to disable SST encryption.
- Set `encrypt=4` for SST encryption with SSL files generated by MySQL. This is the recommended value.

  Considering that you have all three necessary files:

```
[sst]
encrypt=4
ssl-ca=ca.pem
ssl-cert=server-cert.pem
ssl-key=server-key.pem
```

For more information, see *Encrypting PXC Traffic*.

option `sockopt`

Used to specify key/value pairs of socket options, separated by commas, for example:

```
[sst]
sockopt="retry=2,interval=3"
```

The previous example causes socat to try to connect three times (initial attempt and two retries with a 3-second interval between attempts).

This option only applies when socat is used (`transferfmt=socat`). For more information about socket options, see socat (1).

Note: You can also enable SSL based compression with *`sockopt`*. This can be used instead of the Percona XtraBackup `compress` option.

---

option `ncsockopt`

Used to specify socket options for the `netcat` transfer format (`transferfmt=nc`).

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 `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`

Default
```
'.*\.pem$\|.*init\.ok$\|.*galera\.cache$\|.
   *sst_in_progress$\|.*\.sst$\|.*gvwstate\.dat$\|.*grastate\.
   dat$\|.*\.err$\|.*\.log$\|.*RPM_UPGRADE_MARKER$\|.
   *RPM_UPGRADE_HISTORY$'
```

Used to define the files that need to be retained 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$\|.
↪*RPM_UPGRADE_MARKER$\|.*RPM_UPGRADE_HISTORY$\|.*\.xyz$'
```

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 is used to pass XtraBackup options for backup, apply, and move stages. The SST script doesn't alter, tweak, or optimize these options.

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).

option `tmpdir`

Default Empty

Example /path/to/tmp/dir

This option specifies the location for storing the temporary file on a donor node where the transaction log is stored before streaming or copying it to a remote host.

Note: This option can be used on joiner node to specify non-default location to receive temporary SST files. This location must be large enough to hold the contents of the entire database. If tmpdir is empty then default location datadir/.sst will be used.

The `tmpdir` option can be set in the following `my.cnf` groups:

- `[sst]` is the primary location (others are ignored)
- `[xtrabackup]` is the secondary location (if not specified under `[sst]`)
- `[mysqld]` is used if it is not specified in either of the above

wsrep_debug

Specifies whether additional debugging output for the database server error log should be enabled. Disabled by default.

This option can be set in the following `my.cnf` groups:

- Under `[mysqld]` it enables debug logging for `mysqld` and the SST script
- Under `[sst]` it enables debug logging for the SST script only

option `encrypt_threads`

Default 4

Specifies the number of threads that XtraBackup should use for encrypting data. The value is passed using the `--encrypt-threads` option in XtraBackup.

This option affects only SST with XtraBackup and should be specified under the `[sst]` group.

option `backup_threads`

Default `4`

Specifies the number of threads that XtraBackup should use to create backups. See the `--parallel` option in XtraBackup.

This option affects only SST with XtraBackup and should be specified under the `[sst]` group.

XtraBackup SST Dependencies

Each suppored version of Percona XtraDB Cluster is tested against a specific version of Percona XtraBackup:

- Percona XtraDB Cluster 5.6 requires Percona XtraBackup 2.3
- Percona XtraDB Cluster 5.7 requires Percona XtraBackup 2.4
- Percona XtraDB Cluster 8.0 requires Percona XtraBackup 8.0

Other combinations are not guaranteed to work.

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* 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. `xbstream` is the default.
- `pv` is required for *`progress`* and *`rlimit`*.
- `mkfifo` is required for *`progress`*. Provided by `coreutils`.
- `mktemp` is required. Provided by `coreutils`.
- `which` is required.

XtraBackup-based Encryption

Settings related to XtraBackup-based Encryption are no longer allowed in Percona XtraDB Cluster 8.0 when used for *SST*. If it is detected that XtraBackup-based Encryption is enabled, Percona XtraDB Cluster will produce an error.

The XtraBackup-based Encryption is enabled when you specify any of the following options under `[xtrabackup]` in `my.cnf`:

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

Memory Allocation

The amount of memory for XtraBackup is defined by the `--use-memory` option. You can pass it using the *`inno-apply-opts`* option under `[sst]` as follows:

```
[sst]
inno-apply-opts="--use-memory=500M"
```

If it is not specified, the `use-memory` option under `[xtrabackup]` will be used:

```
[xtrabackup]
use-memory=32M
```

If neither of the above are specified, the size of the InnoDB memory buffer will be used:

```
[mysqld]
innodb_buffer_pool_size=24M
```

CHAPTER

SIXTEEN

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.

Note: If MySQL fails for any reason, it will not remove its PID file (which is by design deleted only on clean shutdown). Obviously server will not restart if existing PID file is present. So in case of encountered MySQL failure for any reason with the relevant records in log, PID file should be removed manually.

CHAPTER

SEVENTEEN

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.

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.

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.

Other Reading

- PXC - Failure Scenarios with only 2 nodes

CHAPTER

EIGHTEEN

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.

Manual Monitoring

Manual cluster monitoring can be performed using myq-tools.

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*.

Metrics

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

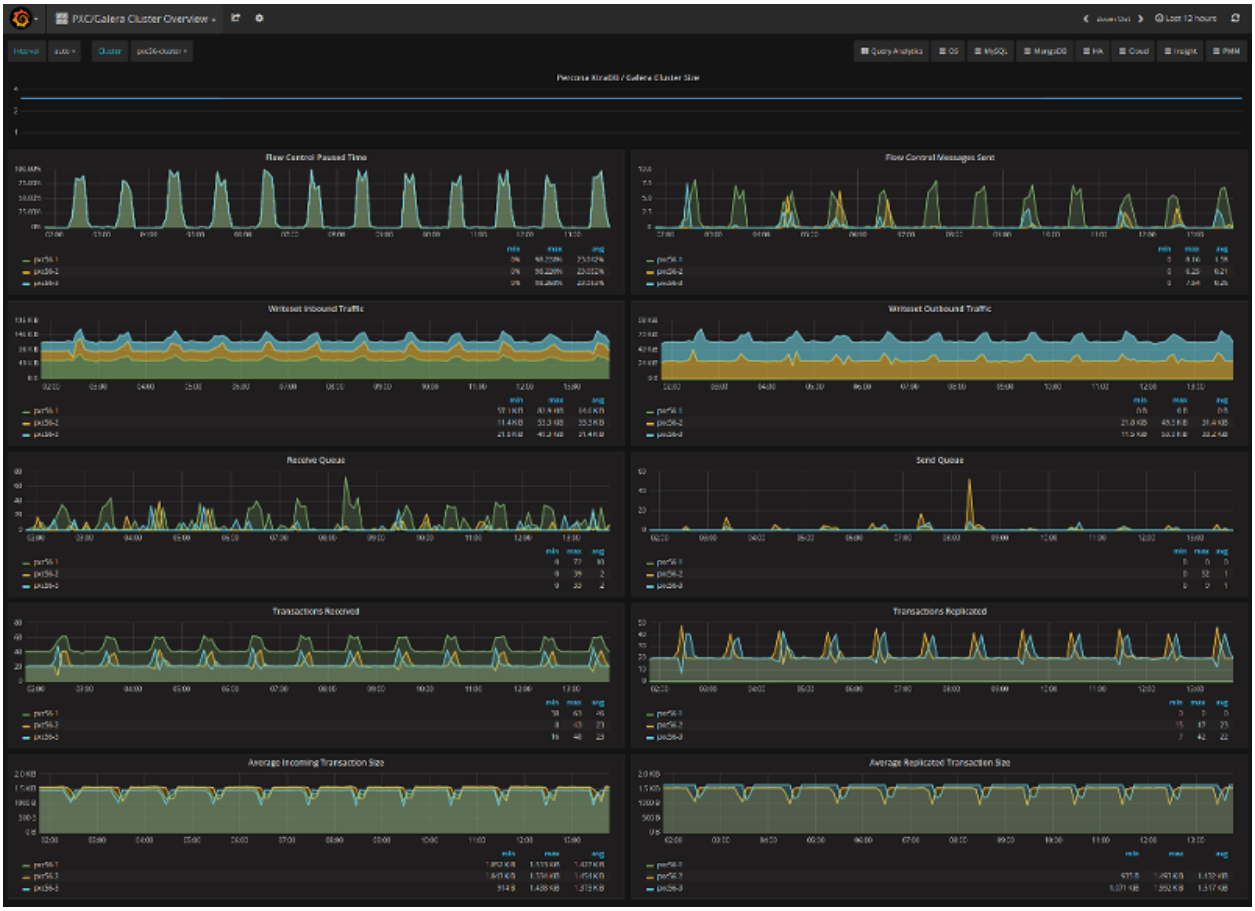
- Queue sizes: *wsrep_local_recv_queue* and *wsrep_local_send_queue*

- 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*

Using Percona Monitoring and Management

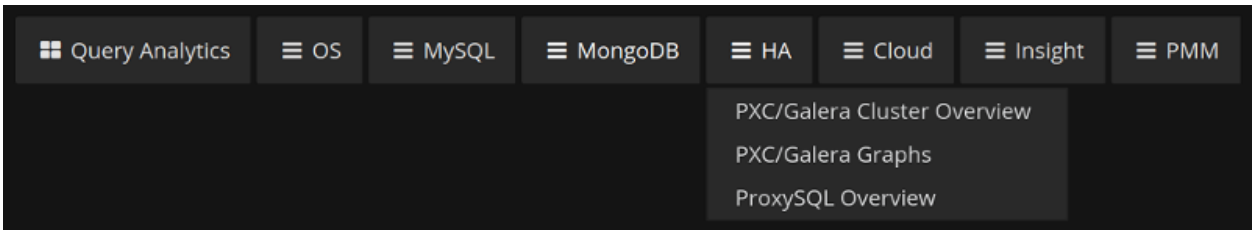
Percona Monitoring and Management includes two dashboards to monitor PXC:

1. PXC/Galera Cluster Overview:



2. PXC/Galera Graphs:

These dashboards are available from the menu:



Please refer to the official documentation for details on Percona Monitoring and Management installation and setup.

Other Reading

- Realtime stats to pay attention to in PXC and Galera

CHAPTER

NINETEEN

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 Source-Replica 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.

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-source 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.

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.

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.

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.

CHAPTER

TWENTY

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

- *Percona XtraDB Cluster threading model*

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 replica apply routine path, and relays the log info apply path with wsrep-customization. These threads are similar to replica 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()`).

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.

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.

CHAPTER

TWENTYONE

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.

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=64K` 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.

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

TWENTYTWO

PERFOMANCE SCHEMA INSTRUMENTATION

To improve monitoring *Percona XtraDB Cluster* has implemented an infrastructure to expose Galera instruments (mutexes, cond-variables, files, threads) as a part of `PERFOMANCE_SCHEMA`.

Although mutexes and condition variables from `wsrep` were already part of `PERFORMANCE_SCHEMA` threads weren't.

Mutexes, condition variables, threads, and files from Galera library also were not part of the `PERFORMANCE_SCHEMA`.

You can see the complete list of available instruments by running:

```
mysql> SELECT * FROM performance_schema.setup_instruments WHERE name LIKE '%galera%'␣
↪OR name LIKE '%wsrep%';

+-----------------------------------------------------------+---------+--------+
| NAME                                                      | ENABLED | TIMED  |
+-----------------------------------------------------------+---------+--------+
| wait/synch/mutex/sql/LOCK_wsrep_ready                     | NO      | NO     |
| wait/synch/mutex/sql/LOCK_wsrep_sst                       | NO      | NO     |
| wait/synch/mutex/sql/LOCK_wsrep_sst_init                  | NO      | NO     |
...
| stage/wsrep/wsrep: in rollback thread                     | NO      | NO     |
| stage/wsrep/wsrep: aborter idle                           | NO      | NO     |
| stage/wsrep/wsrep: aborter active                         | NO      | NO     |
+-----------------------------------------------------------+---------+--------+
73 rows in set (0.00 sec)
```

Some of the most important are:

- Two main actions that Galera does are `REPLICATION` and `ROLLBACK`. Mutexes, condition variables, and threads related to this are part of `PERFORMANCE_SCHEMA`.
- Galera internally uses monitor mechanism to enforce ordering of events. These monitor control events apply and are mainly responsible for the wait between different action. All such monitor mutexes and condition variables are covered as part of this implementation.
- There are lot of other miscellaneous action related to receiving of package and servicing messages. Mutexes and condition variables needed for them are now visible too. Threads that manage receiving and servicing are also being instrumented.

This feature has exposed all the important mutexes, condition variables that lead to lock/threads/files as part of this process.

Besides exposing file it also tracks write/read bytes like stats for file. These stats are not exposed for Galera files as Galera uses `mmap`.

Also, there are some threads that are short-lived and created only when needed especially for SST/IST purpose. They are also tracked but come into PERFORMANCE_SCHEMA tables only if/when they are created.

Stage Info from Galera specific function which server updates to track state of running thread is also visible in PERFORMANCE_SCHEMA.

What is not exposed ?

Galera uses customer data-structure in some cases (like STL structures). Mutexes used for protecting these structures which are not part of mainline Galera logic or doesn't fall in big-picture are not tracked. Same goes with threads that are gcomm library specific.

Galera maintains a process vector inside each monitor for its internal graph creation. This process vector is 65K in size and there are two such vectors per monitor. That is 128K * 3 = 384K condition variables. These are not tracked to avoid hogging PERFORMANCE_SCHEMA limits and sidelining of the main crucial information.

CHAPTER

TWENTYTHREE

DATA AT REST ENCRYPTION

- *Introduction*
- *Configuring PXC to use keyring_file plugin*
  - *keyring_file*
  - *Configuration*
  - *Usage*
  - *Compatibility*
- *Configuring PXC to use keyring_vault plugin*
  - *keyring_vault*
  - *Configuration*
- *Mix-match keyring plugins*
  - *Upgrade and compatibility issues*
- *Temporary file encryption*
- *Migrating Keys Between Keyring Keystores*
  - *Offline Migration*
  - *Online Migration*
  - *Migration server options*

Introduction

The *data at rest* term refers to all the data stored on disk on some server within system tablespace, general tablespace, redo-logs, undo-logs, etc. As an opposite, *data in transit* means data transmitted to other node or client. Data-in-transit can be encrypted using SSL connection (details are available in the encrypt traffic documentation) and therefore supposed to be safe. Below sections are about securing the data-at-rest only.

Percona XtraDB Cluster 8.0 supports all *data at rest* generally-available encryption features available from Percona Server for MySQL 8.0.

See also:

Percona Server for MySQL Documentation: Transparent Data Encryption https://www.percona.com/doc/percona-server/8.0/security/data-at-rest-encryption.html#data-at-rest-encryption

The "data-at-rest" term refers to all the data stored on disk by some server within system tablespace, general tablespace, redo-logs, undo-logs, etc. As the opposite, "data-in-transit" means data transmitted to another node or client. Data-in-transit can be encrypted using SSL connections (details are available in the encrypt traffic documentation).

This feature requires a keyring plugin to be loaded before it can be used. Currently, Percona Server (and in turn Percona XtraDB Cluster) supports two types of keyring plugin: `keyring_file` and `keyring_vault`.

Configuring PXC to use keyring_file plugin

keyring_file

The following subsection covers some of the important semantics of `keyring_file` plugin needed to use it in scope of data-at-rest encryption.

`keyring_file` stores encryption key to a physical file. The location of this file is specified by the `keyring_file_data` parameter configured during startup.

Configuration

Percona XtraDB Cluster inherits upstream (Percona Server) behavior to configure `keyring_file` plugin. The following options are set in the configuration file:

```
[mysqld]
early-plugin-load=keyring_file.so
keyring_file_data=<PATH>/keyring
```

A `SHOW PLUGINS` statement can be used further to check if the plugin has been successfully loaded.

Note: PXC recommends the same configuration on all the nodes of the cluster, and that also means all the nodes of the cluster should have keyring configured. Mismatch in keyring configuration will not allow the joiner node to join the cluster.

If the user has bootstrapped node with keyring enabled, then upcoming nodes of the cluster will inherit the keyring (encrypted key) from the DONOR node or generate it.

Usage

`xtrabackup` re-encrypts the data using a transition-key and the JOINER node re-encrypts it using a newly generated master-key.

The keyring is generated on the JOINER node. SST is done using the `xtrabackup-v2` method. The keyring is sent over explicitly before the real data backup/streaming starts.

Percona XtraDB Cluster doesn't allow you to combine nodes with encryption and nodes without encryption. This combination is not allowed to maintain data consistency. For example, the user creates node-1 with encryption (keyring) enabled and node-2 with encryption (keyring) disabled. If the user tries to create a table with encryption on node-1, it will fail on node-2, causing data inconsistency. A node fails to start if it fails to load the keyring plugin.

Note: If the user does not specify the keyring parameters, the node does not know that it must load the keyring. The JOINER node may start, but it eventually shuts down when the DML level inconsistency with encrypted tablespace is

detected.

If a node does not have an encrypted tablespace, the keyring is not generated and the keyring file is empty. The actual keyring is generated only when the node starts using encrypted tablespace.

The user can rotate the key as needed. This operation is local to the node. The `ALTER INSTANCE ROTATE INNODB MASTER KEY` statement is not replicated on cluster.

The JOINER node generates its own keyring.

Compatibility

Keyring (or, more generally, the Percona XtraDB Cluster SST process) is backward compatible, a higher version JOINER can join from lower version DONOR, but not vice-versa. More details are covered below, in *Upgrade and compatibility issues* section.

Configuring PXC to use keyring_vault plugin

keyring_vault

The `keyring_vault` plugin allows storing the master-key in vault-server (vs. local file as in case of `keyring_file`).

Warning: rsync does not support `keyring_vault`, and SST on a joiner is aborted if rsync is used on the node with `keyring_vault` configured.

Configuration

Configuration options are same as upstream. The `my.cnf` configuration file should contain following options:

```
[mysqld]
early-plugin-load="keyring_vault=keyring_vault.so"
keyring_vault_config="<PATH>/keyring_vault_n1.conf"
```

Also `keyring_vault_n1.conf` file contents should be :

```
vault_url = http://127.0.0.1:8200
secret_mount_point = secret1
token = e0345eb4-35dd-3ddd-3b1e-e42bb9f2525d
vault_ca = /data/keyring_vault_confs/vault_ca.crt
```

Detailed description of these options can be found in the upstream documentation.

Vault-server is an external server, make sure the PXC node can reach the said server.

Note: Percona XtraDB Cluster recommends using the same keyring_plugin on all the nodes of the cluster. Mix-match is recommended to use it only while transitioning from `keyring_file` -> `keyring_vault` or vice-versa.

It is not necessary that all the nodes refer to the same vault server, but ensure the vault server is accessible from the respective node. If a node cannot reach/connect to a vault server, an error is raised during the server boot, and node refuses to start:

```
... [Warning] Plugin keyring_vault reported: 'There is no vault_ca specified in␣
↪keyring_vault's configuration file. ...
... [ERROR] Plugin keyring_vault reported: 'CURL returned this error code: 7 with␣
↪error message : ...
```

If any nodes of the cluster cannot connect to the vault-server, users cannot access the encrypted data on that node.

If the vault-server is accessible but the authentication credential is wrong, the consequences are the same. The corresponding error message looks like the following:

```
... [Warning] Plugin keyring_vault reported: 'There is no vault_ca specified in␣
↪keyring_vault's configuration file. ...
... [ERROR] Plugin keyring_vault reported: 'Could not retrieve list of keys from␣
↪Vault. ...
```

All nodes are not required to use the same mount point. In the case of an accessible vault-server with the wrong mount point, there is no error during server boot, but node refuses to start:

```
mysql> CREATE TABLE t1 (c1 INT, PRIMARY KEY pk(c1)) ENCRYPTION='Y';
ERROR 3185 (HY000): Can't find master key from keyring, please check keyring plugin␣
↪is loaded.

... [ERROR] Plugin keyring_vault reported: 'Could not write key to Vault. ...
... [ERROR] Plugin keyring_vault reported: 'Could not flush keys to keyring'
```

Mix-match keyring plugins

With **xtrabackup** introducing transition-key logic, it is now possible to mix-match keyring plugins. For example, the user has node-1 configured to use `keyring_file` plugin and node-2 configured to use `keyring_vault`.

Note: Percona recommends using the same configuration for all the nodes of the cluster. Mix-match (in keyring plugins) is recommended only during the transition from one keying method to another.

Upgrade and compatibility issues

If all the nodes are using Percona XtraDB Cluster 8.0, then the user can freely configure some nodes to use `keyring_file` and other to use `keyring_vault`. This setup, however, is not recommended and should be used during transitioning to vault only.

Temporary file encryption

Percona Server supports the encryption of temporary file storage which is enabled using `encrypt-tmp-files`. The storage or files are local to the node and have no effect on Percona XtraDB Cluster replication. Percona XtraDB Cluster recommends enabling the variable on all nodes of the cluster, although this action is not mandatory. The parameter to enable this option is the same as Percona Server:

```
[mysqld]
encrypt-tmp-files=ON
```

Migrating Keys Between Keyring Keystores

Percona XtraDB Cluster supports key migration between keystores. The migration can be performed offline or online.

Offline Migration

In offline migration, the node to migrate is shut down, and the migration server takes care of migrating keys for the said server to a new keystore.

Following example illustrates this scenario:

1. Three Percona XtraDB Cluster nodes n1, n2, n3 - all using `keyring_file`, and n2 should be migrated to use `keyring_vault`
2. The user shuts down the n2 node.
3. The user starts the Migration Server (`mysqld` with a special option).
4. Migration Server copies keys from the n2 keyring file and adds them to the vault server.
5. The User starts n2 node with the vault parameter, and keys should be available.

Here is how the migration server output should look like:

```
/dev/shm/pxc80/bin/mysqld --defaults-file=/dev/shm/pxc80/copy_mig.cnf \
--keyring-migration-source=keyring_file.so \
--keyring_file_data=/dev/shm/pxc80/node2/keyring \
--keyring-migration-destination=keyring_vault.so \
--keyring_vault_config=/dev/shm/pxc80/vault/keyring_vault.cnf &

... [Warning] TIMESTAMP with implicit DEFAULT value is deprecated. Please use
    --explicit_defaults_for_timestamp server option (see documentation for more␣
↪details).
... [Note] --secure-file-priv is set to NULL. Operations related to importing and
    exporting data are disabled
... [Warning] WSREP: Node is not a cluster node. Disabling pxc_strict_mode
... [Note] /dev/shm/pxc80/bin/mysqld (mysqld 8.0-debug) starting as process 5710 ...
... [Note] Keyring migration successful.
```

On successful migration, destination keystore receives additional migrated keys (pre-existing keys in the destination keystore are not touched or removed). The source keystore retains the keys as migration performs copy operation and not move operation.

If the migration fails, then destination keystore is left untouched.

Online Migration

In online migration, node to migrate is kept running and migration server takes care of migrating keys for the said server to a new keystore by connecting to the node.

The following example illustrates this scenario:

1. Three Percona XtraDB Cluster nodes n1, n2, n3 - all using `keyring_file`, and n3 should be migrated to use `keyring_vault`
2. User starts the Migration Server (`mysqld` with a special option).
3. Migration Server copies keys from the n3 keyring file and adds them to the vault server.
4. The user restarts n3 node with the vault parameter, and keys should be available.

