Galera Cluster Documentation

Codership Oy

May 28, 2025

CONTENTS

1 Overview of Galera Cluster **4**

2 Technical Description **7**
2.1 Database Replication . . . 9
2.2 Certification-Based Replication . . . 12
2.3 Replication API . . . 14
2.4 Isolation Levels . . . 17
2.5 State Transfers . . . 20
2.6 Flow Control . . . 23
2.7 Node Failure & Recovery . . . 26
2.8 Quorum Components . . . 28
2.9 Streaming Replication . . . 33

3 Installing Galera Cluster **36**
3.1 Galera Cluster for MySQL—Binary Installation . . . 38
3.2 Galera Cluster for MySQL - Source Installation . . . 43
3.3 MariaDB Galera Cluster - Binary Installation . . . 47
3.4 MariaDB Galera Cluster - Source Installation . . . 50
3.5 Percona XtraDB Cluster - Binary Installation . . . 54
3.6 Percona XtraDB Cluster - Source Installation . . . 57

4 Galera Cluster Administration **62**
4.1 Node Provisioning . . . 65
4.2 State Snapshot Transfers . . . 67
4.2.1 Logical State Snapshot . . . 68
4.2.2 Physical State Snapshot . . . 71
4.3 Scriptable State Snapshot Transfers . . . 74
4.4 Galera System Tables . . . 77
4.5 Schema Upgrades . . . 83
4.6 Upgrading Galera Cluster . . . 86
4.7 Recovering Primary Component . . . 91
4.8 Resetting the Quorum . . . 93
4.9 Managing Flow Control . . . 96
4.10 Auto-Eviction . . . 100
4.11 Using Streaming Replication . . . 103
4.12 Galera Arbitrator . . . 105
4.13 Backing Up Cluster Data . . . 109

5 Deployment **111**
5.1 Cluster Deployment Variants . . . 112

5.2 Load Balancing . . . 118
5.2.1 HAProxy . . . 119
5.2.2 Pen Load Balancer . . . 121
5.2.3 Galera Load Balancer (Galera Load Balancer binaries are part of Galera Cluster Enterprise Edition) . . . 123
5.3 Container Deployments . . . 127
5.3.1 Using Docker . . . 128
5.3.2 Using Jails . . . 132

6 Cluster Monitoring **138**
6.1 Using Status Variables . . . 139
6.2 Database Server Logs . . . 145
6.3 The Galera Manager . . . 147
6.3.1 Installing Galera Manager . . . 150
6.3.2 AWS Ports with Galera Manager . . . 157
6.3.3 Galera Manager End-User License Agreement (EULA) . . . 161
6.3.4 Galera Manager Daemon (`gmd`) . . . 163
6.3.5 Deploying a Cluster in Galera Manager . . . 167
6.3.6 Adding Nodes with Galera Manager . . . 175
6.3.7 Adding Users to Galera Manager . . . 182
6.3.8 Loading Initial Data . . . 185
6.3.9 Monitoring a Cluster with Galera Manager . . . 191
6.3.10 Upgrading Galera Manager (`gmd`) . . . 199
6.4 Notification Command . . . 201
6.5 Notification Script Example . . . 203

7 Security **207**
7.1 Firewall Settings . . . 208
7.1.1 Firewall Configuration with `iptables` . . . 209
7.1.2 Firewall Configuration with FirewallD . . . 211
7.1.3 Firewall Configuration with PF . . . 213
7.2 SSL Settings . . . 215
7.2.1 SSL Certificates . . . 216
7.2.2 SSL Configuration . . . 218
7.2.3 SSL for State Snapshot Transfers . . . 221
7.3 SELinux Configuration . . . 225

8 Reference **229**
8.1 MySQL wsrep Options . . . 231
8.2 Galera Functions . . . 268
8.3 Galera Parameters . . . 271
8.3.1 Setting Galera Parameters in MySQL . . . 306
8.4 Galera Status Variables . . . 307
8.5 XtraBackup-v2 Parameters . . . 335
8.6 Galera Load Balancer Parameters . . . 343
8.7 Versioning Information . . . 352
8.8 Legal Notice . . . 354
8.9 Glossary . . . 355

Index **358**

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

This is the Codership Documentation. It documents the latest version of Galera Cluster, as well as related Galera tools, such as the Galera Arbitrator. It also includes, at times, information on features available in upcoming versions of Galera Cluster that haven't been released yet. For such text, the new version number is noted.