```
/dev/shm/pxc80/bin/mysqld --defaults-file=/dev/shm/pxc80/copy_mig.cnf \
--keyring-migration-source=keyring_vault.so \
--keyring_vault_config=/dev/shm/pxc80/keyring_vault3.cnf \
--keyring-migration-destination=keyring_file.so \
--keyring_file_data=/dev/shm/pxc80/node3/keyring \
--keyring-migration-host=localhost \
--keyring-migration-user=root \
--keyring-migration-port=16300 \
--keyring-migration-password='' &
```

On a successful migration, the destination keystore receives additional migrated keys (pre-existing keys in destination keystore are not touched or removed). The source keystore retains the keys as the migration performs copy operation and not move operation.

If the migration fails, then the destination keystore will be left untouched.

Migration server options

- `--keyring-migration-source`: The source keyring plugin that manages the keys to be migrated.
- `--keyring-migration-destination`: The destination keyring plugin to which the migrated keys are to be copied

  Note: For offline migration, no additional key migration options are needed.

- `--keyring-migration-host`: The host where the running server is located. This host is always the local host.
- `--keyring-migration-user`, `--keyring-migration-password`: The username and password for the account used to connect to the running server.
- `--keyring-migration-port`: Used for TCP/IP connections, the running server's port number used to connect.
- `--keyring-migration-socket`: Used for Unix socket file or Windows named pipe connections, the running server socket or named pipe used to connect.

Prerequisite for migration:

Make sure to pass required keyring options and other configuration parameters for the two keyring plugins. For example, if `keyring_file` is one of the plugins, you must set the `keyring_file_data` system variable if the keyring data file location is not the default location.

Other non-keyring options may be required as well. One way to specify these options is by using `--defaults-file` to name an option file that contains the required options.

```
[mysqld]
basedir=/dev/shm/pxc80
datadir=/dev/shm/pxc80/copy_mig
log-error=/dev/shm/pxc80/logs/copy_mig.err
```

```
socket=/tmp/copy_mig.sock
port=16400
```

Part VI

How-tos

CHAPTER

TWENTYFOUR

UPGRADING PERCONA XTRADB CLUSTER

The following documents contain details about relevant changes in the 8.0 series of MySQL and Percona Server for MySQL. Make sure you deal with any incompatible features and variables mentioned in these documents when upgrading to Percona XtraDB Cluster 8.0.

- Changed in Percona Server for MySQL 8.0
- Upgrading MySQL
- Upgrading from MySQL 5.7 to 8.0

In this section

- *Important changes in Percona XtraDB Cluster 8.0*
- *Major Upgrade Scenarios*
- *Minor upgrade*

Important changes in Percona XtraDB Cluster 8.0

- *Traffic encryption is enabled by default*
- *Not recommended to mix Percona XtraDB Cluster 5.7 nodes with Percona XtraDB Cluster 8.0 nodes*
- PXC Strict Mode *is enabled by default*
- *The configuration file layout has changed in Percona XtraDB Cluster 8.0*
- `caching_sha2_password` *is the default authentication plugin*
- **`mysql_upgrade`** *is part of* SST

Traffic encryption is enabled by default

The `pxc_encrypt_cluster_traffic` variable, which enables traffic encryption, is set to `ON` by default in Percona XtraDB Cluster 8.0.

Unless you configure a node accordingly (each node in your cluster must use the same SSL certificates) or try to join a cluster running Percona XtraDB Cluster 5.7 which unencrypted cluster traffic, the node will not be able to join resulting in an error.

```
... [ERROR] ... [Galera] handshake with remote endpoint ...
This error is often caused by SSL issues. ...
```

See also:

sections *Encrypting PXC Traffic*, *Configuring Nodes for Write-Set Replication*

Not recommended to mix Percona XtraDB Cluster 5.7 nodes with Percona XtraDB Cluster 8.0 nodes

Shut down the cluster and upgrade each node to Percona XtraDB Cluster 8.0. It is important that you make backups before attempting an upgrade.

PXC Strict Mode is enabled by default

Percona XtraDB Cluster in 8.0 runs with *PXC Strict Mode* enabled by default. This will deny any unsupported operations and may halt the server if *a strict mode validation fails*. It is recommended to first start the node with the `pxc_strict_mode` variable set to `PERMISSIVE` in the *MySQL* configuration file (on Debian and Ubuntu `/etc/mysql/mysql.conf.d/mysqld.cnf`; on CentOS and Red Hat `/etc/my.cnf`).

After you check the log for any experimental or unsupported features and fix any encountered incompatibilities, you can set the variable back to `ENFORCING` at run time:

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

Also, switching back to `ENFORCING` may be done by restarting the node with the updated configuration file (on Debian and Ubuntu `/etc/mysql/mysql.conf.d/mysqld.cnf`; on CentOS and Red Hat `/etc/my.cnf`).

The configuration file layout has changed in Percona XtraDB Cluster 8.0

All configuration settings are stored in the default *MySQL* configuration file:

- Path on Debian and Ubuntu: `/etc/mysql/mysql.conf.d/mysqld.cnf`
- Path on Red Hat and CentOS: `/etc/my.cnf`

Before you start the upgrade, move your custom settings from `/etc/mysql/percona-xtradb-cluster.conf.d/wsrep.cnf` (on Debian and Ubuntu) or from `/etc/percona-xtradb-cluster.conf.d/wsrep.cnf` (on Red Hat and CentOS) to the new location accordingly.

`caching_sha2_password` is the default authentication plugin

In Percona XtraDB Cluster 8.0, the default authentication plugin is `caching_sha2_password`. The ProxySQL option *–syncusers* will not work if the Percona XtraDB Cluster user is created using `caching_sha2_password`. Use the `mysql_native_password` authentication plugin in these cases.

Be sure you are running on the latest 5.7 version before you upgrade to 8.0.

`mysql_upgrade` is part of SST

`mysql_upgrade` is now run automatically as part of *SST*. You do not have to run it manually when upgrading your system from an older version.

Major Upgrade Scenarios

Upgrading Percona XtraDB Cluster from 5.7 to 8.0 may have slightly different strategies depending on the configuration and workload on your Percona XtraDB Cluster cluster.

Note that the new default value of `pxc-encrypt-cluster-traffic` (set to *ON* versus *OFF* in Percona XtraDB Cluster 5.7) requires additional care. You cannot join a 5.7 node to a Percona XtraDB Cluster 8.0 cluster unless the node has traffic encryption enabled as the cluster may not have some nodes with traffic encryption enabled and some nodes with traffic encryption disabled. For more information, see *Traffic encryption is enabled by default*.

- *Scenario: No active parallel workload or with read-only workload*
- *Scenario: Upgrading from Percona XtraDB Cluster 5.6 to Percona XtraDB Cluster 8.0*

Scenario: No active parallel workload or with read-only workload

If there is no active parallel workload or the cluster has read-only workload while upgrading the nodes, complete the following procedure for each node in the cluster:

1. Shutdown one of the node 5.7 cluster nodes.
2. Remove 5.7 PXC packages without removing the data-directory.
3. Install Percona XtraDB Cluster 8.0 packages.
4. Restart the mysqld service.

Important: Before upgrading, make sure your application can work with a reduced cluster size. Also, during the upgrade, the cluster will operate with an even number of nodes - the cluster may run into the split-brain problem.

This upgrade flow auto-detects the presence of the 5.7 data directory and trigger the upgrade as part of the node bootup process. The data directory is upgraded to be compatible with Percona XtraDB Cluster 8.0. Then the node joins the cluster and enters synced state. The 3-node cluster is restored with 2 nodes running Percona XtraDB Cluster 5.7 and 1 node running Percona XtraDB Cluster 8.0.

Note: Since *SST* is not involved, *SST* based auto-upgrade flow is not started.

Percona XtraDB Cluster 8.0 uses Galera 4 while Percona XtraDB Cluster 5.7 uses Galera-3. The cluster will continue to use the protocol version 3 used in Galera 3 effectively limiting some of the functionality. With all nodes upgraded to version 8.0, protocol version 4 is applied.

Hint: The protocol version is stored in the `protocol_version` column of the ``wsrep_cluster` table.

```
mysql> USE mysql;
...
mysql> SELECT protocol_version from wsrep_cluster;
+------------------+
| protocol_version |
+------------------+
|                4 |
```

```
+------------------+
1 row in set (0.00 sec)
```

As soon as the last 5.7 node shuts down, the configuration of the remaining two nodes is updated to use protocol version 4. A new upgraded node will then join using protocol version 4 and the whole cluster will maintain protocol version 4 enabling the support for additional Galera 4 facilities.

It may take longer to join the last upgraded node since it will invite *IST* to obtain the configuration changes.

Note: Starting from Galera 4, the configuration changes are cached to `gcache` and the configuration changes are donated as part of *IST* or *SST* to help build the certification queue on the JOINING node. As other nodes (say n2 and n3), already using protocol version 4, donate the configuration changes when the JOINER node is booted.

The situation was different for the previous and penultimate nodes since the donation of the configuration changes is not supported by protocol version 3 that they used.

With *IST* involved on joining the last node, the smart IST flow is triggered to take care of the upgrade even before MySQL starts to look at the data directory.

Important: It is not recommended to restart the last node without upgrading it.

Scenario: Upgrading from Percona XtraDB Cluster 5.6 to Percona XtraDB Cluster 8.0

First, upgrade Percona XtraDB Cluster from 5.6 to the latest version of Percona XtraDB Cluster 5.7. Then proceed with the upgrade using the procedure described in *Scenario: No active parallel workload or with read-only workload*.

Minor upgrade

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. Upgrade Percona XtraDB Cluster and Percona XtraBackup packages. For more information, see *Installing Percona XtraDB Cluster*.
4. Back up `grastate.dat`, so that you can restore it if it is corrupted or zeroed out due to network issue.
5. Now, start the cluster node with 8.0 packages installed, Percona XtraDB Cluster will upgrade the data directory as needed - either as part of the startup process or a state transfer (IST/SST).

   In most cases, starting the `mysql` service should run the node with your previous configuration. For more information, see *Adding Nodes to Cluster*.

   ```
   $ sudo service mysql start
   ```

Note: On CentOS, the `/etc/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.

PXC Strict Mode is enabled by default, which may result in denying any unsupported operations and may halt the server. For more information, see *pxc-strict-mode is enabled by default*.

`pxc-encrypt-cluster-traffic` is enabled by default. You need to configure each node accordingly and avoid joining a cluster with unencrypted cluster traffic. For more information, see *Traffic encryption is enabled by default*.

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

CHAPTER

TWENTYFIVE

CRASH RECOVERY

Unlike the standard MySQL replication, a Percona XtraDB Cluster cluster acts like one logical entity, which controls the status and consistency of each node as well as the status of the whole cluster. This allows maintaining the data integrity more efficiently than with traditional asynchronous replication without losing safe writes on multiple nodes at the same time.

However, there are scenarios where the database service can stop with no node being able to serve requests.

- *Scenario: Node A is gracefully stopped*
- *Scenario: Two nodes are gracefully stopped*
- *Scenario: All three nodes are gracefully stopped*
- *Scenario: One node disappears from the cluster*
- *Scenario: Two nodes disappear from the cluster*
- *Scenario: All nodes went down without a proper shutdown procedure*
- *Scenario: The cluster loses its primary state due to split brain*

Scenario: Node A is gracefully stopped

In a three node cluster (node A, Node B, node C), one node (node A, for example) is gracefully stopped: for the purpose of maintenance, configuration change, etc.

In this case, the other nodes receive a "good bye" message from the stopped node and the cluster size is reduced; some properties like quorum calculation or auto increment are automatically changed. As soon as node A is started again, it joins the cluster based on its `wsrep_cluster_address` variable in `my.cnf`.

If the writeset cache (`gcache.size`) on nodes B and/or C still has all the transactions executed while node A was down, joining is possible via *IST*. If *IST* is impossible due to missing transactions in donor's gcache, the fallback decision is made by the donor and *SST* is started automatically.

Scenario: Two nodes are gracefully stopped

Similar to *Scenario: Node A is gracefully stopped*, the cluster size is reduced to 1 — even the single remaining node C forms the primary component and is able to serve client requests. To get the nodes back into the cluster, you just need to start them.

However, when a new node joins the cluster, node C will be switched to the "Donor/Desynced" state as it has to provide the state transfer at least to the first joining node. It is still possible to read/write to it during that process, but it may be much slower, which depends on how large amount of data should be sent during the state transfer. Also, some load balancers may consider the donor node as not operational and remove it from the pool. So, it is best to avoid the situation when only one node is up.

If you restart node A and then node B, you may want to make sure note B does not use node A as the state transfer donor: node A may not have all the needed writesets in its gcache. Specify node C node as the donor in your configuration file and start the *mysql* service:

```
$ systemctl start mysql
```

See also:

Galera Documentation: wsrep_sst_donor option https://galeracluster.com/library/documentation/mysql-wsrep-options.html#wsrep-sst-donor

Scenario: All three nodes are gracefully stopped

The cluster is completely stopped and the problem is to initialize it again. It is important that a PXC node writes its last executed position to the `grastate.dat` file.

By comparing the seqno number in this file, you can see which is the most advanced node (most likely the last stopped). The cluster must be bootstrapped using this node, otherwise nodes that had a more advanced position will have to perform the full *SST* to join the cluster initialized from the less advanced one. As a result, some transactions will be lost). To bootstrap the first node, invoke the startup script like this:

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

Note: Even though you bootstrap from the most advanced node, the other nodes have a lower sequence number. They will still have to join via the full *SST* because the *Galera Cache* is not retained on restart.

For this reason, it is recommended to stop writes to the cluster *before* its full shutdown, so that all nodes can stop at the same position. See also `pc.recovery`.

Scenario: One node disappears from the cluster

This is the case when one node becomes unavailable due to power outage, hardware failure, kernel panic, mysqld crash, `kill -9` on mysqld pid, etc.

Two remaining nodes notice the connection to node A is down and start trying to re-connect to it. After several timeouts, node A is removed from the cluster. The quorum is saved (2 out of 3 nodes are up), so no service disruption happens. After it is restarted, node A joins automatically (as described in *Scenario: Node A is gracefully stopped*).

Scenario: Two nodes disappear from the cluster

Two nodes are not available and the remaining node (node C) is not able to form the quorum alone. The cluster has to switch to a non-primary mode, where MySQL refuses to serve any SQL queries. In this state, the `mysqld` process on node C is still running and can be connected to but any statement related to data fails with an error

```
mysql> select * from test.sbtest1;
ERROR 1047 (08S01): WSREP has not yet prepared node for application use
```

Reads are possible until node C decides that it cannot access node A and node B. New writes are forbidden.

As soon as the other nodes become available, the cluster is formed again automatically. If node B and node C were just network-severed from node A, but they can still reach each other, they will keep functioning as they still form the quorum.

If node A and node B crashed, you need to enable the primary component on node C manually, before you can bring up node A and node B. The command to do this is:

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

This approach only works if the other nodes are down before doing that! Otherwise, you end up with two clusters having different data.

Cross Reference

Adding Nodes to Cluster

Scenario: All nodes went down without a proper shutdown procedure

This scenario is possible in case of a datacenter power failure or when hitting a MySQL or Galera bug. Also, it may happen as a result of data consistency being compromised where the cluster detects that each node has different data. The `grastate.dat` file is not updated and does not contain a valid sequence number (seqno). It may look like this:

```
$ cat /var/lib/mysql/grastate.dat
# GALERA saved state
version: 2.1
uuid: 220dcdcb-1629-11e4-add3-aec059ad3734
seqno: -1
safe_to_bootstrap: 0
```

In this case, you cannot be sure that all nodes are consistent with each other. We cannot use *safe_to_bootstrap* variable to determine the node that has the last transaction committed as it is set to **0** for each node. An attempt to bootstrap from such a node will fail unless you start `mysqld` with the `--wsrep-recover` parameter:

```
$ mysqld --wsrep-recover
```

Search the output for the line that reports the recovered position after the node UUID (**1122** in this case):

```
...
... [Note] WSREP: Recovered position: 220dcdcb-1629-11e4-add3-aec059ad3734:1122
...
```

The node where the recovered position is marked by the greatest number is the best bootstrap candidate. In its `grastate.dat` file, set the *safe_to_bootstrap* variable to **1**. Then, bootstrap from this node.

Note: After a shutdown, you can boostrap from the node which is marked as safe in the `grastate.dat` file.

```
...
safe_to_bootstrap: 1
...
```

See also:

Galera Documentation Introducing the "Safe-To-Bootstrap" feature in Galera Cluster

In recent Galera versions, the option *pc.recovery* (enabled by default) saves the cluster state into a file named `gvwstate.dat` on each member node. As the name of this option suggests (pc – primary component), it saves only a cluster being in the PRIMARY state. An example content of : file may look like this:

```
cat /var/lib/mysql/gvwstate.dat
my_uuid: 76de8ad9-2aac-11e4-8089-d27fd06893b9
#vwbeg
view_id: 3 6c821ecc-2aac-11e4-85a5-56fe513c651f 3
bootstrap: 0
member: 6c821ecc-2aac-11e4-85a5-56fe513c651f 0
member: 6d80ec1b-2aac-11e4-8d1e-b2b2f6caf018 0
member: 76de8ad9-2aac-11e4-8089-d27fd06893b9 0
#vwend
```

We can see a three node cluster with all members being up. Thanks to this new feature, the nodes will try to restore the primary component once all the members start to see each other. This makes the PXC cluster automatically recover from being powered down without any manual intervention! In the logs we will see:

Scenario: The cluster loses its primary state due to split brain

For the purpose of this example, let's assume we have a cluster that consists of an even number of nodes: six, for example. Three of them are in one location while the other three are in another location and they lose network connectivity. It is best practice to avoid such topology: if you cannot have an odd number of real nodes, you can use an additional arbitrator (garbd) node or set a higher pc.weight to some nodes. But when the split brain happens any way, none of the separated groups can maintain the quorum: all nodes must stop serving requests and both parts of the cluster will be continuously trying to re-connect.

If you want to restore the service even before the network link is restored, you can make one of the groups primary again using the same command as described in *Scenario: Two nodes disappear from the cluster*

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

After this, you are able to work on the manually restored part of the cluster, and the other half should be able to automatically re-join using *IST* as soon as the network link is restored.

Warning: If you set the bootstrap option on both the separated parts, you will end up with two living cluster instances, with data likely diverging away from each other. Restoring a network link in this case will not make them re-join until the nodes are restarted and members specified in configuration file are connected again.

Then, as the Galera replication model truly cares about data consistency: once the inconsistency is detected, nodes that cannot execute row change statement due to a data difference – an emergency shutdown will be performed and the only way to bring the nodes back to the cluster is via the full *SST*

Based on material from Percona Database Performance Blog

This article is based on the blog post *Galera replication - how to recover a PXC cluster by *Przemysław Malkowski*:
https://www.percona.com/blog/2014/09/01/galera-replication-how-to-recover-a-pxc-cluster/

CHAPTER

TWENTYSIX

CONFIGURING PERCONA XTRADB CLUSTER ON CENTOS

This tutorial describes how to install and configure three Percona XtraDB Cluster nodes on CentOS 7 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`

Prerequisites

The procedure described in this tutorial requires the following:

- All three nodes have CentOS 7 installed.
- The firewall on all nodes is configured to allow connecting to ports 3306, 4444, 4567 and 4568.
- SELinux on all nodes is disabled.

Different from previous version

The variable `wsrep_sst_auth` has been removed. Percona XtraDB Cluster 8.0 automatically creates the system user *mysql.pxc.internal.session*. During *SST*, the user *mysql.pxc.sst.user* and the role *mysql.pxc.sst.role* are created on the donor node.

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*.

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 First Node*.

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/galera4/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
```

2. Start the first node with the following command:

```
[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                                    |
+----------------------------+---------------------------------------+
75 rows in set (0.00 sec)
```

This output shows that the cluster has been successfully bootstrapped.