Installation & Configuration

Database Replication (page 9)	*State Transfers* (page 20)
Replication API (page 14)	*Flow Control* (page 23)
Installing Galera Cluster (page 36)	*Node Failure & Recovery* (page 26)
Certification-Based Replication (page 12)	*Quorum Components* (page 28)
Isolation Levels (page 17)	*Streaming Replication* (page 33)

Administration

Node Provisioning (page 65)	*Recovering Primary Component* (page 91)
State Snapshot Transfers (page 67)	*Resetting the Quorum* (page 93)
Scriptable State Snapshot Transfers (page 74)	*Managing Flow Control* (page 96)
Galera System Tables (page 77)	*Auto-Eviction* (page 100)
Schema Upgrades (page 83)	*Using Streaming Replication* (page 103)
Upgrading Galera Cluster (page 86)	*Backing Up Cluster Data* (page 109)
crash-recovery	inconsistency-voting
utility-scripts	

Deployment

Load Balancing (page 118)	*Cluster Deployment Variants* (page 112)
Container Deployments (page 127)	*Galera Arbitrator* (page 105)
ldap-plugin	pam-plugin
keyring-plugin	supported-oss

Monitoring

Using Status Variables (page 139)	*Database Server Logs* (page 145)
The Galera Manager (page 147)	*Security* (page 207)
Notification Command (page 201)	audit-log-plugin

Reference

MySQL wsrep Options (page 231)	*Galera Load Balancer Parameters* (page 343)
Galera Functions (page 268)	*XtraBackup-v2 Parameters* (page 335)
Galera Parameters (page 271)	*Galera System Tables* (page 77)
Galera Status Variables (page 307)	*Versioning Information* (page 352)
mariadb-options	mariabackup-options

Miscellaneous

Glossary (page 355)	*Legal Notice* (page 354)
genindex	../whats-new

For resolving problems you might have with the software, you can also check our Knowledge Base. There you will find troubleshooting and best practices articles. You can also post questions on the Codership Forum. The community, as well as our staff monitor and respond to posts made there.

If you need more immediate and personalized assistance, you can get a Support contract with us at Codership. For a quote on the cost of support, write us at info@codership.com or use our on-line form to send us a message.

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search
- Home

- *Docs* (page 1)
- KB
- Training
- FAQ

CHAPTER

ONE

OVERVIEW OF GALERA CLUSTER

Galera Cluster is a synchronous multi-primary database cluster, based on synchronous replication and MySQL and InnoDB. When Galera Cluster is in use, database reads and writes can be directed to any node. Any individual node can be lost without interruption in operations and without using complex failover procedures.

At a high level, Galera Cluster consists of a database server (that is, MySQL or MariaDB) that uses the *Galera Replication Plugin* to manage replication. To be more specific, the MySQL replication plugin API has been extended to provide all the information and hooks required for true multi-primary, synchronous replication. This extended API is called the Write-Set Replication API, or wsrep API.

Through the wsrep API, Galera Cluster provides certification-based replication. A transaction for replication, the write-set not only contains the database rows to replicate, but also includes information on all of the locks that were held by the database during the transaction. Each node then certifies the replicated write-set against other write-sets in the applier queue. The write-set is then applied—if there are no conflicting locks. At this point, the transaction is considered committed, after which each node continues to apply it to the tablespace.

This approach is also called virtually synchronous replication, given that while it is logically synchronous, the actual writing and committing to the tablespace happens independently, and thus asynchronously on each node.

Benefits of Galera Cluster

Galera Cluster provides a significant improvement in high-availability for the MySQL system. The various ways to achieve high-availability have typically provided only some of the features available through Galera Cluster, making the choice of a high-availability solution an exercise in trade-offs.

The following features are available through Galera Cluster:

- **True Multi-Primary**

  You can read and write to any node at any time. Changes to data on one node will be replicated on all.

- **Synchronous Replication**

  There is no replica lag, so no data is lost if a node crashes.

- **Tightly Coupled**

  All nodes hold the same state. There is no diverged data between nodes.

- **Multi-Threaded Replica**

  This allows for better performance and for any workload.

- **No Primary-Replica Failover**

  There is no need for primary/replica operations or to use Virtual IPs (VIP).

- **Hot Standby**

  There is no downtime related to failures or intentionally taking down a node for maintenance since there is no failover.

- **Automatic Node Provisioning**

  There's no need to backup manually the database and copy it to the new node.

- **Supports InnoDB.**

  The InnoDB storage engine provides for transactional tables.

- **Transparent to Applications**

  Generally, you won't have to change an application that will interface with the database as a result of Galera. If you do, it will be minimal changes.

- **No Read and Write Splitting Needed**

  There is no need to split read and write queries.

In summary, Galera Cluster is a high-availability solution that is both robust in terms of data integrity and provides high-performance with instant failovers.

Cloud Implementations with Galera Cluster

An additional benefit of Galera Cluster is good cloud support. Automatic node provisioning makes elastic scale-out and scale-in operations painless. Galera Cluster has been proven to perform extremely well in the cloud, such as when using multiple small node instances, across multiple data centers—AWS zones, for example—or even over Wider Area Networks.

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Certification Replication* (page 12)
- *Database Replication* (page 9)
- *Flow Control* (page 23)
- *Isolation Levels* (page 17)
- *Node Recovery* (page 26)
- *Quorum Components* (page 28)

- *Replication Architecture* (page 14)
- *State Transfers* (page 20)
- *Streaming Replication* (page 33)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

CHAPTER

TWO

TECHNICAL DESCRIPTION

Galera Cluster is a synchronous certification-based replication solution for MySQL, MariaDB and Percona XtraDB. Cluster nodes are identical and fully representative of the cluster state. They allow for unconstrained transparent client access, acting as a single-distributed database server. In order to better understand Galera Cluster, this section provides detailed information on how it works and how you can benefit from it.

Understanding Replication

Replication in the context of databases refers to the frequent copying of data from one database server to another. These sections provide a high-level explanation of replication both in the general sense of how it works, as well as the particulars of how Galera Cluster implements these core concepts.

- *Database Replication* (page 9)

  This section explains how database replication works in general. It provides an overview of the problems inherent in the various replication implementations, including primary-replica, asynchronous and synchronous replication.

- *Certification-Based Replication* (page 12)

  Using group communications and transaction ordering techniques, certification-based replication allows for synchronous replication.

Understanding Galera Cluster

With a better grasp on how replication works, these pages provide a more detailed explanation of how Galera Cluster implements certification-based replication, including the specific architecture of the nodes, how they communicate with each other, as well as replicate data and manage the replication process.

- *Replication API* (page 14)

  While the above sections explain the abstract concepts surrounding certification-based replication, this section covers the specific architecture used by Galera Cluster in implementing write-set replication, including the wsrep API and the Galera Replication and Group Communication plug-ins.

- *Isolation Levels* (page 17)

  In a database system, the server will process concurrent transactions in isolation from each other. The level of isolation determines whether and how these transactions affect one another. This section provides an overview of the isolation levels supported by Galera Cluster.

- *State Transfers* (page 20)

  The actual process that nodes use to replicate data into each other is called provisioning. Galera Cluster supports two provisioning methods: State Snapshot Transfers and Incremental State Transfers. This section presents an overview of each.

- *Flow Control* (page 23)

  Galera Cluster manages the replication process using a feedback mechanism called Flow Control. This allows the node to pause and resume replication according to its performance needs and to prevent any node from lagging too far behind the others in applying transaction. This section provides an overview of Flow Control and the different states nodes can hold.

- *Node Failure & Recovery* (page 26)

  Nodes fail to operate when they lose their connection with the cluster. This can occur for various reasons, such as hardware failures, software crashes, or the loss of network connectivity. This section provides an overview of how nodes and the cluster cope with failure and how they may recover.

- *Quorum Components* (page 28)

  When nodes connect to each other, they form components. The Primary Component is a component that has *Quorum*: it carries the majority of nodes in the cluster. By default, each node represents one vote in quorum calculations. However, you can modify this feature in order to ensure certain stable nodes with strong connections carry a greater value. This section provides an overview of how Galera Cluster handles weighted values in quorum calculations.

- *Streaming Replication* (page 33)

  Normally, nodes transfer all replication and certification events on the transaction commit. With Streaming Replication, the nodes break the transaction into fragments. Then they certify, replicate and apply these replication fragments onto the replica nodes. This section describes Streaming Replication, how it works and the limitations of its use.