Copy the automatically generated temporary password for the superuser account:

```
$ sudo grep 'temporary password' /var/log/mysqld.log
```

Use this password to log in as root:

```
$ mysql -u root -p
```

Change the password for the superuser account and log out. For example:

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'r00tP@$$';
Query OK, 0 rows affected (0.00 sec)
```

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/galera4/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
```

2. Start the second node with the following command:

```
[root@percona2 ~]# systemctl start mysql
```

1. After the server has been started, it should receive *SST* automatically. 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.

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/galera4/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
```

2. Start the third node with the following command:

```
[root@percona3 ~]# systemctl start mysql
```

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.

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.

CHAPTER

TWENTYSEVEN

CONFIGURING PERCONA XTRADB CLUSTER ON UBUNTU

This tutorial describes how to install and configure three Percona XtraDB Cluster nodes on Ubuntu 14 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`

Prerequisites

The procedure described in this tutorial requires he following:

- All three nodes have Ubuntu 14 LTS installed.
- Firewall on all nodes is configured to allow connecting to ports 3306, 4444, 4567 and 4568.
- AppArmor profile for *MySQL* is disabled.

Step 1. Installing PXC

Install Percona XtraDB Cluster on all three nodes as described in *Installing Percona XtraDB Cluster on Debian or 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 `sudo systemctl stop mysql`.

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 First Node*.

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
```

2. Start the first node with the following command:

```
[root@pxc1 ~]# systemctl start mysql@bootstrap.service
```

   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                                   |
```

```
+---------------------------+-------------------------------------+
75 rows in set (0.00 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 PROCESS, 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.

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
```

2. Start the second node with the following command:

```
[root@pxc2 ~]# systemctl start mysql
```

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.

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
```

2. Start the third node with the following command:

```
[root@pxc3 ~]# systemctl start mysql
```

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.

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.

CHAPTER

TWENTYEIGHT

SETTING UP GALERA ARBITRATOR

Galera Arbitrator <http://galeracluster.com/documentation-webpages/arbitrator.html> is a member of *Percona XtraDB Cluster* that is used for voting in case you have a small number of servers (usually two) and don't want to add any more resources. Galera Arbitrator does not need a dedicated server. It can be installed on a machine running some other application. Just make sure it has good network connectivity.

Galera Arbitrator is a member of the cluster that participates in the voting, but not in actual replication (although it receives the same data as other nodes). Also, it is not included in flow control calculations.

This document will show how to add Galera Arbitrator node to an existing cluster.

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

Installation

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

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

on Debian/Ubuntu distributions, or:

```
[root@centos ~]# yum install percona-xtradb-cluster-garbd
```

on CentOS/RHEL distributions.

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 *Configuring 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:

- On Debian or Ubuntu:

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

- On Red Hat Enterprise Linux or CentOS:

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

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

- On Debian or Ubuntu:

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

- On Red Hat Enterprise Linux or CentOS:

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

CHAPTER

TWENTYNINE

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-8.0.x86_64`

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-8.0.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-8.0.x86_64/lib/libgalera_smm.
↪so
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=xtrabackup-v2
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-8.0.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-8.0.x86_64/lib/libgalera_smm.
↪so
```

```
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=xtrabackup-v2
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-8.0.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-8.0.x86_64/lib/libgalera_smm.
↪so
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=xtrabackup-v2
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-
↪0800-94bd91e32efa:0, view# 1: Primary, number of nodes: 1, my index: 0,␣
↪protocol version 1
```

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/mysqld
tcp        0      0 0.0.0.0:4000                0.0.0.0:*                   ␣
↪LISTEN      21895/mysqld
```

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:

```
$ mysql -h127.0.0.1 -P5000 -e "CREATE DATABASE hello_peter"
```

CHAPTER

THIRTY

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 7 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 Percona XtraDB Cluster and Percona Server packages for older versions:
   - Percona XtraDB Cluster 5.6, 5.7
   - Percona Server 5.6, 5.7
2. Install 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=xtrabackup-v2
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@pxc1 ~]# systemctl start mysql@bootstrap.service
```

You should see the following output:

```
2014-01-30 11:52:35 23280 [Note] /usr/sbin/mysqld: ready for connections.
Version: '...'  socket: '/var/lib/mysql/mysql.sock'  port: 3306  Percona XtraDB␣
↪Cluster (GPL), Release ..., Revision ..., wsrep_version
```

7. Start the second and third nodes:

```
$ sudo systemctl start mysql
```

The output should be similar to the following:

```
... [Note] WSREP: Flow-control interval: [28, 28]
... [Note] WSREP: Restored state OPEN -> JOINED (2)
... [Note] WSREP: Member 2 (percona1) synced with group.
... [Note] WSREP: Shifting JOINED -> SYNCED (TO: 2)
... [Note] WSREP: New cluster view: global state: 4827a206-876b-11e3-911c-
↪3e6a77d54953:2, view# 7: Primary, number of nodes: 3, my index: 2, protocol␣
↪version 2
... [Note] WSREP: SST complete, seqno: 2
... [Note] Plugin 'FEDERATED' is disabled.
... [Note] InnoDB: The InnoDB memory heap is disabled
... [Note] InnoDB: Mutexes and rw_locks use GCC atomic builtins
... [Note] InnoDB: Compressed tables use zlib 1.2.3
... [Note] InnoDB: Using Linux native AIO
... [Note] InnoDB: Not using CPU crc32 instructions
... [Note] InnoDB: Initializing buffer pool, size = 128.0M
... [Note] InnoDB: Completed initialization of buffer pool
... [Note] InnoDB: Highest supported file format is Barracuda.
... [Note] InnoDB: 128 rollback segment(s) are active.
... [Note] InnoDB: Waiting for purge to start
... [Note] InnoDB:  Percona XtraDB (http://www.percona.com) ... started; log␣
↪sequence number 1626341
... [Note] RSA private key file not found: /var/lib/mysql//private_key.pem. Some␣
↪authentication plugins will not work.
... [Note] RSA public key file not found: /var/lib/mysql//public_key.pem. Some␣
↪authentication plugins will not work.
... [Note] Server hostname (bind-address): '*'; port: 3306
... [Note] IPv6 is available.
... [Note]   - '::' resolves to '::';
```

```
... [Note] Server socket created on IP: '::'.
... [Note] Event Scheduler: Loaded 0 events
... [Note] /usr/sbin/mysqld: ready for connections.
Version: '...'  socket: '/var/lib/mysql/mysql.sock'  port: 3306  Percona XtraDB␣
↪Cluster (GPL), Release ..., Revision ..., wsrep_version
... [Note] WSREP: inited wsrep sidno 1
... [Note] WSREP: wsrep_notify_cmd is not defined, skipping notification.
... [Note] WSREP: REPL Protocols: 5 (3, 1)
... [Note] WSREP: Assign initial position for certification: 2, protocol version:␣
↪3
... [Note] WSREP: Service thread queue flushed.
... [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.

CHAPTER

THIRTYONE

LOAD BALANCING WITH HAPROXY

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

Start by installing HAProxy on a *node* that you intend to use for load balancing. The operating systems that support Percona XtraDB Cluster provide the *haproxy* package and you can install it using the package manager.

Debian or Ubuntu

```
$ sudo apt update
$ sudo apt install haproxy
```

Red Hat or CentOS:

```
$ sudo yum update
$ sudo yum install haproxy
```

Supported versions of HAProxy

The lowest supported version of HAProxy is 1.4.20.

To start HAProxy, use the `haproxy` command. You may pass any number of configuration parameters on the command line. To use a configuration file, use the `-f` option.

```
$ # Passing one configuration file
$ sudo haproxy -f haproxy-1.cfg

$ # Passing multiple configuration files
$ sudo haproxy -f haproxy-1.cfg haproxy-2.cfg

$ # Passing a directory
$ sudo haproxy -f conf-dir
```

You can pass the name of an existing configuration file or a directory. HAProxy includes all files with the *.cfg* extension in the the supplied directory. Another way to pass multiple files is to use `-f` multiple times.

See also:

HAProxy Documentation:

- Managing HAProxy (including available options)
- More information about how to configure HAProxy

Example of the configuration file for HAProxy:

```
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
```

Table 31.1: Options set in the configuration file

HAProxy option (with links to HAProxy documentation)	Description
global	A section in the configuration file for process-wide parameters
defaults	A section in the configuration file for default parameters for all other following sections
listen	A section in the configuration file that defines a complete proxy with its frontend and backend parts combined in one section
balance	Load balancing algorithm to be used in a backend
clitimeout	Set the maximum inactivity time on the client side
contimeout	Set the maximum time to wait for a connection attempt to a server to succeed.
daemon	Makes the process fork into background (recommended mode of operation)
gid	Changes the process' group ID to <number>
log	Adds a global syslog server
maxconn	Sets the maximum per-process number of concurrent connections to <number>
mode	Set the running mode or protocol of the instance
option dontlognull	Disable logging of null connections
option tcplog	Enable advanced logging of TCP connections with session state and timers
redispatch	Enable or disable session redistribution in case of connection failure
retries	Set the number of retries to perform on a server after a connection failure
server	Declare a server in a backend
srvtimeout	Set the maximum inactivity time on the server side
uid	Changes the process' user ID to <number>

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
```

Important: In Percona XtraDB Cluster 8.0, the default authentication plugin is `caching_sha2_password`. HAProxy does not support this authentication plugin. Create a mysql user using the `mysql_native_password` authentication plugin.

```
mysql> CREATE USER 'haproxy_user'@'%' IDENTIFIED WITH mysql_native_password by '$3Kr$t
↪';
```

See also:

***MySQL* Documentation: CREATE USER statement** https://dev.mysql.com/doc/refman/8.0/en/create-user.html

CHAPTER

THIRTYTWO

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 in the ProxySQL admin interface. These include runtime parameters, server grouping, and traffic-related settings.

See also:

More information about ProxySQL.

ProxySQL v2 natively supports Percona XtraDB Cluster. With this version, `proxysql-admin` tool does not require any custom scripts to keep track of Percona XtraDB Cluster status.

Important: In version 8.0, Percona XtraDB Cluster does not support ProxySQL v1.

Manual Configuration

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

Node	Host Name	IP address
Node 1	pxc1	192.168.70.71
Node 2	pxc2	192.168.70.72
Node 3	pxc3	192.168.70.73
Node 4	proxysql	192.168.70.74

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:

Changes in the installation procedure

In Percona XtraDB Cluster 8.0, ProxySQL is not installed automatically as a dependency of the `percona-xtradb-cluster-client-8.0` package. You should install the `proxysql` package separately.

- On Debian or Ubuntu:

```
root@proxysql:~# apt-get install percona-xtradb-cluster-client
root@proxysql:~# apt-get install proxysql2
```

- On Red Hat Enterprise Linux or CentOS:

```
[root@proxysql ~]# yum install Percona-XtraDB-Cluster-client-80
[root@proxysql ~]# yum install proxysql2
```

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 -u admin -padmin -h 127.0.0.1 -P 6032

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.5.30 (ProxySQL Admin Module)

Copyright (c) 2009-2020 Percona LLC and/or its affiliates
Copyright (c) 2000, 2020, 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 source and replicas, 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,
↪'192.168.70.71',3306);
mysql@proxysql> INSERT INTO mysql_servers(hostgroup_id, hostname, port) VALUES (0,
↪'192.168.70.72',3306);
mysql@proxysql> INSERT INTO mysql_servers(hostgroup_id, hostname, port) VALUES (0,
↪'192.168.70.73',3306);
```

To see the nodes:

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

+--------------+---------------+------+--------+--------+-------------+---------------
↪--+---------------------+---------+----------------+---------+
| hostgroup_id | hostname      | port | status | weight | compression | max_
↪connections | max_replication_lag | use_ssl | max_latency_ms | comment |
+--------------+---------------+------+--------+--------+-------------+---------------
↪--+---------------------+---------+----------------+---------+
| 0            | 192.168.70.71 | 3306 | ONLINE | 1      | 0           | 1000
↪  | 0                   | 0       | 0              |         |
| 0            | 192.168.70.72 | 3306 | ONLINE | 1      | 0           | 1000
↪  | 0                   | 0       | 0              |         |
```

```
| 0            | 192.168.70.73 | 3306 | ONLINE | 1       | 0           | 1000
↪   | 0                   | 0       | 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 WITH mysql_native_password by '$3Kr
↪$t';
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_us
↪DESC LIMIT 6;
+---------------+------+------------------+----------------------+---------------+
| hostname      | port | time_start_us    | connect_success_time | connect_error |
+---------------+------+------------------+----------------------+---------------+
| 192.168.70.71 | 3306 | 1469635762434625 | 1695                 | NULL          |
| 192.168.70.72 | 3306 | 1469635762434625 | 1779                 | NULL          |
| 192.168.70.73 | 3306 | 1469635762434625 | 1627                 | NULL          |
| 192.168.70.71 | 3306 | 1469635642434517 | 1557                 | NULL          |
| 192.168.70.72 | 3306 | 1469635642434517 | 2737                 | NULL          |
| 192.168.70.73 | 3306 | 1469635642434517 | 1447                 | NULL          |
+---------------+------+------------------+----------------------+---------------+
6 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM monitor.mysql_server_ping_log ORDER BY time_start_us DESC LIMIT
↪6;
+---------------+------+------------------+-------------------+------------+
| hostname      | port | time_start_us    | ping_success_time | ping_error |
+---------------+------+------------------+-------------------+------------+
| 192.168.70.71 | 3306 | 1469635762416190 | 948               | NULL       |
| 192.168.70.72 | 3306 | 1469635762416190 | 803               | NULL       |
| 192.168.70.73 | 3306 | 1469635762416190 | 711               | NULL       |
| 192.168.70.71 | 3306 | 1469635702416062 | 783               | NULL       |
| 192.168.70.72 | 3306 | 1469635702416062 | 631               | NULL       |
```

```
| 192.168.70.73 | 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 and save these changes to disk (ensuring that they persist after ProxySQL shuts down):

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

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

```
root@proxysql:~# mysql -u sbuser -psbpass -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: 5.5.30 (ProxySQL)

Copyright (c) 2009-2020 Percona LLC and/or its affiliates
Copyright (c) 2000, 2020, 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'@'192.168.70.74' IDENTIFIED BY 'sbpass';
Query OK, 0 rows affected (0.01 sec)

mysql@pxc3> GRANT ALL ON *.* TO 'sbuser'@'192.168.70.74';
Query OK, 0 rows affected (0.00 sec)
```

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_10ms | cnt_50ms | cnt_100ms | cnt_500ms | cnt_1s | cnt_5s | cnt_10s |␣
↪cnt_INFs |
+-------------------+---------------+-----------+-----------+-----------+---------+---
↪------+----------+----------+-----------+-----------+--------+--------+---------+---
↪-------+
| ALTER_TABLE       | 0             | 0         | 0         | 0         | 0       | 0       |␣
↪       | 0        | 0        | 0         | 0         | 0      | 0      | 0       | 0       |␣
↪        |
| ANALYZE_TABLE     | 0             | 0         | 0         | 0         | 0       | 0       |␣
↪       | 0        | 0        | 0         | 0         | 0      | 0      | 0       | 0       |␣
↪        |
| BEGIN             | 2212625       | 3686      | 55        | 2162      | 899     |␣
↪569     | 1        | 0        | 0         | 0         | 0      | 0      | 0       |␣
↪0        |
| CHANGE_MASTER     | 0             | 0         | 0         | 0         | 0       | 0       |␣
↪       | 0        | 0        | 0         | 0         | 0      | 0      | 0       | 0       |␣
↪        |
| COMMIT            | 21522591      | 3628      | 0         | 0         | 0       |␣
↪1765    | 1590     | 272      | 1         | 0         | 0      | 0      | 0       |␣
↪0        |
| CREATE_DATABASE   | 0             | 0         | 0         | 0         | 0       | 0       |␣
↪       | 0        | 0        | 0         | 0         | 0      | 0      | 0       | 0       |␣
↪        |
| CREATE_INDEX      | 0             | 0         | 0         | 0         | 0       | 0       |␣
↪       | 0        | 0        | 0         | 0         | 0      | 0      | 0       | 0       |␣
↪        |
...
| DELETE            | 2904130       | 3670      | 35        | 1546      | 1346    |␣
↪723     | 19       | 1        | 0         | 0         | 0      | 0      | 0       |␣
↪0        |
| DESCRIBE          | 0             | 0         | 0         | 0         | 0       | 0       |␣
↪       | 0        | 0        | 0         | 0         | 0      | 0      | 0       | 0       |␣
↪        |
...
| INSERT            | 19531649      | 3660      | 39        | 1588      | 1292    |␣
↪723     | 12       | 2        | 0         | 1         | 0      | 1      | 2       |␣
↪0        |
...
| SELECT            | 35049794      | 51605     | 501       | 26180     | 16606   |␣
↪8241    | 70       | 3        | 4         | 0         | 0      | 0      | 0       |␣
↪0        |
| SELECT_FOR_UPDATE | 0             | 0         | 0         | 0         | 0       | 0       |␣
↪       | 0        | 0        | 0         | 0         | 0      | 0      | 0       | 0       |␣
↪        |
...
| UPDATE            | 6402302       | 7367      | 75        | 2503      | 3020    |␣
↪1743    | 23       | 3        | 0         | 0         | 0      | 0      | 0       |␣
↪0        |
| USE               | 0             | 0         | 0         | 0         | 0       | 0       |␣
↪       | 0        | 0        | 0         | 0         | 0      | 0      | 0       | 0       |␣
↪        |
| SHOW              | 19691         | 2         | 0         | 0         | 0       | 0       |␣
↪       | 1        | 1        | 0         | 0         | 0      | 0      | 0       | 0       |␣
↪        |
| UNKNOWN           | 0             | 0         | 0         | 0         | 0       | 0       |␣
↪       | 0        | 0        | 0         | 0         | 0      | 0      | 0       | 0       |␣
↪        |
```

```
+------------------+--------------+-----------+-----------+-----------+---------+---
↪------+----------+----------+-----------+-----------+--------+--------+---------+---
↪-------+
45 rows in set (0.00 sec)
```

Automatic failover

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            | 192.168.70.71 | 3306 | ONLINE |
| 0            | 192.168.70.72 | 3306 | ONLINE |
| 0            | 192.168.70.73 | 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            | 192.168.70.71 | 3306 | ONLINE       |
| 0            | 192.168.70.72 | 3306 | ONLINE       |
| 0            | 192.168.70.73 | 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            | 192.168.70.71 | 3306 | ONLINE |
| 0            | 192.168.70.72 | 3306 | ONLINE |
| 0            | 192.168.70.73 | 3306 | ONLINE |
+--------------+---------------+------+--------+
3 rows in set (0.00 sec)
```

Assisted Maintenance Mode

Usually, to take a node down for maintenance, you need to identify that node, update its status in ProxySQL to `OFFLINE_SOFT`, wait for ProxySQL to divert traffic from this node, and then initiate the shutdown or perform maintenance tasks. Percona XtraDB Cluster includes a special *maintenance mode* for nodes that enables you to take a node down without adjusting ProxySQL manually. The mode is controlled using the *pxc_maint_mode* variable, which is monitored by ProxySQL and can be set to one of the following values:

- `DISABLED`: This is the default state that tells ProxySQL to route traffic to the node as usual.
- `SHUTDOWN`: This state is set automatically when you initiate node shutdown.

  You may need to shut down a node when upgrading the OS, adding resources, changing hardware parts, relocating the server, etc.

  When you initiate node shutdown, Percona XtraDB Cluster does not send the signal immediately. Intead, it changes the state to `pxc_maint_mode=SHUTDOWN` and waits for a predefined period (10 seconds by default). When ProxySQL detects that the mode is set to `SHUTDOWN`, it changes the status of this node to `OFFLINE_SOFT`, which stops creation of new connections for the node. After the transition period, any long-running transactions that are still active are aborted.
- `MAINTENANCE`: You can change to this state if you need to perform maintenace on a node without shutting it down.

  You may need to isolate the node for some time, so that it does not receive traffic from ProxySQL while you resize the buffer pool, truncate the undo log, defragment or check disks, etc.

  To do this, manually set `pxc_maint_mode=MAINTENANCE`. Control is not returned to the user for a predefined period (10 seconds by default). When ProxySQL detects that the mode is set to `MAINTENANCE`, it stops routing traffic to the node. Once control is returned, you can perform maintenance activity.

  Note: Any data changes will still be replicated across the cluster.

  After you finish maintenance, set the mode back to `DISABLED`. When ProxySQL detects this, it starts routing traffic to the node again.

You can increase the transition period using the *pxc_maint_transition_period* variable to accomodate for long-running transactions. If the period is long enough for all transactions to finish, there should hardly be any disruption in cluster workload.

During the transition period, the node continues to receive existing write-set replication traffic, ProxySQL avoids openning new connections and starting transactions, but the user can still open conenctions to monitor status.

Note: If you increase the transition period, the packaging script may determine it as a server stall.

Related sections

- *Setting up a testing environment with ProxySQL*

CHAPTER

THIRTYTHREE

THE `proxysql-admin` TOOL WITH PROXYSQL V2

With ProxySQL 2.0.3, the **`proxysql-admin`** tool now uses the native ProxySQL support for Percona XtraDB Cluster and does not require custom bash scripts to keep track of Percona XtraDB Cluster status. As a result, `proxysql_galera_checker` and `proxysql_node_monitor` have been removed.

Summary of Changes in `proxysql-admin` Tool with ProxySQL v2

- *Added Features*
- *Changed Features*
- *Removed Features*
- *Limitations*

Added Features

- New option `--use-ssl` to use SSL for connections between ProxySQL and the backend database servers
- New option `--max-transactions-behind` to determine the maximum number of writesets that can be queued before the node is SHUNNED to avoid stale reads. The default value is 100
- New operation `--update-cluster` to update the cluster membership by adding server nodes as found. (Note that nodes are added but not removed). The `--writer-hg` option may be used to specify which galera hostgroup to update. The `--remove-all-servers` option instructs to remove all servers from the mysql_servers table before updating the cluster.
- Hostgroups can be specified on the command-line: `--writer-hg`, `--reader-hg`, `--backup-writer-hg`, and `--offline-hg`. Previously, these host groups were only read from the configuration file.
- The `--enable` and `--update-cluster` options used simultaneously have special meaning. If the cluster has not been enabled, then `--enable` is run. If the cluster has already been enabled, then `--update-cluster` is run.
- New command `--is-enabled` to see if a cluster has been enabled. This command checks for the existence of a row in the mysql_galera_hostgroups table. The `--writer-hg` option may be used to specify the writer hostgroup used to search the mysql_galera_hostgroups table.
- New command `--status` to display galera hostgroup information. This command lists all rows in the current `mysql_galera_hostgroups` table as well as all servers that belong to these hostgroups. With the `--writer-hg` option, only the information for the galera hostgroup with that writer hostgroup is displayed.

Changed Features

- Setting `--node-check-interval` changes the ProxySQL global variable `mysql-monitor_galera_healthcheck_interval`. Note that this is a global variable, not a per-cluster variable.
- The option `--write-node` takes only a single address as a parameter. In the singlewrite mode we only set the weight if `--write-node` specifies *address:port*. A priority list of addresses is no longer accepted.
- The option `--writers-as-readers` option accepts a different set of values. Due to changes in the behavior of ProxySQL between version 1.4 and version 2.0 related to Galera support, the values of `--writers-as-readers` have been changed. This option accepts the following values: yes, no, and backup.

  yes writers, backup-writers, and read-only nodes can act as readers.

  no only read-only nodes can act as readers.

  backup only backup-writers can act as readers.
- The commands `--syncusers`, `--sync-multi-cluster-users`, `--adduser`, and `--disable` can use the `--writer-hg` option.
- The command `--disable` removes all users associated with the galera cluster hostgroups. Previously, this command only removed the users with the **CLUSTER_APP_USERNAME**.
- The command `--disable` accepts the `--writer-hg` option to disable the Galera cluster associated with that hostgroup overriding the value specified in the configuration file.

Removed Features

- Asynchronous replica reader support has been removed: the `--include-slaves` option is not supported.
- A list of nodes in the priority order is not supported in ProxySQL v2. Only a single node is supported at this time.
- Since the galera_proxysql_checker and galera_node_monitor scripts are no longer run in the scheduler, automatic cluster membership updates are not supported.
- Checking the pxc_maint_mode variable is no longer supported
- Using desynced nodes if no other nodes are available is no longer supported.
- The server status is no longer maintained in the mysql_servers table.

Limitations

- With `--writers-as-readers=backup` read-only nodes are not allowed. This a limitation of ProxySQL 2.0. Note that backup is the default value of `--writers-as-readers` when `--mode=singlewrite`

Installing ProxySQL v2

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

- Installing on Debian or Ubuntu:

```
$ sudo apt-get install proxysql2
```

- Installing on Red Hat Enterprise Linux or CentOS:

```
$ sudo yum install proxysql2
```

Alternatively, you can download the packages from https://www.percona.com/downloads/proxysql2/.

To start ProxySQL, run the following command:

```
$ sudo service proxysql2 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.

Automatic Configuration

The `proxysql2` package from Percona includes the `proxysql-admin` tool for configuring Percona XtraDB Cluster nodes with ProxySQL.

Before using the `proxysql-admin` tool, ensure that ProxySQL and Percona XtraDB Cluster nodes you want to add are running. For security purposes, please ensure to change the default user settings in the ProxySQL configuration file.

Important: The `proxysql-admin` tool can only be used for *initial* ProxySQL configuration.

The *ProxySQL Admin* (**`proxysql-admin`** tool) is specially developed by Percona to automate this configuration. Bug reports and feature proposals are welcome in the ProxySQL Admin issue tracking system.

To view the usage information, run `proxysql-admin` without any options:

```
Usage: proxysql-admin [ options ]

Options:

--config-file=<config-file>         Read login credentials from a configuration file
                                    (command line options override any configuration␣
↪file values)

--writer-hg=<number>                The hostgroup that all traffic will be sent to
                                    by default. Nodes that have 'read-only=0' in MySQL
                                    will be assigned to this hostgroup.
--backup-writer-hg=<number>         If the cluster has multiple nodes with 'read-only=0
↪'
                                    and max_writers set, then additional nodes (in␣
↪excess
                                    of max_writers), will be assigned to this␣
↪hostgroup.
--reader-hg=<number>                The hostgroup that read traffic should be sent to.
                                    Nodes with 'read-only=0' in MySQL will be assigned
                                    to this hostgroup.
--offline-hg=<number>               Nodes that are determined to be OFFLINE will
                                    assigned to this hostgroup.
```

```
--proxysql-datadir=<datadir>        Specify the proxysql data directory location
--proxysql-username=<user_name>     ProxySQL service username
--proxysql-password[=<password>]    ProxySQL service password
--proxysql-port=<port_num>          ProxySQL service port number
--proxysql-hostname=<host_name>     ProxySQL service hostname

--cluster-username=<user_name>      Percona XtraDB Cluster node username
--cluster-password[=<password>]     Percona XtraDB Cluster node password
--cluster-port=<port_num>           Percona XtraDB Cluster node port number
--cluster-hostname=<host_name>      Percona XtraDB Cluster node hostname

--cluster-app-username=<user_name> Percona XtraDB Cluster node application username
--cluster-app-password[=<password>] Percona XtraDB Cluster node application passwrod
--without-cluster-app-user          Configure Percona XtraDB Cluster without␣
↪application user

--monitor-username=<user_name>      Username for monitoring Percona XtraDB Cluster␣
↪nodes through ProxySQL
--monitor-password[=<password>]     Password for monitoring Percona XtraDB Cluster␣
↪nodes through ProxySQL
--use-existing-monitor-password     Do not prompt for a new monitor password if one is␣
↪provided.

--node-check-interval=<NUMBER>      The interval at which the proxy should connect
                                    to the backend servers in order to monitor the
                                    Galera staus of a node (in milliseconds).
                                    (default: 5000)
--mode=[loadbal|singlewrite]        ProxySQL read/write configuration mode
                                    currently supporting: 'loadbal' and 'singlewrite'
                                    (default: 'singlewrite')
--write-node=<IPADDRESS>:<PORT>     Specifies the node that is to be used for
                                    writes for singlewrite mode.  If left unspecified,
                                    the cluster node is then used as the write node.
                                    This only applies when 'mode=singlewrite' is used.
--max-connections=<NUMBER>          Value for max_connections in the mysql_servers␣
↪table.
                                    This is the maximum number of connections that
                                    ProxySQL will open to the backend servers.
                                    (default: 1000)
--max-transactions-behind=<NUMBER> Determines the maximum number of writesets a node
                                    can have queued before the node is SHUNNED to avoid
                                    stale reads.
                                    (default: 100)
--use-ssl=[yes|no]                  If set to 'yes', then connections between ProxySQL
                                    and the backend servers will use SSL.
                                    (default: no)
--writers-are-readers=[yes|no|backup]
                                    If set to 'yes', then all writers (backup-writers␣
↪also)
                                    are added to the reader hostgroup.
                                    If set to 'no', then none of the writers (backup-
↪writers also)
                                    will be added to the reader hostgroup.
                                    If set to 'backup', then only the backup-writers
                                    will be added to the reader hostgroup.
                                    (default: backup)
--remove-all-servers                When used with --update-cluster, this will remove␣
↪all
```

```
                                servers belonging to the current cluster before
                                updating the list.
--debug                         Enables additional debug logging.
--help                          Dispalys this help text.

These options are the possible operations for proxysql-admin.
One of the options below must be provided.
--adduser                       Adds the Percona XtraDB Cluster application user␣
↪to the ProxySQL database
--disable, -d                   Remove any Percona XtraDB Cluster configurations␣
↪from ProxySQL
--enable, -e                    Auto-configure Percona XtraDB Cluster nodes into␣
↪ProxySQL
--update-cluster                Updates the cluster membership, adds new cluster␣
↪nodes
                                to the configuration.
--update-mysql-version          Updates the `mysql-server_version` variable in␣
↪ProxySQL with the version
                                from a node in the cluster.
--quick-demo                    Setup a quick demo with no authentication
--syncusers                     Sync user accounts currently configured in MySQL␣
↪to ProxySQL
                                May be used with --enable.
                                (deletes ProxySQL users not in MySQL).
                                See :ref:`pxc.proxysql.v2.admin-tool.syncusers`␣
↪for more information
--sync-multi-cluster-users      Sync user accounts currently configured in MySQL␣
↪to ProxySQL
                                May be used with --enable.
                                (doesn't delete ProxySQL users not in MySQL)
--add-query-rule                Create query rules for synced mysql user. This is␣
↪applicable only
                                for singlewrite mode and works only with --
↪syncusers
                                and --sync-multi-cluster-users options
--is-enabled                    Checks if the current configuration is enabled in␣
↪ProxySQL.
--status                        Returns a status report on the current␣
↪configuration.
                                If "--writer-hg=<NUM>" is specified, than the
                                data corresponding to the galera cluster with that
                                writer hostgroup is displayed. Otherwise,␣
↪information
                                for all clusters will be displayed.
--force                         This option will skip existing configuration␣
↪checks in mysql_servers,
                                mysql_users and mysql_galera_hostgroups tables.␣
↪This option will only
                                work with ``proxysql-admin --enable``.
--disable-updates               Disable admin updates for ProxySQL cluster for the
                                current operation. The default is to not change the
                                admin variable settings.  If this option is␣
↪specifed,
                                these options will be set to false.
                                (default: updates are not disabled)
--version, -v                   Prints the version info
```

Preparing Configuration File

It is recommended to provide the connection and authentication information in the ProxySQL configuration file (`/etc/proxysql-admin.cnf`). Do not specify this information on the command line.

By default, the configuration file contains the following:

```
# proxysql admin interface credentials.
export PROXYSQL_DATADIR='/var/lib/proxysql'
export PROXYSQL_USERNAME='admin'
export PROXYSQL_PASSWORD='admin'
export PROXYSQL_HOSTNAME='localhost'
export PROXYSQL_PORT='6032'

# PXC admin credentials for connecting to pxc-cluster-node.
export CLUSTER_USERNAME='admin'
export CLUSTER_PASSWORD='admin'
export CLUSTER_HOSTNAME='localhost'
export CLUSTER_PORT='3306'

# proxysql monitoring user. proxysql admin script will create this user in pxc to␣
↪monitor pxc-nodes.
export MONITOR_USERNAME="monitor"
export MONITOR_PASSWORD="monit0r"

# Application user to connect to pxc-node through proxysql
export CLUSTER_APP_USERNAME="proxysql_user"
export CLUSTER_APP_PASSWORD="passw0rd"

# ProxySQL hostgroup IDs
export WRITER_HOSTGROUP_ID='10'
export READER_HOSTGROUP_ID='11'
export BACKUP_WRITER_HOSTGROUP_ID='12'
export OFFLINE_HOSTGROUP_ID='13'

# ProxySQL read/write configuration mode.
export MODE="singlewrite"

# max_connections default (used only when INSERTing a new mysql_servers entry)
export MAX_CONNECTIONS="1000"

# Determines the maximum number of writesets a node can have queued
# before the node is SHUNNED to avoid stale reads.
export MAX_TRANSACTIONS_BEHIND=100

# Connections to the backend servers (from ProxySQL) will use SSL
export USE_SSL="no"

# Determines if a node should be added to the reader hostgroup if it has
# been promoted to the writer hostgroup.
# If set to 'yes', then all writers (including backup-writers) are added to
# the read hostgroup.
# If set to 'no', then none of the writers (including backup-writers) are added.
# If set to 'backup', then only the backup-writers will be added to
# the read hostgroup.
export WRITERS_ARE_READERS="backup"
```

Running `proxysql-admin` tool

It is recommended to *change default ProxySQL credentials* before running ProxySQL in production. Make sure that you provide ProxySQL location and credentials in the configuration file.

Provide superuser credentials for one of the Percona XtraDB Cluster nodes. The `proxysql-admin` script will detect other nodes in the cluster automatically.

- *–enable*
- *–disable*
- *–adduser*
- *–syncusers*
- *–sync-multi-cluster-users*
- *–add-query-rule*
- *–quick-demo*
- *–update-cluster*
- *–is-enabled*
- *–status*
- *–force*
- *–update-mysql-version*

–enable

This option creates the entry for the Galera hostgroups and adds the Percona XtraDB Cluster nodes to ProxySQL.

It will also add two new users into the Percona XtraDB Cluster with the USAGE privilege; one is for monitoring the cluster nodes through ProxySQL, and another is for connecting to the PXC Cluster node via the ProxySQL console.

```
$ sudo proxysql-admin --config-file=/etc/proxysql-admin.cnf --enable
```

Output

```
This script will assist with configuring ProxySQL for use with
Percona XtraDB Cluster (currently only PXC in combination
with ProxySQL is supported)

ProxySQL read/write configuration mode is singlewrite

Configuring the ProxySQL monitoring user.
ProxySQL monitor user name as per command line/config-file is monitor

User 'monitor'@'127.%' has been added with USAGE privileges

Configuring the Percona XtraDB Cluster application user to connect through ProxySQL
Percona XtraDB Cluster application user name as per command line/config-file is␣
↪proxysql_user
```

```
Percona XtraDB Cluster application user 'proxysql_user'@'127.%' has been added with␣
↪ALL privileges, this user is created for testing purposes
Adding the Percona XtraDB Cluster server nodes to ProxySQL

Write node info
+-----------+--------------+-------+--------+
| hostname  | hostgroup_id | port  | weight |
+-----------+--------------+-------+--------+
| 127.0.0.1 | 10           | 26100 | 1000   |
+-----------+--------------+-------+--------+

ProxySQL configuration completed!

ProxySQL has been successfully configured to use with Percona XtraDB Cluster

You can use the following login credentials to connect your application through␣
↪ProxySQL

mysql --user=proxysql_user -p --host=localhost --port=6033 --protocol=tcp
```

You can use the following login credentials to connect your application through ProxySQL:

```
$ mysql --user=proxysql_user -p --host=127.0.0.1 --port=6033 --protocol=tcp
mysql> select hostgroup_id,hostname,port,status from runtime_mysql_servers;
```

Example of output

```
+--------------+-----------+-------+--------+
| hostgroup_id | hostname  | port  | status |
+--------------+-----------+-------+--------+
| 10           | 127.0.0.1 | 25000 | ONLINE |
| 11           | 127.0.0.1 | 25100 | ONLINE |
| 11           | 127.0.0.1 | 25200 | ONLINE |
| 12           | 127.0.0.1 | 25100 | ONLINE |
| 12           | 127.0.0.1 | 25200 | ONLINE |
+--------------+-----------+-------+--------+
5 rows in set (0.00 sec)
```

```
mysql> select * from mysql_galera_hostgroups\G
```

Output

```
writer_hostgroup: 10
backup_writer_hostgroup: 12
reader_hostgroup: 11
offline_hostgroup: 13
active: 1
max_writers: 1
writer_is_also_reader: 2
max_transactions_behind: 100
comment: NULL
1 row in set (0.00 sec)
```

The `--enable` command may be used together with `--update-cluster`. If the cluster has not been setup, then the enable function will be run. If the cluster has been setup, then the update cluster function will be run.

–disable

This option will remove Percona XtraDB Cluster nodes from ProxySQL and stop the ProxySQL monitoring daemon.

```
$ proxysql-admin --config-file=/etc/proxysql-admin.cnf --disable
```

Output

```
Removing cluster application users from the ProxySQL database.
Removing cluster nodes from the ProxySQL database.
Removing query rules from the ProxySQL database if any.
Removing the cluster from the ProxySQL database.
ProxySQL configuration removed!
```

A specific galera cluster can be disabled by using the –writer-hg option with `--disable`.

–adduser

This option will aid with adding the Cluster application user to the ProxySQL database for you

```
$ proxysql-admin --config-file=/etc/proxysql-admin.cnf --adduser
```

Output

```
Adding Percona XtraDB Cluster application user to ProxySQL database
Enter Percona XtraDB Cluster application user name: root
Enter Percona XtraDB Cluster application user password:
Added Percona XtraDB Cluster application user to ProxySQL database!
```

–syncusers

This option synchronizes user accounts currently configured in Percona XtraDB Cluster with the ProxySQL database except password-less users and admin users.

Important: This option does not work if the Percona XtraDB Cluster user is created using the `caching_sha2_password` plugin (used by default in Percona XtraDB Cluster 8.0). Create a mysql user using the `mysql_native_password` authentication plugin.

```
mysql> CREATE USER 'proxysql'@'%' IDENTIFIED WITH mysql_native_password by '$3Kr$t';
```

See also:

***MySQL* Documentation: CREATE USER statement** https://dev.mysql.com/doc/refman/8.0/en/create-user.html

It also deletes ProxySQL users not in Percona XtraDB Cluster from the ProxySQL database.

```
$ /usr/bin/proxysql-admin --syncusers
```

Output

```
Syncing user accounts from Percona XtraDB Cluster to ProxySQL
Synced Percona XtraDB Cluster users to the ProxySQL database!
```

From ProxySQL DB

```
mysql> select username from mysql_users;
```

Output

```
+---------------+
| username      |
+---------------+
| monitor       |
| one           |
| proxysql_user |
| two           |
+---------------+
4 rows in set (0.00 sec)
```

From PXC

```
mysql> select user,host from mysql.user where authentication_string!='' and user not␣
↪in ('admin','mysql.sys');
```

Output

```
+---------------+-------+
| user          | host  |
+---------------+-------+
| monitor       | 192.% |
| proxysql_user | 192.% |
| two           | %     |
| one           | %     |
+---------------+-------+
4 rows in set (0.00 sec)
```

–sync-multi-cluster-users

This option works in the same way as –syncusers but it does not delete ProxySQL users that are not present in the Percona XtraDB Cluster. It is to be used when syncing proxysql instances that manage multiple clusters.

–add-query-rule

Create query rules for synced mysql user. This is applicable only for singlewrite mode and works only with *–syncusers* and *–sync-multi-cluster-users* options.

```
Syncing user accounts from PXC to ProxySQL

Note : 'admin' is in proxysql admin user list, this user cannot be added to ProxySQL
-- (For more info, see https://github.com/sysown/proxysql/issues/709)
Adding user to ProxySQL: test_query_rule
Added query rule for user: test_query_rule

Synced PXC users to the ProxySQL database!
```

–quick-demo

This option is used to setup a dummy proxysql configuration.

```
$ sudo  proxysql-admin --quick-demo
```

Output

```
You have selected the dry test run mode. WARNING: This will create a test user (with␣
↪all privileges) in the Percona XtraDB Cluster & ProxySQL installations.
You may want to delete this user after you complete your testing!
Would you like to proceed with '--quick-demo' [y/n] ? y
Setting up proxysql test configuration!

Do you want to use the default ProxySQL credentials (admin:admin:6032:127.0.0.1) [y/
↪n] ? y
Do you want to use the default Percona XtraDB Cluster credentials (root::3306:127.0.0.
↪1) [y/n] ? n

Enter the Percona XtraDB Cluster username (super user): root
Enter the Percona XtraDB Cluster user password:
Enter the Percona XtraDB Cluster port: 25100
Enter the Percona XtraDB Cluster hostname: localhost

ProxySQL read/write configuration mode is singlewrite

Configuring ProxySQL monitoring user..

User 'monitor'@'127.%' has been added with USAGE privilege
Configuring the Percona XtraDB Cluster application user to connect through ProxySQL
Percona XtraDB Cluster application user 'pxc_test_user'@'127.%' has been added with␣
↪ALL privileges, this user is created for testing purposes
Adding the Percona XtraDB Cluster server nodes to ProxySQL

ProxySQL configuration completed!

ProxySQL has been successfully configured to use with Percona XtraDB Cluster

You can use the following login credentials to connect your application through␣
↪ProxySQL

mysql --user=pxc_test_user  --host=127.0.0.1 --port=6033 --protocol=tcp
```

```
mysql> select hostgroup_id,hostname,port,status from runtime_mysql_servers;
```

Output

```
+--------------+-----------+-------+--------+
| hostgroup_id | hostname  | port  | status |
+--------------+-----------+-------+--------+
| 10           | 127.0.0.1 | 25000 | ONLINE |
| 11           | 127.0.0.1 | 25100 | ONLINE |
| 11           | 127.0.0.1 | 25200 | ONLINE |
| 12           | 127.0.0.1 | 25100 | ONLINE |
| 12           | 127.0.0.1 | 25200 | ONLINE |
+--------------+-----------+-------+--------+
5 rows in set (0.00 sec)
```

–update-cluster

This option will check the Percona XtraDB Cluster to see if any new nodes have joined the cluster. If so, the new nodes are added to ProxySQL. Any offline nodes are not removed from the cluster by default.

If used with `--remove-all-servers`, then the server list for this configuration will be removed before running the update cluster function.

A specific galera cluster can be updated by using the `--writer-hg` option with `--update-cluster`. Otherwise the cluster specified in the config file will be updated.

If `--write-node` is used with `--update-cluster`, then that node will be made the writer node (by giving it a larger weight), if the node is in the server list and is ONLINE. This should only be used if the mode is _singlewrite_.

```
$ sudo proxysql-admin --update-cluster --writer-hg=10 --remove-all-servers
```

Output

```
Removing all servers from ProxySQL
Cluster node (127.0.0.1:25000) does not exist in ProxySQL, adding to the writer␣
↪hostgroup(10)
Cluster node (127.0.0.1:25100) does not exist in ProxySQL, adding to the writer␣
↪hostgroup(10)
Cluster node (127.0.0.1:25200) does not exist in ProxySQL, adding to the writer␣
↪hostgroup(10)
Waiting for ProxySQL to process the new nodes...