Related Documents

- *Certification Replication* (page 12)
- *Database Replication* (page 9)
- *Flow Control* (page 23)
- *Isolation Levels* (page 17)
- *Node Recovery* (page 26)
- *Quorum Components* (page 28)
- *Replication Architecture* (page 14)
- *State Transfers* (page 20)
- *Streaming Replication* (page 33)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Certification Replication* (page 12)
- *Database Replication* (page 9)
- *Flow Control* (page 23)
- *Isolation Levels* (page 17)
- *Node Recovery* (page 26)
- *Quorum Components* (page 28)
- *Replicaiton Architecture* (page 14)
- *State Transfers* (page 20)
- *Streaming Replication* (page 33)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

2.1 Database Replication

Database replication refers to the frequent copying of data from one node—a database on a server—into another. Think of a database replication system as a distributed database, where all nodes share the same level of information. This system is also known as a *database cluster*.

The database clients, such as web browsers or computer applications, do not see the database replication system, but they benefit from close to native DBMS (Database Management System) behavior.

Primaries and Replicas

Many DATABASE MANAGEMENT SYSTEMS (DBMS) replicate the database.

The most common replication setup uses a primary/replica relationship between the original data set and the copies.

In this system, the primary database server logs the updates to the data and propagates those logs through the network to the replicas. The replica database servers receive a stream of updates from the primary and apply those changes.

Another common replication setup uses multi-primary replication, where all nodes function as primaries.

In a multi-primary replication system, you can submit updates to any database node. These updates then propagate through the network to other database nodes. All database nodes function as primaries. There are no logs available and the system provides no indicators sent to tell you if the updates were successful.

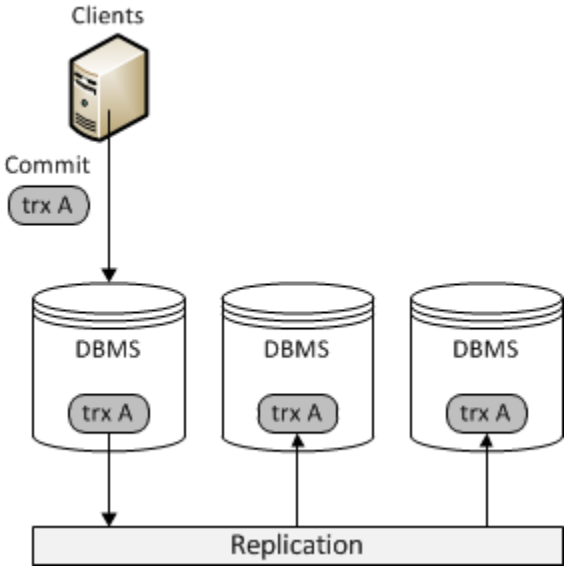


Fig. 1: *Primary/Primary Replication*

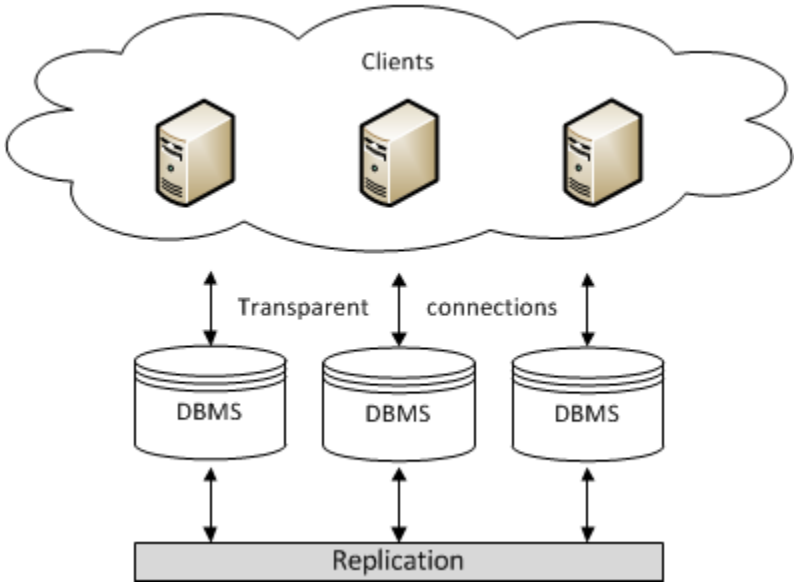


Fig. 2: *Multi-primary Replication*

Asynchronous and Synchronous Replication

In addition to the setup of how different nodes relate to one another, there is also the protocol for how they propagate database transactions through the cluster.