Cluster node info
+---------------+-------+-----------+-------+-----------+
| hostgroup     | hg_id | hostname  | port  | weight    |
+---------------+-------+-----------+-------+-----------+
| writer        | 10    | 127.0.0.1 | 25000 | 1000      |
| reader        | 11    | 127.0.0.1 | 25100 | 1000      |
| reader        | 11    | 127.0.0.1 | 25200 | 1000      |
| backup-writer | 12    | 127.0.0.1 | 25100 | 1000      |
| backup-writer | 12    | 127.0.0.1 | 25200 | 1000      |
```

```
+---------------+-------+-----------+------+------------+

Cluster membership updated in the ProxySQL database!
```

–is-enabled

This option will check if a galera cluster (specified by the writer hostgroup, either from `--writer-hg` or from the config file) has any active entries in the `mysql_galera_hostgroups` table in ProxySQL.

Value	Is returned if there is
0	An entry corresponding to the writer hostgroup and is set to *active* in ProxySQL.
1	No entry corresponding to the writer hostgroup.
2	An entry corresponding to the writer hostgroup but is not active.

```
$ sudo proxysql-admin --is-enabled --writer-hg=10
The current configuration has been enabled and is active

$ sudo proxysql-admin --is-enabled --writer-hg=20
ERROR (line:2925) : The current configuration has not been enabled
```

–status

If used with the `--writer-hg` option, this will display information about the given Galera cluster which uses that writer hostgroup. Otherwise it will display information about all Galera hostgroups (and their servers) being supported by this ProxySQL instance.

```
$ sudo proxysql-admin --status --writer-hg=10
```

Output

```
mysql_galera_hostgroups row for writer-hostgroup: 10
+--------+--------+---------------+---------+--------+-------------+------------------
↪-----+------------------+
| writer | reader | backup-writer | offline | active | max_writers | writer_is_also_
↪reader | max_trans_behind |
+--------+--------+---------------+---------+--------+-------------+------------------
↪-----+------------------+
| 10     | 11     | 12            | 13      | 1      | 1           | 2
↪     | 100              |
+--------+--------+---------------+---------+--------+-------------+------------------
↪-----+------------------+

mysql_servers rows for this configuration
+---------------+-------+-----------+-------+--------+-----------+----------+---------
↪+------------+
| hostgroup     | hg_id | hostname  | port  | status | weight    | max_conn | use_ssl
↪| gtid_port |
+---------------+-------+-----------+-------+--------+-----------+----------+---------
↪+------------+
| writer        | 10    | 127.0.0.1 | 25000 | ONLINE | 1000000   | 1000     | 0
↪| 0          |
| reader        | 11    | 127.0.0.1 | 25100 | ONLINE | 1000      | 1000     | 0
↪| 0          |
```

```
| reader         | 11    | 127.0.0.1 | 25200 | ONLINE | 1000       | 1000      | 0
↪| 0          |
| backup-writer | 12    | 127.0.0.1 | 25100 | ONLINE | 1000       | 1000      | 0
↪| 0          |
| backup-writer | 12    | 127.0.0.1 | 25200 | ONLINE | 1000       | 1000      | 0
↪| 0          |
+---------------+-------+-----------+-------+--------+-----------+-----------+---------
↪+-----------+
```

–force

This will skip existing configuration checks with the `--enable` option in *mysql_servers*, *mysql_users*, and *mysql_galera_hostgroups* tables.

–update-mysql-version

This option will updates mysql server version (specified by the writer hostgroup, either from `--writer-hg` or from the config file) in proxysql db based on online writer node.

```
$  sudo proxysql-admin --update-mysql-version --writer-hg=10
ProxySQL MySQL version changed to 5.7.26
```

Extra options

- *–mode*
- *–node-check-interval*
- *–write-node*

–mode

This option allows you to setup the read/write mode for PXC cluster nodes in the ProxySQL database based on the hostgroup. For now, the only supported modes are *loadbal* and *singlewrite*. *singlewrite* is the default mode, and it will configure Percona XtraDB Cluster to only accept writes on a single node only. Depending on the value of `--writers-are-readers`, the write node may accept read requests also.

All other remaining nodes will be read-only and will only receive read statements.

With the `--write-node` option we can control which node ProxySQL will use as the writer node. The writer node is specified as an address:port - **10.0.0.51:3306** If `--write-node` is used, the writer node is given a weight of **1000000** (the default weight is **1000**).

The mode *loadbal* on the other hand is a load balanced set of evenly weighted read/write nodes.

singlewrite mode setup:

```
$ sudo grep "MODE" /etc/proxysql-admin.cnf
$ export MODE="singlewrite"
$ sudo proxysql-admin --config-file=/etc/proxysql-admin.cnf --write-node=127.0.0.
↪1:25000 --enable
```

Output

```

```

ProxySQL read/write configuration mode is singlewrite [..] ProxySQL configuration completed!

```
mysql> select hostgroup_id,hostname,port,status from runtime_mysql_servers;
```

Output

```
+--------------+-----------+-------+--------+
| hostgroup_id | hostname  | port  | status |
+--------------+-----------+-------+--------+
| 10           | 127.0.0.1 | 25000 | ONLINE |
| 11           | 127.0.0.1 | 25100 | ONLINE |
| 11           | 127.0.0.1 | 25200 | ONLINE |
| 12           | 127.0.0.1 | 25100 | ONLINE |
| 12           | 127.0.0.1 | 25200 | ONLINE |
+--------------+-----------+-------+--------+
5 rows in set (0.00 sec)
```

loadbal mode setup

```
$ sudo proxysql-admin --config-file=/etc/proxysql-admin.cnf --mode=loadbal --enable
```

Output

```
This script will assist with configuring ProxySQL (currently only Percona XtraDB␣
↪cluster in combination with ProxySQL is supported)

ProxySQL read/write configuration mode is loadbal
[..]
ProxySQL has been successfully configured to use with Percona XtraDB Cluster

You can use the following login credentials to connect your application through␣
↪ProxySQL
```

```
$ mysql --user=proxysql_user --password=*****  --host=127.0.0.1 --port=6033 --
↪protocol=tcp
```

```
mysql> select hostgroup_id,hostname,port,status from runtime_mysql_servers;
```

Output

```
+--------------+-----------+-------+--------+
| hostgroup_id | hostname  | port  | status |
+--------------+-----------+-------+--------+
| 10           | 127.0.0.1 | 25000 | ONLINE |
| 10           | 127.0.0.1 | 25100 | ONLINE |
| 10           | 127.0.0.1 | 25200 | ONLINE |
+--------------+-----------+-------+--------+
3 rows in set (0.01 sec)
```

–node-check-interval

This option configures the interval for the cluster node health monitoring by ProxySQL (in milliseconds). This is a global variable and will be used by all clusters that are being served by this ProxySQL instance. This can only be used with `--enable`.

```
$ proxysql-admin --config-file=/etc/proxysql-admin.cnf --node-check-interval=5000 --
↪enable
```

–write-node

This option is used to choose which node will be the writer node when the mode is *singlewrite*. This option can be used with *–enable* and *–update-cluster*.

A single IP address and port combination is expected. For instance, "–write-node=127.0.0.1:3306"

ProxySQL Status

Simple script to dump ProxySQL config and stats

```
$ proxysql-status admin admin 127.0.0.1 6032
```

CHAPTER

THIRTYFOUR

SETTING UP A TESTING ENVIRONMENT WITH PROXYSQL

This section describes how to set up Percona XtraDB Cluster in a virtualized testing environment based on ProxySQL. To test the cluster, we will use the *sysbench* benchmark tool.

It is assumed that each Percona XtraDB Cluster node is installed on Amazon EC2 micro instances running CentOS 7. However, the information in this section should apply if you used another virtualization technology (for example, VirtualBox) with any Linux distribution.

Each of the tree Percona XtraDB Cluster nodes is installed on a separate virtual machine. One more virtual machine has ProxySQL, which redirects requests to the nodes.

Tip: Running ProxySQL 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 three cluster nodes, as described in *Configuring Percona XtraDB Cluster on CentOS*.
2. On the client node, install *ProxySQL* and `sysbench`:

```
$ yum -y install proxysql2 sysbench
```

3. When all cluster nodes are started, configure ProxySQL using the admin interface.

   Tip: 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.

   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 (assuming that ProxySQL IP is 192.168.70.74):

```
root@proxysql:~# mysql -u admin -padmin -h 127.0.0.1 -P 6032

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.5.30 (ProxySQL Admin Module)
```

```
Copyright (c) 2009-2020 Percona LLC and/or its affiliates
Copyright (c) 2000, 2020, 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>
```

To see the ProxySQL databases and tables use the `SHOW DATABASES` and `SHOW TABLES` commands:

```
mysql> SHOW DATABASES;
+-----+---------------+-------------------------------------+
| seq | name          | file                                |
+-----+---------------+-------------------------------------+
| 0   | main          |                                     |
| 2   | disk          | /var/lib/proxysql/proxysql.db       |
| 3   | stats         |                                     |
| 4   | monitor       |                                     |
| 5   | stats_monitor | /var/lib/proxysql/proxysql_stats.db |
+-----+---------------+-------------------------------------+
5 rows in set (0.00 sec)
```

```
mysql> SHOW TABLES;
+----------------------------------------------------+
| tables                                             |
+----------------------------------------------------+
| global_variables                                   |
| mysql_aws_aurora_hostgroups                        |
| mysql_collations                                   |
| mysql_firewall_whitelist_rules                     |
| mysql_firewall_whitelist_sqli_fingerprints         |
| mysql_firewall_whitelist_users                     |
| mysql_galera_hostgroups                            |
| mysql_group_replication_hostgroups                 |
| mysql_query_rules                                  |
| mysql_query_rules_fast_routing                     |
| mysql_replication_hostgroups                       |
| mysql_servers                                      |
| mysql_users                                        |
| proxysql_servers                                   |
| restapi_routes                                     |
| runtime_checksums_values                           |
| runtime_global_variables                           |
| runtime_mysql_aws_aurora_hostgroups                |
| runtime_mysql_firewall_whitelist_rules             |
| runtime_mysql_firewall_whitelist_sqli_fingerprints |
| runtime_mysql_firewall_whitelist_users             |
| runtime_mysql_galera_hostgroups                    |
| runtime_mysql_group_replication_hostgroups         |
| runtime_mysql_query_rules                          |
| runtime_mysql_query_rules_fast_routing             |
| runtime_mysql_replication_hostgroups               |
| runtime_mysql_servers                              |
| runtime_mysql_users                                |
| runtime_proxysql_servers                           |
```

```
| runtime_restapi_routes                             |
| runtime_scheduler                                  |
| scheduler                                          |
+----------------------------------------------------+
32 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 saving 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.

```
INSERT INTO mysql_servers (hostname,hostgroup_id,port,weight) VALUES ('192.168.70.71',
↪10,3306,1000);
INSERT INTO mysql_servers (hostname,hostgroup_id,port,weight) VALUES ('192.168.70.72',
↪10,3306,1000);
INSERT INTO mysql_servers (hostname,hostgroup_id,port,weight) VALUES ('192.168.70.73',
↪10,3306,1000);
```

ProxySQL v2.0 supports Percona XtraDB Cluster natlively. It uses the concept of *hostgroups* (see the value of *hostgroup_id* in the *mysql_servers* table) to group cluster nodes to balance the load in a cluster by routing different types of traffic to different groups.

This information is stored in the *[runtime_]mysql_galera_hostgroups* table.

Columns of the *[runtime_]mysql_galera_hostgroups* table

Column name	Description
writer_hostgroup	The ID of the hostgroup that refers to the WRITER node
backup_writer_hostgroup	The ID of the hostgroup that contains candidate WRITER servers
reader_hostgroup	The ID of the hostgroup that contains candidate READER servers
offline_hostgroup	The ID of the hostgroup that will eventually contain the WRITER node that will be put OFFLINE
active	*1* (Yes) to inidicate that this configuration should be used; *0* (No) - otherwise
max_writers	The maximum number of WRITER nodes that must operate simultaneously. For most cases, a reasonable value is *1*. The value in this column may not exceed the total number of nodes.
writer_is_also_reader	*1* (Yes) to keep the given node in both *reader_hostgroup* and *writer_hostgroup*. *0* (No) to remove the given node from *reader_hostgroup* if it already belongs to *writer_hostgroup*.
max_transactions_behind	As soon as the value of `wsrep_local_recv_queue` exceeds the number stored in this column the given node is set to *OFFLINE*. Set the value carefully based on the behaviour of the node.
comment	Helpful extra information about the given node

Make sure that the variable *mysql-server_version* refers to the correct version. For Percona XtraDB Cluster 8.0, set it to *8.0* accordingly:

```
mysql> UPDATE GLOBAL_VARIABLES
SET variable_value='8.0'
WHERE variable_name='mysql-server_version';

mysql> LOAD MYSQL SERVERS TO RUNTIME;
mysql> SAVE MYSQL SERVERS TO DISK;
```

See also:

***Percona* Blogpost: ProxySQL Native Support for Percona XtraDB Cluster (PXC)** https://www.percona.com/blog/2019/02/20/proxysql-native-support-for-percona-xtradb-cluster-pxc/

Given the nodes from the *mysql_servers* table, you may set up the hostgroups as follows:

```
mysql> INSERT INTO mysql_galera_hostgroups (
writer_hostgroup, backup_writer_hostgroup, reader_hostgroup,
offline_hostgroup, active, max_writers, writer_is_also_reader,
max_transactions_behind)
VALUES (10, 12, 11, 13, 1, 1, 2, 100);
```

This command configures ProxySQL as follows:

WRITER hostgroup hostgroup *10*

READER hostgroup hostgroup *11*

BACKUP WRITER hostgroup hostgroup *12*

OFFLINE hostgroup hostgroup *13*

Set up ProxySQL query rules for read/write split using the *mysql_query_rules* table:

```
mysql> INSERT INTO mysql_query_rules (
username,destination_hostgroup,active,match_digest,apply)
VALUES ('appuser',10,1,'^SELECT.*FOR UPDATE',1);

mysql> INSERT INTO mysql_query_rules (
username,destination_hostgroup,active,match_digest,apply)
VALUES ('appuser',11,1,'^SELECT ',1);
```

```
mysql> LOAD MYSQL QUERY RULES TO RUNTIME;
mysql> SAVE MYSQL QUERY RULES TO DISK;

mysql> select hostgroup_id,hostname,port,status,weight from runtime_mysql_servers;
+--------------+----------------+------+--------+--------+
| hostgroup_id | hostname       | port | status | weight |
+--------------+----------------+------+--------+--------+
| 10           | 192.168.70.73 | 3306  | ONLINE | 1000   |
| 11           | 192.168.70.72 | 3306  | ONLINE | 1000   |
| 11           | 192.168.70.71 | 3306  | ONLINE | 1000   |
| 12           | 192.168.70.72 | 3306  | ONLINE | 1000   |
| 12           | 192.168.70.71 | 3306  | ONLINE | 1000   |
+--------------+----------------+------+--------+--------+
5 rows in set (0.00 sec)
```

See also:

ProxySQL Blog: *MySQL* read/write split with ProxySQL https://proxysql.com/blog/configure-read-write-split/

ProxySQL Documentation: *mysql_query_rules* table https://github.com/sysown/proxysql/wiki/Main-(runtime)#mysql_query_rules

ProxySQL failover behavior

Notice that all servers were inserted into the *mysql_servers* table with the READER hostgroup set to *10* (see the value of the *hostgroup_id* column):

```
mysql> SELECT * FROM mysql_servers;

+--------------+---------------+------+--------+     +---------+
| hostgroup_id | hostname      | port | weight | ... | comment |
+--------------+---------------+------+--------+     +---------+
| 10           | 192.168.70.71 | 3306 | 1000   |     |         |
| 10           | 192.168.70.72 | 3306 | 1000   |     |         |
| 10           | 192.168.70.73 | 3306 | 1000   |     |         |
+--------------+---------------+------+--------+     +---------+
3 rows in set (0.00 sec)
```

This configuration implies that ProxySQL elects the writer automatically. If the elected writer goes offline, ProxySQL assigns another (failover). You might tweak this mechanism by assigning a higher weight to a selected node. ProxySQL directs all write requests to this node. However, it also becomes the mostly utilized node for reading requests. In case of a failback (a node is put back online), the node with the highest weight is automatically elected for write requests.

Creating a 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> CREATE USER 'proxysql'@'%' IDENTIFIED WITH mysql_native_password BY
↪'ProxySQLPa55';
mysql> GRANT USAGE ON *.* TO 'proxysql'@'%';
```

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

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

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

Saving and loading the configuration

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

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

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

```
mysql> SELECT * FROM monitor.mysql_server_connect_log ORDER BY time_start_us DESC␣
↪LIMIT 6;
+---------------+------+------------------+----------------------+---------------+
| hostname      | port | time_start_us    | connect_success_time | connect_error |
+---------------+------+------------------+----------------------+---------------+
| 192.168.70.71 | 3306 | 1469635762434625 | 1695                 | NULL          |
| 192.168.70.72 | 3306 | 1469635762434625 | 1779                 | NULL          |
| 192.168.70.73 | 3306 | 1469635762434625 | 1627                 | NULL          |
| 192.168.70.71 | 3306 | 1469635642434517 | 1557                 | NULL          |
| 192.168.70.72 | 3306 | 1469635642434517 | 2737                 | NULL          |
| 192.168.70.73 | 3306 | 1469635642434517 | 1447                 | NULL          |
+---------------+------+------------------+----------------------+---------------+
6 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM monitor.mysql_server_ping_log ORDER BY time_start_us DESC LIMIT␣
↪6;
+---------------+------+------------------+-------------------+------------+
| hostname      | port | time_start_us    | ping_success_time | ping_error |
+---------------+------+------------------+-------------------+------------+
| 192.168.70.71 | 3306 | 1469635762416190 | 948               | NULL       |
| 192.168.70.72 | 3306 | 1469635762416190 | 803               | NULL       |
| 192.168.70.73 | 3306 | 1469635762416190 | 711               | NULL       |
| 192.168.70.71 | 3306 | 1469635702416062 | 783               | NULL       |
| 192.168.70.72 | 3306 | 1469635702416062 | 631               | NULL       |
| 192.168.70.73 | 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> 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> INSERT INTO mysql_users (username,password) VALUES ('appuser','$3kRetp@$sW0rd
↪');
Query OK, 1 row affected (0.00 sec)
```

Note: ProxySQL currently doesn't encrypt passwords.

See also:

More information about password encryption in ProxySQL

Load the user into runtime space and save these changes to disk (ensuring that they persist after ProxySQL shuts down):

```
mysql> LOAD MYSQL USERS TO RUNTIME;
mysql> SAVE MYSQL USERS TO DISK;
```

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

```
root@proxysql:~# mysql -u appuser -p$3kRetp@$sW0rd -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: 5.5.30 (ProxySQL)

Copyright (c) 2009-2020 Percona LLC and/or its affiliates
Copyright (c) 2000, 2020, 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> CREATE USER 'appuser'@'192.168.70.74'
IDENTIFIED WITH mysql_native_password by '$3kRetp@$sW0rd';

mysql> GRANT ALL ON *.* TO 'appuser'@'192.168.70.74';
```

Testing the cluster with the *sysbench* benchmark tool

After you set up Percona XtraDB Cluster in your testing environment, you can test it using the `sysbench` benchmarking tool.

1. Create a database (*sysbenchdb* in this example; you can use a different name):

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

2. Populate the table with data for the benchmark. Note that you should pass the database you have created as the value of the *–mysql-db* parameter, and the name of the user who has full access to this database as the value of the *–mysql-user* parameter:

```
$ sysbench /usr/share/sysbench/oltp_insert.lua --mysql-db=sysbenchdb \
--mysql-host=127.0.0.1 --mysql-port=6033 --mysql-user=appuser \
--mysql-password=$3kRetp@$sW0rd --db-driver=mysql --threads=10 --tables=10 \
--table-size=1000 prepare
```

3. Run the benchmark on port 6033:

```
$ sysbench /usr/share/sysbench/oltp_read_write.lua --mysql-db=sysbenchdb \
--mysql-host=127.0.0.1 --mysql-port=6033 --mysql-user=appuser \
--mysql-password=$3kRetp@$sW0rd --db-driver=mysql --threads=10 --tables=10 \
--skip-trx=true --table-size=1000 --time=100 --report-interval=10 run
```

Related sections and additional reading

- *Load balancing with ProxySQL*
- *Configuring Percona XtraDB Cluster on CentOS*
- Percona Blogpost: ProxySQL Native Support for Percona XtraDB Cluster (PXC)
- Github repository for the sysbench benchmarking tool

Part VII

Reference

CHAPTER

THIRTYFIVE

PERCONA XTRADB CLUSTER 8.0 RELEASE NOTES

Percona XtraDB Cluster 8.0.20-11.3

Date October 22, 2020

Installation Installing Percona XtraDB Cluster

Bugs Fixed

- PXC-3456: Allow specific characters in SST method names and SST request data.

Percona XtraDB Cluster 8.0.20-11.2

Date October 9, 2020

Installation Installing Percona XtraDB Cluster

This release fixes the security vulnerability CVE-2020-15180

Percona XtraDB Cluster 8.0.20-11

Date October 1, 2020

Installation Installing Percona XtraDB Cluster

Improvements

- PXC-2603: Update Index for PXC status variables - apply consistent definitions

Bugs Fixed

- PXC-3159: Modify error handling to close the communication channels and abort the joiner node when donor crashes (previously was Known Issue)
- PXC-3352: Modify wsrep_row_upd_check_foreign_constraints() to remove the check for DELETE
- PXC-3371: Fix Directory creation in build-binary.sh
- PXC-3370: Provide binary tarball with shared libs and glibc suffix & minimal tarballs

- PXC-3360: Update sysbench commands in PXC-ProxySQL configuration doc page
- PXC-3312: Prevent cleanup of statement diagnostic area in case of transaction replay.
- PXC-3167: Correct GCache buffer repossession processing
- PXC-3347: Modify PERCONA_SERVER_EXTENSION for bintarball and modify MYSQL_SERVER_SUFFIX

Known Issues (unfixed problems that you should be aware of)

- PXC-3039: No useful error messages if an SSL-disabled node tries to join SSL-enabled cluster
- PXC-3092: Log warning at startup if keyring is specified but cluster traffic encryption is turned off
- PXC-3093: Garbd logs Completed SST Transfer Incorrectly (Timing is not correct)

Percona XtraDB Cluster 8.0.19-10

Date June 18, 2020

Installation Installing Percona XtraDB Cluster

Improvements

- PXC-2189: Modify Reference Architecture for Percona XtraDB Cluster (PXC) to include ProxySQL
- PXC-3182: Modify processing to not allow writes on 8.0 nodes while 5.7 nodes are still on the cluster
- PXC-3187: Add dependency package installation note in PXC binary tarball installation doc.
- PXC-3138: Document mixed cluster write (PXC8 while PXC5.7 nodes are still part of the cluster) should not be completed.
- PXC-3066: Document that pxc-encrypt-cluster-traffic=OFF is not just about traffic encryption
- PXC-2993: Document the dangers of running with strict mode disabled and Group Replication at the same time
- PXC-2980: Modify Documentation to include AutoStart Up Process after Installation
- PXC-2604: Modify garbd processing to support Operator

Bugs Fixed

- PXC-3298: Correct galera_var_reject_queries test to remove display value width
- PXC-3320: Correction on PXC installation doc
- PXC-3270: Modify wsrep_ignore_apply_errors variable default to restore 5.x behavior
- PXC-3179: Correct replication of CREATE USER ... RANDOM PASSWORD
- PXC-3080: Modify to process the ROTATE_LOG_EVENT synchronously to perform proper cleanup
- PXC-2935: Remove incorrect assertion when –thread_handling=pool-of-threads is used
- PXC-2500: Modify ALTER USER processing when executing thread is Galera applier thread to correct assertion
- PXC-3234: Correct documentation link in spec file

- PXC-3204: Modify to set wsrep_protocol_version correctly when wsrep_auto_increment_control is disabled
- PXC-3189: Correct SST processing for super_read_only
- PXC-3184: Modify startup to correct crash when socat not found and SST Fails
- PXC-3169: Modify wsrep_reject_queries to enhance error messaging
- PXC-3165: Allow COM_FIELD_LIST to be executed when WSREP is not ready
- PXC-3145: Modify to end mysqld process when the joiner fails during an SST
- PXC-3043: Update required donor version to PXC 5.7.28 (previously was Known Issue)
- PXC-3036: Document correct method for starting, stopping, bootstrapping
- PXC-3287: Correct link displayed on help client command
- PXC-3031: Modify processing for garbd to prevent issues when multiple requests are started at approximately the same time and request an SST transfers to prevent SST from hanging

Known Issues

- PXC-3039: No useful error messages if an SSL-disabled node tries to join SSL-enabled cluster
- PXC-3092: Abort startup if keyring is specified but cluster traffic encryption is turned off
- PXC-3093: Garbd logs Completed SST Transfer Incorrectly (Timing is not correct)
- PXC-3159: Killing the Donor or Connection lost during SST Process Leaves Joiner Hanging

Percona XtraDB Cluster 8.0.18-9.3

Date April 29, 2020

Installation Installing Percona XtraDB Cluster

Improvements

- PXC-2495: Modified documentation for wsrep_sst_donor to include results when IP address is used
- PXC-3002: Enhanced service_startup_timeout options to allow it to be disabled
- PXC-2331: Modified the SST process to run mysql_upgrade
- PXC-2991: Enhanced Strict Mode Processing to handle Group Replication Plugin
- PXC-2985: Enabled Service for Automated Startup on Reboot with valid grastate.dat
- PXC-2980: Modified Documentation to include AutoStart Up Process after Installation
- PXC-2722: Enabled Support for Percona XtraBackup (PXB) 8.0.8 in Percona XtraDB Cluster (PXC) 8.0
- PXC-2602: Added Ability to Configure xbstream options with wsrep_sst_xtrabackup
- PXC-2455: Implemented the use of Percona XtraBackup (PXB) 8.0.5 in Percona XtraDB Cluster (PXC) 8.0
- PXC-2259: Updated wsrep-files-index.htrml to include new files created by Percona XtraDB Cluster (PXC)
- PXC-2197: Modified SST Documentation to Include Package Dependencies for Percona XtraBackup (PXB)
- PXC-2194: Improvements to the PXC upgrade guide

- PXC-2191: Revised Documentation on innodb_deadlock to Clarify Cluster Level Deadlock Processing
- PXC-3017: Remove these SST encryption methods. encrypt=1, encrypt=2, and encrypt=3
- PXC-2189: Modified Reference Architecture for Percona XtraDB Cluster (PXC) to include ProxySQL

Bugs Fixed

- PXC-2537: Modified mysqladmin password command to prevent node crash
- PXC-2958: Modified User Documentation to include wsrep_certification_rules and cert.optimistic_pa
- PXC-2045: Removed debian.cnf reference from logrotate/logcheck configuration Installed on Xenial/Stretch
- PXC-2292: Modified Processing to determine Type of Key Cert when IST/SST
- PXC-2974: Modified Percona XtraDB Cluster (PXC) Dockerfile to Integrate Galera wsrep recovery Process
- PXC-3145: When the joiner fails during an SST, the mysqld process stays around (doesn't exit)
- PXC-3128: Removed Prior Commit to Allow High Priority High Transaction Processing
- PXC-3076: Modified Galera build to remove python3 components
- PXC-2912: Modified netcat Configuration to Include -N Flag on Donor
- PXC-2476: Modified process to determine and process IST or SST and with keyring_file processing
- PXC-2204: Modified Shutdown using systemd after Bootstrap to provide additional messaging
- PXB-2142: Transition key was written to backup / stream
- PXC-2969: Modified pxc_maint_transition_period Documentation to Include Criteria for Use

Known Issues

- PXC-2978: Certificate Information not Displayed when pxc-encrypt-cluster-traffic=ON
- PXC-3039: No useful error messages if an SSL-disabled node tries to join SSL-enabled cluster
- PXC-3043: Update required donor version to PXC 5.7.28
- PXC-3063: Data at Rest Encryption not Encrypting Record Set Cache
- PXC-3092: Abort startup if keyring is specified but cluster traffic encryption is turned off
- PXC-3093: Garbd logs Completed SST Transfer Incorrectly (Timing is not correct)
- PXC-3159: Killing the Donor or Connection lost during SST Process Leaves Joiner Hanging

CHAPTER

THIRTYSIX

INDEX OF WSREP STATUS VARIABLES

variable `wsrep_apply_oooe`

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

See also:

Galera status variable: wsrep_apply_oooe

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.

See also:

Galera status variable: wsrep_apply_oool

variable `wsrep_apply_window`

Average distance between highest and lowest concurrently applied sequence numbers.

See also:

Galera status variable: wsrep_apply_window

variable `wsrep_causal_reads_`

Shows the number of writesets processed while the variable *`wsrep_causal_reads`* was set to `ON`.

See also:

MySQL wsrep options: wsrep_causal_reads

variable `wsrep_cert_bucket_count`

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.

See also:

Galera status variable: wsrep_cert_deps_distance

variable `wsrep_cert_index_size`

Number of entries in the certification index.

See also:

Galera status variable: wsrep_cert_index_size

variable `wsrep_cert_interval`

Average number of write-sets received while a transaction replicates.

See also:

Galera status variable: wsrep_cert_interval

variable `wsrep_cluster_conf_id`

The number of cluster membership changes that have taken place.

See also:

Galera status variable: wsrep_cluster_conf_id

variable `wsrep_cluster_size`

Current number of nodes in the cluster.

See also:

Galera status variable: wsrep_cluster_size

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.

See also:

Galera status variable: wsrep_cluster_state_uuid

variable `wsrep_cluster_status`

Status of the cluster component. Possible values are:

- `Primary`
- `Non-Primary`
- `Disconnected`

See also:

Galera status variable: wsrep_cluster_status

variable `wsrep_commit_ooo`

This variable shows how often a transaction was committed out of order.

See also:

Galera status variable: wsrep_commit_ooo

variable `wsrep_commit_ool`

This variable currently has no meaning.

See also:

Galera status variable: wsrep_commit_ool

variable `wsrep_commit_window`

Average distance between highest and lowest concurrently committed sequence number.

See also:

Galera status variable: wsrep_commit_window

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.

See also:

Galera status variable: wsrep_connected

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.

See also:

Galera status variable: wsrep_evs_delayed

variable `wsrep_evs_evict_list`

List of UUIDs of the evicted nodes.

See also:

Galera status variable: wsrep_evs_evict_list

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>`.

See also:

Galera status variable: wsrep_evs_repl_latency

variable `wsrep_evs_state`

Internal EVS protocol state.

See also:

Galera status variable: wsrep_evs_state

variable `wsrep_flow_control_interval`

This variable shows the lower and upper limits for Galera flow control. The upper limit is the maximum allowed number of requests in the queue. If the queue reaches the upper limit, new requests are denied. As existing requests get processed, the queue decreases, and once it reaches the lower limit, new requests will be allowed again.

variable `wsrep_flow_control_interval_high`

Shows the upper limit for flow control to trigger.

variable `wsrep_flow_control_interval_low`

Shows the lower limit for flow control to stop.

variable `wsrep_flow_control_paused`

Time since the last status query that was paused due to flow control.

See also:

Galera status variable: wsrep_flow_control_paused

variable `wsrep_flow_control_paused_ns`

Total time spent in a paused state measured in nanoseconds.

See also:

Galera status variable: wsrep_flow_control_paused_ns

variable `wsrep_flow_control_recv`

Number of `FC_PAUSE` events received since the last status query. Unlike most status variables, the counter for this one does not reset every time you run the query.

See also:

Galera status variable: wsrep_flow_control_recv

variable `wsrep_flow_control_sent`

Number of `FC_PAUSE` events sent since the last status query. Unlike most status variables, the counter for this one does not reset every time you run the query.

See also:

Galera status variable: wsrep_flow_control_sent

variable `wsrep_flow_control_status`

This variable shows whether a node has flow control enabled for normal traffic. It does not indicate the status of flow control during SST.

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.

See also:

Galera status variable: wsrep_gcomm_uuid

variable `wsrep_incoming_addresses`

Shows the comma-separated list of incoming node addresses in the cluster.

See also:

Galera status variable: wsrep_incoming_addresses

variable `wsrep_ist_receive_status`

Displays the progress of IST for joiner node. If IST is not running, the value is blank. If IST is running, the value is the percentage of transfer completed.

variable `wsrep_ist_receive_seqno_end`

The sequence number of the last transaction in IST.

variable `wsrep_ist_receive_seqno_current`

The sequence number of the current transaction in IST.

variable `wsrep_ist_receive_seqno_start`

The sequence number of the first transaction in IST.

variable `wsrep_last_applied`

Sequence number of the last applied transaction.

variable `wsrep_last_committed`

Sequence number of the last committed transaction.

variable `wsrep_local_bf_aborts`

Number of local transactions that were aborted by replica transactions while being executed.

See also:

Galera status variable: wsrep_local_bf_aborts

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).

See also:

Galera status variable: wsrep_local_cached_downto

variable `wsrep_local_cert_failures`

Number of writesets that failed the certification test.

See also:

Galera status variable: wsrep_local_cert_failures

variable `wsrep_local_commits`

Number of writesets commited on the node.

See also:

Galera status variable: wsrep_local_commits

variable `wsrep_local_index`

Node's index in the cluster.

See also:

Galera status variable: wsrep_local_index

variable `wsrep_local_recv_queue`

Current length of the receive queue (that is, the number of writesets waiting to be applied).

See also:

Galera status variable: wsrep_local_recv_queue

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.

See also:

Galera status variable: wsrep_local_recv_queue_avg

variable `wsrep_local_replays`

Number of transaction replays due to *asymmetric lock granularity*.

See also:

Galera status variable: wsrep_local_replays

variable `wsrep_local_send_queue`

Current length of the send queue (that is, the number of writesets waiting to be sent).

See also:

Galera status variable: wsrep_local_send_queue

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`.

See also:

Galera status variable: wsrep_local_send_queue_avg

variable `wsrep_local_state`

Internal Galera cluster FSM state number

See also:

Galera status 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

See also:

Galera status variable: wsrep_local_state_comment

variable `wsrep_local_state_uuid`

The *UUID* of the state stored on the node.

See also:

Galera status variable: wsrep_local_state_uuid

variable `wsrep_monitor_status`

The status of the local monitor (local and replicating actions), apply monitor (apply actions of write-set), and commit monitor (commit actions of write sets). In the value of this variable, each monitor (L: Local, A: Apply, C: Commit) is represented as a *last_entered*, and *last_left* pair:

```
wsrep_monitor_status (L/A/C) [ ( 7, 5), (2, 2), ( 2, 2) ]
```

last_entered Shows which transaction or write-set has recently entered the queue

last_left Shows which last transaction or write-set has been executed and left the queue

According to the Galera protocol, transactions can be applied in parallel but must be committed in a given order. This rule implies that there can be multiple transactions in the *apply* state at a given point of time but transactions are *committed* sequentially.

See also:

Galera Documentation: Database replication https://galeracluster.com/library/documentation/tech-desc-introduction.html

variable `wsrep_protocol_version`

Version of the wsrep protocol used.

See also:

Galera status variable: wsrep_protocol_version

variable `wsrep_provider_name`

Name of the wsrep provider (usually `Galera`).

See also:

Galera status variable: wsrep_provider_name

variable `wsrep_provider_vendor`

Name of the wsrep provider vendor (usually `Codership Oy`)

See also:

Galera status variable: wsrep_provider_vendor

variable `wsrep_provider_version`

Current version of the wsrep provider.

See also:

Galera status variable: wsrep_provider_version

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`).

See also:

Galera status variable: wsrep_ready

variable `wsrep_received`

Total number of writesets received from other nodes.

See also:

Galera status variable: wsrep_received

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.

See also:

Galera status variable: wsrep_replicated

variable `wsrep_replicated_bytes`

Total size of replicated writesets. To compute the actual size of bytes sent over network to cluster peers, multiply the value of this variable by the number of cluster peers in the given *`network segment`*.

See also:

Galera status variable: wsrep_replicated_bytes

CHAPTER

THIRTYSEVEN

INDEX OF WSREP SYSTEM VARIABLES

Percona XtraDB Cluster introduces a number of MySQL system variables related to write-set replication.

variable `pxc_encrypt_cluster_traffic`

> **Command Line** `--pxc-encrypt-cluster-traffic`
>
> **Config File** Yes
>
> **Scope** Global
>
> **Dynamic** No
>
> **Default Value** `ON`

Enables automatic configuration of SSL encryption. When disabled, you need to configure SSL manually to encrypt Percona XtraDB Cluster traffic.

Possible values:

- `ON`, `1`, `true`: Enabled (default)
- `OFF`, `0`, `false`: Disabled

For more information, see *SSL Automatic Configuration*.

variable `pxc_maint_mode`

> **Command Line** `--pxc-maint-mode`
>
> **Config File** Yes
>
> **Scope** Global
>
> **Dynamic** Yes
>
> **Default Value** `DISABLED`

Specifies the maintenance mode for taking a node down without adjusting settings in ProxySQL. The following values are available:

- `DISABLED`: This is the default state that tells ProxySQL to route traffic to the node as usual.
- `SHUTDOWN`: This state is set automatically when you initiate node shutdown.
- `MAINTENANCE`: You can manually change to this state if you need to perform maintenance on a node without shutting it down.

For more information, see *Assisted Maintenance Mode*.

variable `pxc_maint_transition_period`

> **Command Line** `--pxc-maint-transition-period`

Config File Yes

Scope Global

Dynamic Yes

Default Value `10` (ten seconds)

Defines the transition period when you change *`pxc_maint_mode`* to `SHUTDOWN` or `MAINTENANCE`. By default, the period is set to 10 seconds, which should be enough for most transactions to finish. You can increase the value to accommodate for longer-running transactions.

For more information, see *Assisted Maintenance Mode*.

variable `pxc_strict_mode`

Command Line `--pxc-strict-mode`

Config File Yes

Scope Global

Dynamic Yes

Default Value `ENFORCING` or `DISABLED`

Controls *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 validation 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_auto_increment_control`

Command Line `--wsrep-auto-increment-control`

Config File Yes

Scope Global

Dynamic Yes

Default Value `ON`

Enables automatic adjustment of auto-increment system variables depending on the size of the cluster:

- `auto_increment_increment` controls the interval between successive `AUTO_INCREMENT` column values
- `auto_increment_offset` determines the starting point for the `AUTO_INCREMENT` column value

This helps prevent auto-increment replication conflicts across the cluster by giving each node its own range of auto-increment values. It is enabled by default.

Automatic adjustment may not be desirable depending on application's use and assumptions of auto-increments. It can be disabled in source-replica clusters.

See also:

MySQL wsrep option: wsrep_auto_increment_control

variable `wsrep_causal_reads`

Command Line `--wsrep-causal-reads`

Config File Yes

Scope Global, Session

Dynamic Yes

Default Value `OFF`

In some cases, the source may apply events faster than a replica, which can cause source and replica to become out of sync for a brief moment. When this variable is set to `ON`, the replica will wait until that event is applied before doing any other queries. Enabling this variable will result in larger latencies.

Note: This variable was deprecated because enabling it is the equivalent of setting *`wsrep_sync_wait`* to `1`.

See also:

MySQL wsrep option: wsrep_causal_reads

variable `wsrep_certification_rules`

Command Line `--wsrep-certification-rules`

Config File Yes

Scope Global

Dynamic Yes

Values STRICT, OPTIMIZED

Default Value STRICT

This variable controls how certification is done in the cluster, in particular this affects how foreign keys are handled.

STRICT Two INSERTs that happen at about the same time on two different nodes in a child table, that insert different (non conflicting rows), but both rows point to the same row in the parent table **may result** in the certification failure.

OPTIMIZED Two INSERTs that happen at about the same time on two different nodes in a child table, that insert different (non conflicting rows), but both rows point to the same row in the parent table **will not result** in the certification failure.

See also:

Galera Cluster Documentation: *MySQL* wsrep options https://galeracluster.com/library/documentation/mysql-wsrep-options.html#wsrep-certification-rules

variable `wsrep_certify_nonPK`

Command Line `--wsrep-certify-nonpk`

Config File Yes

Scope Global

Dynamic Yes

Default Value `ON`

Enables automatic generation of primary keys for rows that don't have them. Write set replication requires primary keys on all tables to allow for parallel applying of transactions. This variable is enabled by default. As a rule, make sure that all tables have primary keys.

See also:

MySQL wsrep option: wsrep_certify_nonPK

variable `wsrep_cluster_address`

Command Line `--wsrep-cluster-address`

Config File Yes

Scope Global

Dynamic Yes

Defines the back-end schema, IP addresses, ports, and options that the node uses when connecting to the cluster. This variable needs to specify at least one other node's address, which 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:

```
<schema>://<address>[?<option1>=<value1>[&<option2>=<value2>]],...
```

The only back-end schema currently supported is `gcomm`. The IP address can contain a port number after a colon. Options are specified after `?` and separated by `&`. You can specify multiple addresses separated by commas.

For example:

```
wsrep_cluster_address="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). It is not recommended to have empty cluster address in production config after the cluster has been bootstrapped initially. If you want to bootstrap a new cluster with a node, you should pass the `--wsrep-new-cluster` option when starting.

See also:

MySQL wsrep option: wsrep_cluster_address

variable `wsrep_cluster_name`

Command Line `--wsrep-cluster-name`

Config File Yes

Scope Global

Dynamic Yes

Default Value `my_wsrep_cluster`

Specifies the name of the cluster and should be identical on all nodes.

Note: It should not exceed 32 characters.

See also:

MySQL wsrep option: wsrep_cluster_name

variable `wsrep_convert_lock_to_trx`

Command Line `--wsrep-convert-lock-to-trx`

Config File Yes

Scope Global

Dynamic Yes

Default Value `OFF`

Defines whether locking sessions should be converted into transactions. By default, this is disabled.

Enabling this variable can help older applications to work in a multi-source setup by converting `LOCK/UNLOCK TABLES` statements into `BEGIN/COMMIT` statements. It is not the same as support for locking sessions, but it does prevent the database from ending up in a logically inconsistent state. Enabling this variable can also result in having huge write-sets.

See also:

MySQL wsrep option: wsrep_convert_lock_to_trx

variable `wsrep_data_home_dir`

Command Line No

Config File Yes

Scope Global

Dynamic No

Default Value `/var/lib/mysql` (or whatever path is specified by *datadir*)

Specifies the path to the directory where the wsrep provider stores its files (such as `grastate.dat`).

See also:

MySQL wsrep option: wsrep_data_home_dir

variable `wsrep_dbug_option`

Command Line `--wsrep-dbug-option`

Config File Yes

Scope Global

Dynamic Yes

Defines `DBUG` options to pass to the wsrep provider.

See also:

MySQL wsrep option: wsrep_dbug_option

variable `wsrep_debug`

Command Line `--wsrep-debug`

Config File Yes

Scope Global

Dynamic Yes

Default Value `NONE`

Enables debug level logging for the database server and wsrep-lib - an integration library for WSREP API with additional convenience for transaction processing. By default, `--wsrep-debug` is disabled.

This variable can be used when trying to diagnose problems or when submitting a bug.

You can set `wsrep_debug` in the following `my.cnf` groups:

- Under `[mysqld]` it enables debug logging for `mysqld` and the SST script
- Under `[sst]` it enables debug logging for the SST script only

This variable may be set to one of the following values:

NONE No debug-level messages.

SERVER

`wsrep-lib` general debug-level messages and detailed debug-level messages from the *server_state* part are printed out. Galera debug-level logs are printed out.

TRANSACTION Same as SERVER + wsrep-lib *transaction* part

STREAMING Same as TRANSACTION + wsrep-lib *streaming* part

CLIENT Same as STREAMING + wsrep-lib *client_service* part

Note: Do not enable debugging in production environments, because it logs authentication info (that is, passwords).

See also:

MySQL wsrep option: wsrep_debug

variable `wsrep_desync`

Command Line No

Config File Yes

Scope Global

Dynamic Yes

Default Value `OFF`

Defines whether the node should participate in Flow Control. By default, this variable is disabled, meaning that if the receive queue becomes too big, the node engages in Flow Control: it works through the receive queue until it reaches a more manageable size. For more information, see *`wsrep_local_recv_queue`* and *`wsrep_flow_control_interval`*.

Enabling this variable will disable Flow Control for the node. It will continue to receive write-sets that it is not able to apply, the receive queue will keep growing, and the node will keep falling behind the cluster indefinitely.

Toggling this back to `OFF` will require an IST or an SST, depending on how long it was desynchronized. This is similar to cluster desynchronization, which occurs during RSU TOI. Because of this, it's not a good idea to enable `wsrep_desync` for a long period of time or for several nodes at once.

Note: You can also desync a node using the `/*! WSREP_DESYNC */` query comment.

See also:

MySQL wsrep option: wsrep_desync

variable `wsrep_dirty_reads`

Command Line `--wsrep-dirty-reads`

Config File Yes

Scope Session, Global

Dynamic Yes

Default Value `OFF`

Defines whether the node accepts read queries when in a non-operational state, that is, when it loses connection to the Primary Component. By default, this variable is disabled and the node rejects all queries, because there is no way to tell if the data is correct.

If you enable this variable, the node will permit read queries (`USE`, `SELECT`, `LOCK TABLE`, and `UNLOCK TABLES`), but any command that modifies or updates the database on a non-operational node will still be rejected (including DDL and DML statements, such as `INSERT`, `DELETE`, and `UPDATE`).

To avoid deadlock errors, set the *`wsrep_sync_wait`* variable to `0` if you enable `wsrep_dirty_reads`.

See also:

MySQL wsrep option: wsrep_dirty_reads

variable `wsrep_drupal_282555_workaround`

Command Line `--wsrep-drupal-282555-workaround`

Config File Yes

Scope Global

Dynamic Yes

Default Value `OFF`

Enables a workaround for MySQL InnoDB bug that affects Drupal (Drupal bug #282555 and MySQL bug #41984). In some cases, duplicate key errors would occur when inserting the `DEFAULT` value into an `AUTO_INCREMENT` column.

See also:

MySQL wsrep option: wsrep_drupal_282555_workaround

variable `wsrep_forced_binlog_format`

Command Line `--wsrep-forced-binlog-format`

Config File Yes

Scope Global

Dynamic Yes

Default Value `NONE`

Defines a binary log format that will always be effective, regardless of the client session `binlog_format` variable value.

Possible values for this variable are:

- `ROW`: Force row-based logging format
- `STATEMENT`: Force statement-based logging format
- `MIXED`: Force mixed logging format
- `NONE`: Do not force the binary log format and use whatever is set by the `binlog_format` variable (default)

See also:

MySQL wsrep option: wsrep_forced_binlog_format

variable `wsrep_ignore_apply_errors`

Command Line `--wsrep-ignore-apply-errors`

Config File Yes

Scope Global

Dynamic Yes

Default Value 0

Defines the rules of wsrep applier behavior on errors. You can change the settings by editing the `my.cnf` file under `[mysqld]` or at runtime.

Note: In Percona XtraDB Cluster version 8.0.19-10, the default value has changed from `7` to `0`. If you have been working with an earlier version of the PXC 8.0 series, you may see different behavior when upgrading to this version or later.

The variable has the following options:

Value	Description
WSREP_IGNORE_ERRORS_NONE	All replication errors are treated as errors and will shutdown the node (default behavior)
WSREP_IGNORE_ERRORS_ON_RECONCILING_DDL	DROP DATABASE, DROP TABLE, DROP INDEX, ALTER TABLE are converted to a warning if they result in ER_DB_DROP_EXISTS, ER_BAD_TABLE_ERROR OR ER_CANT_DROP_FIELD_OR_KEY errors
WSREP_IGNORE_ERRORS_ON_RECONCILING_DML	DELETE events are treated as warnings if they failed because the deleted row was not found (ER_KEY_NOT_FOUND)
WSREP_IGNORE_ERRORS_ON_DDL	All DDL errors will be treated as a warning
WSREP_IGNORE_ERRORS_MAX	Infers WSREP_IGNORE_ERRORS_ON_RECONCILING_DDL, WSREP_IGNORE_ERRORS_ON_RECONCILING_DML and WSREP_IGNORE_ERRORS_ON_DDL

Setting the variable between `0` and `7` has the following behavior:

Setting	Behavior
0	WSREP_IGNORE_ERRORS_NONE
1	WSREP_IGNORE_ERRORS_ON_RECONCILING_DDL
2	WSREP_IGNORE_ERRORS_ON_RECONCILING_DML
3	WSREP_IGNORE_ERRORS_ON_RECONCILING_DDL, WSREP_IGNORE_ERRORS_ON_RECONCILING_DML
4	WSREP_IGNORE_ERRORS_ON_DDL
5	WSREP_IGNORE_ERRORS_ON_DDL, WSREP_IGNORE_ERRORS_ON_RECONCILING_DDL
6	WSREP_IGNORE_ERRORS_ON_DDL, WSREP_IGNORE_ERRORS_ON_RECONCILING_DML
7	WSREP_IGNORE_ERRORS_ON_DDL, WSREP_IGNORE_ERRORS_ON_RECONCILING_DML, WSREP_IGNORE_ERRORS_ON_RECONCILING_DDL

variable `wsrep_min_log_verbosity`

Command Line `--wsrep-min-log-verbosity`

Config File Yes

Scope Global

Dynamic Yes

Default Value 3

This variable defines the *minimum* logging verbosity of wsrep/Galera and acts in conjunction with the `log_error_verbosity` variable. The *`wsrep_min_log_verbosity`* has the same values as `log_error_verbosity`.

The actual log verbosity of wsrep/Galera can be greater than the value of *`wsrep_min_log_verbosity`* if `log_error_verbosity` is greater than *`wsrep_min_log_verbosity`*.

A few examples:

log_error_verbosity	wsrep_min_log_verbosity	*MySQL* Logs Verbosity	wsrep Logs Verbosity
2	3	system error, warning	system error, warning, info
1	3	system error	system error, warning, info
1	2	system error	system error, warning
3	1	system error, warning, info	system error, warning, info

Note the case where `log_error_verbosity=3` and `wsrep_min_log_verbosity=1`. The actual log verbosity of wsrep/Galera is *3* (system error, warning, info) because `log_error_verbosity` is greater.

See also:

***MySQL* Documentation: log_error_verbosity** https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html#sysvar_log_error_verbosity

Galera Cluster Documentation: Database Server Logs https://galeracluster.com/library/documentation/log.html

variable `wsrep_load_data_splitting`

Command Line `--wsrep-load-data-splitting`

Config File Yes

Scope Global

Dynamic Yes

Default Value `ON`

Defines whether the node should split large `LOAD DATA` transactions. This variable is enabled by default, meaning that `LOAD DATA` commands are split into transactions of 10 000 rows or less.

If you disable this variable, then huge data loads may prevent the node from completely rolling the operation back in the event of a conflict, and whatever gets committed stays committed.

Note: It doesn't work as expected with `autocommit=0` when enabled.

See also:

MySQL wsrep option: wsrep_load_data_splitting

variable `wsrep_log_conflicts`

Command Line `--wsrep-log-conflicts`

Config File Yes

Scope Global

Dynamic No

Default Value `OFF`

Defines whether the node should log additional information about conflicts. By default, this variable is disabled and Percona XtraDB Cluster uses standard logging features in MySQL.

If you enable this variable, it will also log table and schema where the conflict occurred, as well as the actual values for keys that produced the conflict.

See also:

MySQL wsrep option: wsrep_log_conflicts

variable `wsrep_max_ws_rows`

Command Line `--wsrep-max-ws-rows`

Config File Yes

Scope Global

Dynamic Yes

Default Value `0` (no limit)

Defines the maximum number of rows each write-set can contain.

By default, there is no limit for the maximum number of rows in a write-set. The maximum allowed value is `1048576`.

See also:

MySQL wsrep option: wsrep_max_ws_rows

variable `wsrep_max_ws_size`

Command Line `--wsrep_max_ws_size`

Config File Yes

Scope Global

Dynamic Yes

Default Value `2147483647` (2 GB)

Defines the maximum write-set size (in bytes). Anything bigger than the specified value will be rejected.

You can set it to any value between `1024` and the default `2147483647`.

See also:

MySQL wsrep option: wsrep_max_ws_size

variable `wsrep_node_address`

Command Line `--wsrep-node-address`

Config File Yes

Scope Global

Dynamic No

Default Value IP of the first network interface (`eth0`) and default port (`4567`)

Specifies the network address of the node. By default, this variable is set to the IP address of the first network interface (usually `eth0` or `enp2s0`) and the default port (`4567`).

While default value should be correct in most cases, there are situations when you need to specify it manually. For example:

- Servers with multiple network interfaces
- Servers that run multiple nodes
- Network Address Translation (NAT)
- Clusters with nodes in more than one region
- Container deployments, such as Docker
- Cloud deployments, such as Amazon EC2 (use the global DNS name instead of the local IP address)

The value should be specified in the following format:

```
<ip_address>[:port]
```

Note: The value of this variable is also used as the default value for the *`wsrep_sst_receive_address`* variable and the *`ist.recv_addr`* option.

See also:

MySQL wsrep option: wsrep_node_address

variable `wsrep_node_incoming_address`

Command Line `--wsrep-node-incoming-address`

Config File Yes

Scope Global

Dynamic No

Default Value `AUTO`

Specifies the network address from which the node expects client connections. By default, it uses the IP address from *`wsrep_node_address`* and port number 3306.

This information is used for the *`wsrep_incoming_addresses`* variable which shows all active cluster nodes.

See also:

MySQL wsrep option: wsrep_node_incoming_address

variable `wsrep_node_name`

Command Line `--wsrep-node-name`

Config File Yes

Scope Global

Dynamic Yes

Default Value The node's host name

Defines a unique name for the node. Defaults to the host name.

In many situations, you may use the value of this variable as a means to identify the given node in the cluster as the alternative to using the node address (the value of the *wsrep_node_address*).

Note: The variable *wsrep_sst_donor* is an example where you may only use the value of *wsrep_node_name* and the node address is not permitted.

---

variable wsrep_notify_cmd

Command Line `--wsrep-notify-cmd`

Config File Yes

Scope Global

Dynamic Yes

Specifies the notification command that the node should execute whenever cluster membership or local node status changes. This can be used for alerting or to reconfigure load balancers.

Note: The node will block and wait until the command or script completes and returns before it can proceed. If the script performs any potentially blocking or long-running operations, such as network communication, you should consider initiating such operations in the background and have the script return immediately.

See also:

MySQL wsrep option: wsrep_notify_cmd

variable wsrep_on

Command Line No

Config File No

Scope Session

Dynamic Yes

Default Value `ON`

Defines if current session transaction changes for a node are replicated to the cluster.

If set to `OFF` for a session, no transaction changes are replicated in that session. The setting does not cause the node to leave the cluster, and the node communicates with other nodes.

See also:

MySQL wsrep option: wsrep_on

variable wsrep_OSU_method

Command Line `--wsrep-OSU-method`

Config File Yes

Scope Global and Session

Dynamic Yes

Default Value `TOI`

Defines the method for Online Schema Upgrade that the node uses to replicate DDL statements. The following methods are available:

- `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 node. This guarantees data consistency.

  In the case of DDL statements, the cluster will have parts of the 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 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, the 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.

  Important: Under the `TOI` method, when DDL operations are performed, MDL (Metadata Locking) is ignored. If MDL is important, use the `RSU` method.

- `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, the 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.

See also:

MySQL wsrep option: wsrep_OSU_method

variable `wsrep_provider`

Command Line `--wsrep-provider`

Config File Yes

Scope Global

Dynamic Yes

Specifies the path to the Galera library. This is usually `/usr/lib64/libgalera_smm.so` on *CentOS/RHEL* and `/usr/lib/libgalera_smm.so` on *Debian/Ubuntu*.

If you do not specify a path or the value is not valid, the node will behave as standalone instance of MySQL.

See also:

MySQL wsrep option: wsrep_provider

variable `wsrep_provider_options`

Command Line `--wsrep-provider-options`

Config File Yes

Scope Global

Dynamic No

Specifies optional settings for the replication provider documented in *Index of wsrep_provider options*. These options affect how various situations are handled during replication.

See also:

MySQL wsrep option: wsrep_provider_options

variable `wsrep_recover`

Command Line `--wsrep-recover`

Config File Yes

Scope Global

Dynamic No

Default Value `OFF`

Location mysqld_safe

Recovers database state after crash by parsing GTID from the log. If the GTID is found, it will be assigned as the initial position for server.

variable `wsrep_reject_queries`

Command Line No

Config File Yes

Scope Global

Dynamic Yes

Default Value `NONE`

Defines whether the node should reject queries from clients. Rejecting queries can be useful during upgrades, when you want to keep the node up and apply write-sets without accepting queries.

When a query is rejected, the following error appears:

```
ERROR 1047 (08S01): WSREP has not yet prepared node for application use
```

The following values are available:

- `NONE`: Accept all queries from clients (default)
- `ALL`: Reject all new queries from clients, but maintain existing client connections
- `ALL_KILL`: Reject all new queries from clients and kill existing client connections

---

Note: This variable doesn't affect Galera replication in any way, only the applications that connect to the database are affected. If you want to desync a node, use *`wsrep_desync`*.

When a query is rejected, an error is returned.

See also:

MySQL wsrep option: wsrep_reject_queries

variable `wsrep_replicate_myisam`

Command Line `--wsrep-replicate-myisam`

Config File Yes

Scope Session, Global

Dynamic No

Default Value `OFF`

Defines whether DML statements for MyISAM tables should be replicated. It is disabled by default, because MyISAM replication is still experimental.

On the global level, *`wsrep_replicate_myisam`* can be set only during startup. 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 the origin node. Mixing 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 the same table on different nodes
- `CREATE TABLE AS SELECT` (CTAS) statements use TOI replication. MyISAM tables are created and loaded even if *`wsrep_replicate_myisam`* is set to **ON**.

---

variable `wsrep_restart_slave`

Command Line `--wsrep-restart-slave`

Config File Yes

Scope Global

Dynamic Yes

Default Value `OFF`

Defines whether replication replica should be restarted when the node joins back to the cluster. Enabling this can be useful because asynchronous replication replica thread is stopped when the node tries to apply the next replication event while the node is in non-primary state.

See also:

MySQL wsrep option: wsrep_restart_slave

variable `wsrep_retry_autocommit`

Command Line `--wsrep-retry-autocommit`

Config File Yes

Scope Global

Dynamic No

Default Value `1`

Specifies the number of times autocommit transactions will be retried 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 returning an error to the client, 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`, autocommit transactions won't be retried.

See also:

MySQL wsrep option: wsrep_retry_autocommit

variable `wsrep_RSU_commit_timeout`

Command Line `--wsrep-RSU-commit-timeout`

Config File Yes

Scope Global

Dynamic Yes

Default Value `5000`

Range From `5000` (5 millisecons) to `31536000000000` (365 days)

Specifies the timeout in microseconds to allow active connection to complete COMMIT action before starting RSU.

While running RSU it is expected that user has isolated the node and there is no active traffic executing on the node. RSU has a check to ensure this, and waits for any active connection in `COMMIT` state before starting RSU.

By default this check has timeout of 5 millisecons, but in some cases COMMIT is taking longer. This variable sets the timeout, and has allowed values from the range of (5 millisecons, 365 days). The value is to be set in microseconds. Unit of variable is in micro-secs so set accordingly.

Note: RSU operation will not auto-stop node from receiving active traffic. So there could be a continuous flow of active traffic while RSU continues to wait, and that can result in RSU starvation. User is expected to block active RSU traffic while performing operation.

---

variable `wsrep_slave_FK_checks`

Command Line `--wsrep-slave-FK-checks`

Config File Yes

Scope Global

Dynamic Yes

Default Value `ON`

Defines whether foreign key checking is done for applier threads. This is enabled by default.

See also:

MySQL wsrep option: wsrep_slave_FK_checks

variable `wsrep_slave_threads`

Command Line `--wsrep-slave-threads`

Config File Yes

Scope Global

Dynamic Yes

Default Value `1`

Specifies the number of threads that can apply replication transactions in parallel. Galera supports true parallel replication that applies transactions in parallel only when it is safe to do so. This variable is dynamic. You can increase/decrease it at any time.

Note: When you decrease the number of threads, it won't kill the threads immediately, but stop them after they are done applying current transaction (the effect with an 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 for flow control: when the node is in `JOINED` state, increasing the number of replica threads can speed up the catchup to `SYNCED`.

You can also estimate the optimal value for this from *`wsrep_cert_deps_distance`* as suggested on this page.

For more configuration tips, see this document.

See also:

MySQL wsrep option: wsrep_slave_threads

variable `wsrep_slave_UK_checks`

Command Line `--wsrep-slave-UK-checks`

Config File Yes

Scope Global

Dynamic Yes

Default Value `OFF`

Defines whether unique key checking is done for applier threads. This is disabled by default.

See also:

MySQL wsrep option: wsrep_slave_UK_checks

variable `wsrep_sst_donor`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Specifies a list of nodes (using their *`wsrep_node_name`* values) that the current node should prefer as donors for *SST* and *IST*.

Warning: Using IP addresses of nodes instead of node names (the value of *wsrep_node_name*) as values of *wsrep_sst_donor* results in an error.

Error message

[ERROR] WSREP: State transfer request failed unrecoverably: 113 (No route to host). Most likely it is due to inability to communicate with the cluster primary component. Restart required.

If the value is empty, the first node in SYNCED state in the index becomes the donor and will not be able to serve requests during the state transfer.

To consider other nodes if the listed nodes are not available, add a comma at the end of the list, for example:

```
wsrep_sst_donor=node1,node2,
```

If you remove the trailing comma from the previous example, then the joining node will consider *only* `node1` and `node2`.

Note: By default, the joiner node does not wait for more than 100 seconds to receive the first packet from a donor. This is implemented via the *sst-initial-timeout* option. If you set the list of preferred donors without the trailing comma or believe that all nodes in the cluster can often be unavailable for SST (this is common for small clusters), then you may want to increase the initial timeout (or disable it completely if you don't mind the joiner node waiting for the state transfer indefinitely).

See also:

MySQL wsrep option: wsrep_sst_donor

variable `wsrep_sst_method`

Command Line `--wsrep-sst-method`

Config File Yes

Scope Global

Dynamic Yes

Default Value xtrabackup-v2

Defines the method or script for *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 credentials (`<user>:<password>`) on the donor node. Privileges and perimssions for running *Percona XtraBackup* can be found in Percona XtraBackup documentation.

  For more information, see *Percona XtraBackup SST Configuration*.

- `<custom_script_name>`: Galera supports Scriptable State Snapshot Transfer. This enables users to create their own custom scripts for performing 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. This can be used when initially starting the cluster and manually restoring the same data to all nodes. It shouldn't be used permanently because it could lead to data inconsistency across the nodes.

Note: `xtrabackup-v2` provides support for clusters with GTIDs and async replicas.

See also:

MySQL wsrep option: wsrep_sst_method

variable `wsrep_sst_receive_address`

Command Line `--wsrep-sst-receive-address`

Config File Yes

Scope Global

Dynamic Yes

Default Value `AUTO`

Specifies the network address where donor node should send state transfers. By default, this variable is set to `AUTO`, meaning that the IP address from *`wsrep_node_address`* is used.

See also:

MySQL wsrep option: wsrep_sst_receive_address

variable `wsrep_start_position`

Command Line `--wsrep-start-position`

Config File Yes

Scope Global

Dynamic Yes

Default Value `00000000-0000-0000-0000-000000000000:-1`

Specifies the node's start position as `UUID:seqno`. By setting all the nodes to have the same value for this variable, the cluster can be set up without the state transfer.

See also:

MySQL wsrep option: wsrep_start_position

variable `wsrep_sync_wait`

Command Line `--wsrep-sync-wait`

Config File Yes

Scope Session

Dynamic Yes

Default Value `0`

Controls cluster-wide causality checks on certain statements. Checks ensure that the statement is executed on a node that is fully synced with the cluster.

Note: Causality checks of any type can result in increased latency.

The type of statements to undergo checks is determined by bitmask:

- `0`: Do not run causality checks for any statements. This is the default.