- **Synchronous Replication** Uses the approach of eager replication. Nodes keep all replicas synchronized by updating all replicas in a single transaction. In other words, when a transaction commits, all nodes have the same value.
- **Asynchronous Replication** Uses the approach of lazy replication. The primary database asynchronously propagates replica updates to other nodes. After the primary node propagates the replica, the transaction commits. In other words, when a transaction commits, for at least a short time, some nodes hold different values.

Advantages of Synchronous Replication

In theory, there are several advantages that synchronous replication has over asynchronous replication. For instance:

- **High Availability** Synchronous replication provides highly available clusters and guarantees 24/7 service availability, given that:
  - No data loss when nodes crash.
  - Data replicas remain consistent.
  - No complex, time-consuming failovers.
- **Improved Performance** Synchronous replications allows you to execute transactions on all nodes in the cluster in parallel to each other, increasing performance.
- **Causality across the Cluster** Synchronous replication guarantees causality across the whole cluster. For example, a `SELECT` query issued after a transaction always sees the effects of the transaction, even if it were executed on another node.

Disadvantages of Synchronous Replication

Traditionally, eager replication protocols coordinate nodes one operation at a time. They use a two phase commit, or distributed locking. A system with n number of nodes due to process o operations with a throughput of t transactions per second gives you m messages per second with:

$$m = n \times o \times t$$

What this means that any increase in the number of nodes leads to an exponential growth in the transaction response times and in the probability of conflicts and deadlock rates.

For this reason, asynchronous replication remains the dominant replication protocol for database performance, scalability and availability. Widely adopted open source databases, such as MySQL and PostgreSQL only provide asynchronous replication solutions.

Solving the Issues in Synchronous Replication

There are several issues with the traditional approach to synchronous replication systems. Over the past few years, researchers from around the world have begun to suggest alternative approaches to synchronous database replication.

In addition to theory, several prototype implementations have shown much promise. These are some of the most important improvements that these studies have brought about:

- **Group Communication** This is a high-level abstraction that defines patterns for the communication of database nodes. The implementation guarantees the consistency of replication data.

- **Write-sets** This bundles database writes in a single write-set message. The implementation avoids the coordination of nodes one operation at a time.
- **Database State Machine** This processes read-only transactions locally on a database site. The implementation updates transactions are first executed locally on a database site, on shallow copies, and then broadcast as a read-set to the other database sites for certification and possibly commits.
- **Transaction Reordering** This reorders transactions before the database site commits and broadcasts them to the other database sites. The implementation increases the number of transactions that successfully pass the certification test.

The certification-based replication system that Galera Cluster uses is built on these approaches.

Related Documents

- *Certification Replication* (page 12)
- *Database Replication* (page 9)
- *Flow Control* (page 23)
- *Isolation Levels* (page 17)
- *Node Recovery* (page 26)
- *Quorum Components* (page 28)
- *Replicaiton Architecture* (page 14)
- *State Transfers* (page 20)
- *Streaming Replication* (page 33)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

2.2 Certification-Based Replication

Certification-based replication uses group communication and transaction ordering techniques to achieve synchronous replication.

Transactions execute optimistically in a single node, or replica, and then at commit time, they run a coordinated certification process to enforce global consistency. It achieves global coordination with the help of a broadcast service that establishes a global total order among concurrent transactions.

Certification-Based Replication Requirements

It is not possible to implement certification-based replication for all database systems. It requires certain features of the database in order to work;

- **Transactional Database:** The database must be transactional. Specifically, it has to be able to rollback uncommitted changes.
- **Atomic Changes:** Replication events must be able to change the database, atomically. All of a series of database operations in a transaction must occur, else nothing occurs.
- **Global Ordering:** Replication events must be ordered globally. Specifically, they are applied on all instances in the same order.

How Certification-Based Replication Works

The main idea in certification-based replication is that a transaction executes conventionally until it reaches the commit point, assuming there is no conflict. This is called optimistic execution.