- 1: Perform checks for READ statements (including SELECT, SHOW, and BEGIN or START TRANSACTION).
- 2: Perform checks for UPDATE and DELETE statements.
- 3: Perform checks for READ, UPDATE, and DELETE statements.
- 4: Perform checks for INSERT and REPLACE statements.
- 5: Perform checks for READ, INSERT, and REPLACE statements.
- 6: Perform checks for UPDATE, DELETE, INSERT, and REPLACE statements.
- 7: Perform checks for READ, UPDATE, DELETE, INSERT, and REPLACE statements.

Note: Setting *wsrep_sync_wait* to 1 is the equivalent of setting the deprecated *wsrep_causal_reads* to ON.

See also:

MySQL wsrep option: wsrep_sync_wait

CHAPTER

THIRTYEIGHT

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";
```

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 `cert.optimistic_pa`

Enabled Allows the full range of parallelization as determined by the certification algorithm.

Disabled Limits the parallel applying window so that it does not exceed the parallel applying window seen on the source. In this case, the action starts applying no sooner than all actions on the source are committed.

cli Yes

conf Yes

scope Global

dyn Yes

default NO

See also:

Galera Cluster Documentation:

- Parameter: cert.optimistic_pa
- Setting parallel slave threads

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 10

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 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).

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.freeze_purge_at_seqno`

Command Line Yes

Config File Yes

Scope Local, Global

Dynamic Yes

Default Value 0

This variable controls the purging of the gcache and enables retaining more data in it. This variable makes it possible to use *IST (Incremental State Transfer)* when the node rejoins instead of *SST (State Snapshot Transfer)*.

Set this variable on an existing node of the cluster (that will continue to be part of the cluster and can act as a potential *donor node*). This node continues to retain the write-sets and allows restarting the node to rejoin by using *IST*.

See also:

Percona Database Performance Blog:

- All You Need to Know About GCache (Galera-Cache)
- Want IST Not SST for Node Rejoins? We Have a Solution!

The *`gcache.freeze_purge_at_seqno`* variable takes three values:

-1 (default) No freezing of gcache, the purge operates as normal.

A valid seqno in gcache The freeze purge of write-sets may not be smaller than the selected seqno. The best way to select an optimal value is to use the value of the variable *`wsrep_last_applied`* from the node that you plan to shut down.

now The freeze purge of write-sets is no less than the smallest seqno currently in gcache. Using this value results in freezing the gcache-purge instantly. Use this value if selecting a valid seqno in gcache is difficult.

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 `gcache.keep_pages_count` is set to a non-zero value, excess overflow pages will be deleted (starting from the oldest to the newest).

Whenever either the `gcache.keep_pages_count` or the *`gcache.keep_pages_size`* variable is 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 limit the total size of overflow pages rather than the count of all overflow pages. Whenever `gcache.keep_pages_size` is set to a non-zero value, excess overflow pages will be deleted (starting from the oldest to the newest) until the total size is below the specified value.

Whenever either the *`gcache.keep_pages_count`* or the `gcache.keep_pages_size` variable is updated at runtime to a non-zero value, cleanup is called on excess overflow pages to delete them.

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

Size of the page files in page storage. The limit on overall page storage is the size of the disk. Pages are prefixed by gcache.page.

See also:

Galera Cluster Documentation: gcache.page_size https://galeracluster.com/library/documentation/galera-parameters.html#gcache-page-size

***Percona* Database Performance Blog: All You Need to Know About GCache (Galera-Cache)** https://www.percona.com/blog/2016/11/16/all-you-need-to-know-about-gcache-galera-cache/

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 `gcomm.thread_prio`

Using this option, you can raise the priority of the gcomm thread to a higher level than it normally uses.

The format for this option is: <policy>:<priority>. The priority value is an integer. The policy value supports the following options:

other Default time-sharing scheduling in Linux. The threads can run until blocked by an I/O request or preempted by higher priorities or superior scheduling designations.

fifo First-in First-out (FIFO) scheduling. These threads always immediately preempt any currently running other, batch or idle threads. They can run until they are either blocked by an I/O request or preempted by a FIFO thread of a higher priority.

rr Round-robin scheduling. These threads always preempt any currently running other, batch or idle threads. The scheduler allows these threads to run for a fixed period of a time. If the thread is still running when this time period is exceeded, they are stopped and moved to the end of the list, allowing another round-robin thread of the same priority to run in their place. They can otherwise continue to run until they are blocked by an I/O request or are preempted by threads of a higher priority.

See also:

Full definition in Galera Cluster documentation: gcomm.thread_prio

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 Yes

Default Value 1

This variable is used for replication flow control. Replication is resumed when the replica queue drops below *gcs.fc_factor* * *gcs.fc_limit*.

variable `gcs.fc_limit`

Command Line Yes

Config File Yes

Scope Global

Dynamic Yes

Default Value 100

This variable is used for replication flow control. Replication is paused when the replica queue exceeds this limit. In the default operation mode, flow control limit is dynamically recalculated based on the amount of nodes in the cluster, but this recalculation can be turned off with use of the *gcs.fc_master_slave* variable to make manual setting of the *gcs.fc_limit* having an effect (e.g. for configurations when writing is done to a single node in Percona XtraDB Cluster).

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 source node in the cluster. It affects whether flow control limit is recalculated dynamically (when `NO`) or not (when `YES`).

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 source-replica setups, because replicas won't automatically reconnect to source 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-source setup, but should simplify things in source-replica 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 10

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 `CRC32-C` option is optimized and may be hardware accelerated on Intel CPUs. The following values are available:

- `0` - disable checksum
- `1` - `CRC32`
- `2` - `CRC32-C`

The following is an example of the variable use:

```
wsrep_provider_options="socket.checksum=2"
```

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_ca`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

This variable is used to specify the path to the Certificate Authority (CA) certificate file.

variable `socket.ssl_cert`

Command Line Yes

Config File Yes

Scope Global

Dynamic No

This variable is used to specify the path to the server's certificate file (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 to the server's private key file (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.

CHAPTER

THIRTYNINE

INDEX OF FILES CREATED BY PXC

- **`GRA_*.log`** These files contain binlog events in ROW format representing the failed transaction. That means that the replica 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 8.0-log created 160809 16:04:05 at startup
# 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    error_code=0
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, @@session.autocommit=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_server=8/*!*/;
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 database: 'test'. Query: 'drop table test', Error_code: 1051
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.

- `gcache.page`

  See *gcache.page_size*

  See also:

  ***Percona* Database Performance Blog: All You Need to Know About GCache (Galera-Cache)**
  https://www.percona.com/blog/2016/11/16/all-you-need-to-know-about-gcache-galera-cache/

- **`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
```

CHAPTER

FORTY

FREQUENTLY ASKED QUESTIONS

- *How do I report bugs?*
- *How do I solve locking issues like auto-increment?*
- *What if a node crashes and InnoDB recovery rolls back some transactions?*
- *How can I check the Galera node health?*
- *How does Percona XtraDB Cluster handle big transactions?*
- *Is it possible to have different table structures on the nodes?*
- *What if a node fails or there is a network issue between nodes?*
- *How would the quorum mechanism handle split brain?*
- *Why a node stops accepting commands if the other one fails in a 2-node setup?*
- *Is it possible to set up a cluster without state transfer?*
- *What TCP ports are used by Percona XtraDB Cluster?*
- *Is there "async" mode or only "sync" commits are supported?*
- *Does it work with regular MySQL replication?*
- *Why the init script (/etc/init.d/mysql) does not start?*
- *What does "nc: invalid option – 'd"' in the sst.err log file mean?*

How do I report bugs?

All bugs can be reported on JIRA. Please submit `error.log` files from **all** the nodes.

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.

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).

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:

```
mysql> CREATE USER 'clustercheck'@'localhost' IDENTIFIED BY PASSWORD
'*2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT PROCESS ON *.* TO 'clustercheck'@'localhost';
```

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`

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.

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.

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.

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"
```

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.

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.

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; "
```

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.

Does it work with regular MySQL replication?

Yes. On the node you are going to use as source, you should enable `log-bin` and `log-slave-update` options.

Why the init script (/etc/init.d/mysql) does not start?

Try to disable SELinux with the following command:

```
echo 0 > /selinux/enforce
```

What does “nc: invalid option – ‘d’” in the sst.err log file mean?

This error is specific to Debian and Ubuntu. Percona XtraDB Cluster uses `netcat-openbsd` package. This dependency has been fixed. Future releases of Percona XtraDB Cluster will be compatible with any `netcat` (see bug PXC-941).

CHAPTER

FORTYONE

GLOSSARY

ACID Set of properties that guarantee database transactions are processed reliably. Stands for *Atomicity*, *Consistency*, *Isolation*, *Durability*.

Atomicity Atomicity means that database operations are applied following a "all or nothing" rule. A transaction is either fully applied or not at all.

Consistency Consistency means that each transaction that modifies the database takes it from one consistent state to another.

Durability Once a transaction is committed, it will remain so.

Foreign Key A referential constraint between two tables. Example: A purchase order in the purchase_orders table must have been made by a customer that exists in the customers table.

Isolation The Isolation requirement means that no transaction can interfere with another.

InnoDB A *Storage Engine* for MySQL and derivatives (*Percona Server*, *MariaDB*) originally written by Innobase Oy, since acquired by Oracle. It provides *ACID* compliant storage engine with *foreign key* support. InnoDB is the default storage engine on all platforms.

Jenkins Jenkins is a continuous integration system that we use to help ensure the continued quality of the software we produce. It helps us achieve the aims of:

- no failed tests in trunk on any platform,
- aid developers in ensuring merge requests build and test on all platforms,
- no known performance regressions (without a damn good explanation).

LSN Log Serial Number. A term used in relation to the *InnoDB* or *XtraDB* storage engines.

MariaDB A fork of *MySQL* that is maintained primarily by Monty Program AB. It aims to add features, fix bugs while maintaining 100% backwards compatibility with MySQL.

MyISAM A *MySQL Storage Engine* that was the default until MySQL 5.5.

MySQL An open source database that has spawned several distributions and forks. MySQL AB was the primary maintainer and distributor until bought by Sun Microsystems, which was then acquired by Oracle. As Oracle owns the MySQL trademark, the term MySQL is often used for the Oracle distribution of MySQL as distinct from the drop-in replacements such as *MariaDB* and *Percona Server*.

NUMA Non-Uniform Memory Access (NUMA) is a computer memory design used in multiprocessing, where the memory access time depends on the memory location relative to a processor. Under NUMA, a processor can access its own local memory faster than non-local memory, that is, memory local to another processor or memory shared between processors. The whole system may still operate as one unit, and all memory is basically accessible from everywhere, but at a potentially higher latency and lower performance.

Percona Server for MySQL Percona's branch of *MySQL* with performance and management improvements.

Percona Server See *Percona Server for MySQL*

Storage Engine A *Storage Engine* is a piece of software that implements the details of data storage and retrieval for a database system. This term is primarily used within the *MySQL* ecosystem due to it being the first widely used relational database to have an abstraction layer around storage. It is analogous to a Virtual File System layer in an Operating System. A VFS layer allows an operating system to read and write multiple file systems (e.g. FAT, NTFS, XFS, ext3) and a Storage Engine layer allows a database server to access tables stored in different engines (e.g. *MyISAM*, InnoDB).

XtraDB Percona's improved version of *InnoDB* providing performance, features and reliability above what is shipped by Oracle in InnoDB.

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 the 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 an existing node.

Percona XtraDB Cluster: uses the **`xtrabackup`** program for this purpose. **`xtrabackup`** 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 replica information (same as the regular *Percona XtraBackup* backup).

The SST method is configured with the *`wsrep_sst_method`* variable.

In Percona XtraDB Cluster 8.0, the **`mysql_upgrade`** command is now run automatically as part of *SST*. You do not have to run it manually when upgrading your system from an older version.

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).

mysql.pxc.internal.session This user is used by the SST process to run the SQL commands needed for *SST*, such as creating the *mysql.pxc.sst.user* and assigning it the role *mysql.pxc.sst.role*).

Difference from previous major version 5.7

The `wsrep_sst_auth` variable is no longer needed and has been removed.

---

mysql.pxc.sst.role This role has all the privileges needed to run xtrabackup to create a backup on the donor node.

mysql.pxc.sst.user This user (set up on the donor node) is assigned the *mysql.pxc.sst.role* and runs the **`xtrabackup`** to make backups. The password for this is randomly generated for each SST. The password is generated automatically for each *SST*.

- genindex
- search

INDEX

Symbols

.frm, **238**

A

ACID, **235**
Atomicity, **235**

B

backup_threads (option), 70
base_dir (variable), 209
base_host (variable), 209
base_port (variable), 209

C

cert.log_conflicts (variable), 210
cert.optimistic_pa (variable), 210
cluster replication, **237**
compressor (option), 68
Consistency, **235**
cpat (option), 68

D

datadir, **237**
debug (variable), 210
decompressor (option), 68
donor node, **237**
Durability, **235**

E

encrypt (option), 66
encrypt_threads (option), 69
evs.auto_evict (variable), 210
evs.causal_keepalive_period (variable), 211
evs.debug_log_mask (variable), 211
evs.delay_margin (variable), 211
evs.delayed_keep_period (variable), 211
evs.evict (variable), 211
evs.inactive_check_period (variable), 212
evs.inactive_timeout (variable), 212
evs.info_log_mask (variable), 212
evs.install_timeout (variable), 212
evs.join_retrans_period (variable), 212
evs.keepalive_period (variable), 213
evs.max_install_timeouts (variable), 213
evs.send_window (variable), 213
evs.stats_report_period (variable), 213
evs.suspect_timeout (variable), 213
evs.use_aggregate (variable), 214
evs.user_send_window (variable), 214
evs.version (variable), 214
evs.view_forget_timeout (variable), 214

F

Foreign Key, **235**

G

gcache.dir (variable), 215
gcache.freeze_purge_at_seqno (variable), 215
gcache.keep_pages_count (variable), 215
gcache.keep_pages_size (variable), 216
gcache.mem_size (variable), 216
gcache.name (variable), 216
gcache.page_size (variable), 216
gcache.size (variable), 217
gcomm.thread_prio (variable), 217
gcs.fc_debug (variable), 217
gcs.fc_factor (variable), 217
gcs.fc_limit (variable), 218
gcs.fc_master_slave (variable), 218
gcs.max_packet_size (variable), 218
gcs.max_throttle (variable), 218
gcs.recv_q_hard_limit (variable), 219
gcs.recv_q_soft_limit (variable), 219
gcs.sync_donor (variable), 219
gmcast.listen_addr (variable), 219
gmcast.mcast_addr (variable), 219
gmcast.mcast_ttl (variable), 220
gmcast.peer_timeout (variable), 220
gmcast.segment (variable), 220
gmcast.time_wait (variable), 220
gmcast.version (variable), 220
GTID, **236**

H

HAProxy, **236**

I

ibdata, **237**
inno-apply-opts (option), 68
inno-backup-opts (option), 68
inno-move-opts (option), 68
InnoDB, **236**
InnoDB A, **235**
Isolation, **235**
IST, **236**
ist.recv_addr (variable), 221

J

Jenkins, **235**
joiner node, **237**

L

LSN, **235**, **236**

M

MariaDB, **235**
my.cnf, **237**
MyISAM, **235**, **236**
MySQL, **235**
mysql.pxc.internal.session, **238**
mysql.pxc.sst.role, **238**
mysql.pxc.sst.user, **238**

N

ncsockopt (option), 67
node, **237**
NUMA, **235**

P

pc.announce_timeout (variable), 221
pc.checksum (variable), 221
pc.ignore_quorum (variable), 221
pc.ignore_sb (variable), 221
pc.linger (variable), 222
pc.npvo (variable), 222
pc.recovery (variable), 222
pc.version (variable), 222
pc.wait_prim (variable), 222
pc.wait_prim_timeout (variable), 223
pc.weight (variable), 223
Percona Server, **236**
Percona Server for MySQL, **235**
Percona XtraDB Cluster, **237**
primary cluster, **237**
progress (option), 67
protonet.backend (variable), 223
protonet.version (variable), 223
pxc_encrypt_cluster_traffic (variable), 189
pxc_maint_mode (variable), 189
pxc_maint_transition_period (variable), 189
pxc_strict_mode (variable), 190

Q

quorum, **237**

R

rebuild (option), 67
repl.causal_read_timeout (variable), 223
repl.commit_order (variable), 224
repl.key_format (variable), 224
repl.max_ws_size (variable), 224
repl.proto_max (variable), 225
rlimit (option), 67

S

socket.checksum (variable), 225
socket.ssl (variable), 225
socket.ssl_ca (variable), 225
socket.ssl_cert (variable), 225
socket.ssl_cipher (variable), 226
socket.ssl_compression (variable), 226
socket.ssl_key (variable), 226
sockopt (option), 66
split brain, **237**
ssl-ca (option), 66
ssl-cert (option), 66
ssl-key (option), 66
SST, **236**
sst-initial-timeout (option), 69
Storage Engine, **236**
streamfmt (option), 65

T

time (option), 67
tmpdir (option), 69
transferfmt (option), 65

U

use_extra (option), 67
UUID, **236**

W

wsrep_apply_oooe (variable), 181
wsrep_apply_oool (variable), 181
wsrep_apply_window (variable), 181
wsrep_auto_increment_control (variable), 190
wsrep_causal_reads (variable), 191
wsrep_causal_reads_ (variable), 181
wsrep_cert_bucket_count (variable), 181

wsrep_cert_deps_distance (variable), 181
wsrep_cert_index_size (variable), 181
wsrep_cert_interval (variable), 181
wsrep_certification_rules (variable), 191
wsrep_certify_nonPK (variable), 192
wsrep_cluster_address (variable), 192
wsrep_cluster_conf_id (variable), 182
wsrep_cluster_name (variable), 192
wsrep_cluster_size (variable), 182
wsrep_cluster_state_uuid (variable), 182
wsrep_cluster_status (variable), 182
wsrep_commit_oooe (variable), 182
wsrep_commit_oool (variable), 182
wsrep_commit_window (variable), 182
wsrep_connected (variable), 182
wsrep_convert_lock_to_trx (variable), 193
wsrep_data_home_dir (variable), 193
wsrep_dbug_option (variable), 193
wsrep_debug (variable), 193
wsrep_desync (variable), 194
wsrep_dirty_reads (variable), 195
wsrep_drupal_282555_workaround (variable), 195
wsrep_evs_delayed (variable), 183
wsrep_evs_evict_list (variable), 183
wsrep_evs_repl_latency (variable), 183
wsrep_evs_state (variable), 183
wsrep_flow_control_interval (variable), 183
wsrep_flow_control_interval_high (variable), 183
wsrep_flow_control_interval_low (variable), 183
wsrep_flow_control_paused (variable), 183
wsrep_flow_control_paused_ns (variable), 183
wsrep_flow_control_recv (variable), 184
wsrep_flow_control_sent (variable), 184
wsrep_flow_control_status (variable), 184
wsrep_forced_binlog_format (variable), 195
wsrep_gcache_pool_size (variable), 184
wsrep_gcomm_uuid (variable), 184
wsrep_ignore_apply_errors (variable), 196
wsrep_incoming_addresses (variable), 184
wsrep_ist_receive_seqno_current (variable), 184
wsrep_ist_receive_seqno_end (variable), 184
wsrep_ist_receive_seqno_start (variable), 184
wsrep_ist_receive_status (variable), 184
wsrep_last_applied (variable), 184
wsrep_last_committed (variable), 184
wsrep_load_data_splitting (variable), 197
wsrep_local_bf_aborts (variable), 185
wsrep_local_cached_downto (variable), 185
wsrep_local_cert_failures (variable), 185
wsrep_local_commits (variable), 185
wsrep_local_index (variable), 185
wsrep_local_recv_queue (variable), 185
wsrep_local_recv_queue_avg (variable), 185
wsrep_local_replays (variable), 185
wsrep_local_send_queue (variable), 185
wsrep_local_send_queue_avg (variable), 186
wsrep_local_state (variable), 186
wsrep_local_state_comment (variable), 186
wsrep_local_state_uuid (variable), 186
wsrep_log_conflicts (variable), 198
wsrep_max_ws_rows (variable), 198
wsrep_max_ws_size (variable), 198
wsrep_min_log_verbosity (variable), 197
wsrep_monitor_status (variable), 186
wsrep_node_address (variable), 199
wsrep_node_incoming_address (variable), 199
wsrep_node_name (variable), 199
wsrep_notify_cmd (variable), 200
wsrep_on (variable), 200
wsrep_OSU_method (variable), 200
wsrep_protocol_version (variable), 186
wsrep_provider (variable), 201
wsrep_provider_name (variable), 187
wsrep_provider_options (variable), 201
wsrep_provider_vendor (variable), 187
wsrep_provider_version (variable), 187
wsrep_ready (variable), 187
wsrep_received (variable), 187
wsrep_received_bytes (variable), 187
wsrep_recover (variable), 202
wsrep_reject_queries (variable), 202
wsrep_repl_data_bytes (variable), 187
wsrep_repl_keys (variable), 187
wsrep_repl_keys_bytes (variable), 187
wsrep_repl_other_bytes (variable), 187
wsrep_replicate_myisam (variable), 203
wsrep_replicated (variable), 187
wsrep_replicated_bytes (variable), 188
wsrep_restart_slave (variable), 203
wsrep_retry_autocommit (variable), 203
wsrep_RSU_commit_timeout (variable), 204
wsrep_slave_FK_checks (variable), 204
wsrep_slave_threads (variable), 204
wsrep_slave_UK_checks (variable), 205
wsrep_sst_donor (variable), 205
wsrep_sst_method (variable), 206
wsrep_sst_receive_address (variable), 207
wsrep_start_position (variable), 207
wsrep_sync_wait (variable), 207

X

XtraBackup, **237**
XtraDB, **236**, **237**
XtraDB Cluster, **237**