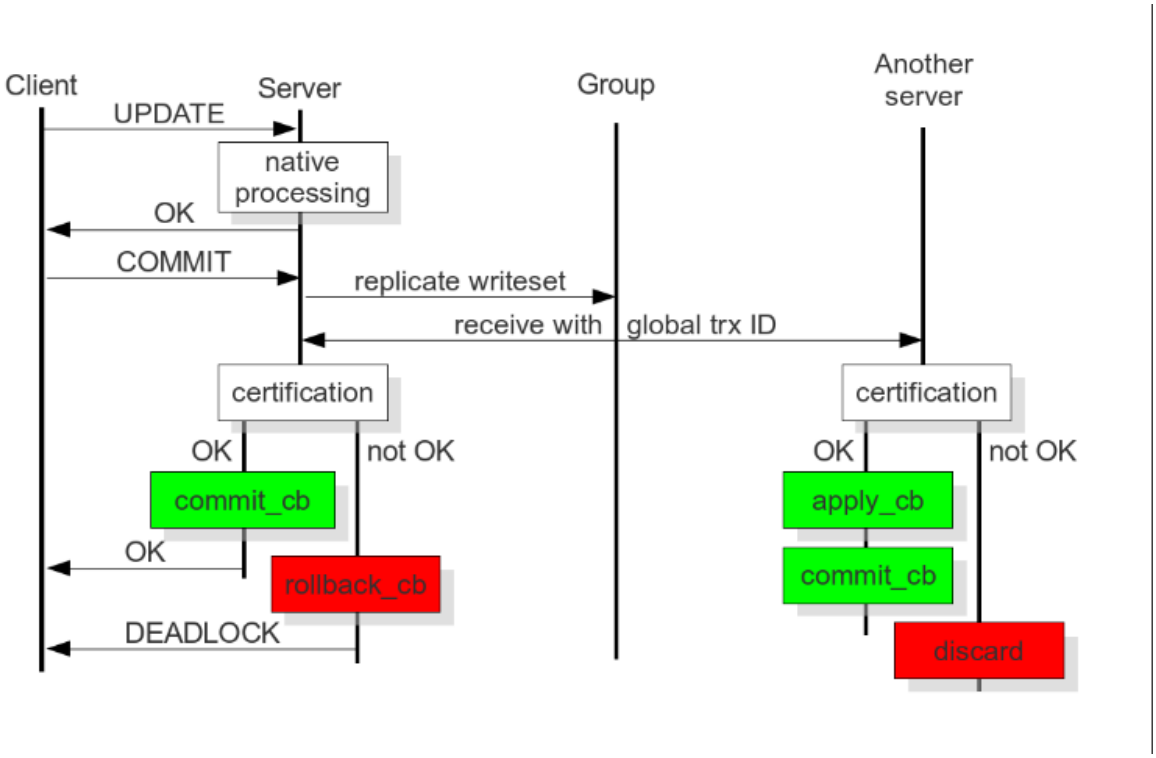


Fig. 3: *Certification Based Replication*

When the client issues a `COMMIT` command, but before the actual commit occurs, all changes made to the database by the transaction and primary keys of the changed rows, are collected into a write-set. The database then sends this write-set to all of the other nodes.

The write-set then undergoes a deterministic certification test, using the primary keys. This is done on each node in the cluster, including the node that originates the write-set. It determines whether or not the node can apply the write-set.

If the certification test fails, the node drops the write-set and the cluster rolls back the original transaction. If the test succeeds, though, the transaction commits and the write-set is applied to the rest of the cluster.

Certification-Based Replication in Galera Cluster

The implementation of certification-based replication in Galera Cluster depends on the global ordering of transactions.

Galera Cluster assigns each transaction a global ordinal sequence number, or `seqno`, during replication. When a transaction reaches the commit point, the node checks the sequence number against that of the last successful transaction. The interval between the two is the area of concern, given that transactions that occur within this interval have not seen the effects of each other. All transactions in this interval are checked for primary key conflicts with the transaction in question. The certification test fails if it detects a conflict.

The procedure is deterministic and all replica receive transactions in the same order. Thus, all nodes reach the same decision about the outcome of the transaction. The node that started the transaction can then notify the client application whether or not it has committed the transaction.

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

2.3 Replication API

Synchronous replication systems generally use eager replication. Nodes in a cluster will synchronize with all of the other nodes by updating the replicas through a single transaction. This means that when a transaction commits, all of the nodes will have the same value. This process takes place using *write-set* replication through group communication.

The internal architecture of Galera Cluster revolves around four components:

- **Database Management System (DBMS):** The database server that runs on an individual node. Galera Cluster can use MySQL, MariaDB or Percona XtraDB.
- **wsrep API:** This is the interface to the database server and it is the replication provider. It consists of two main elements:

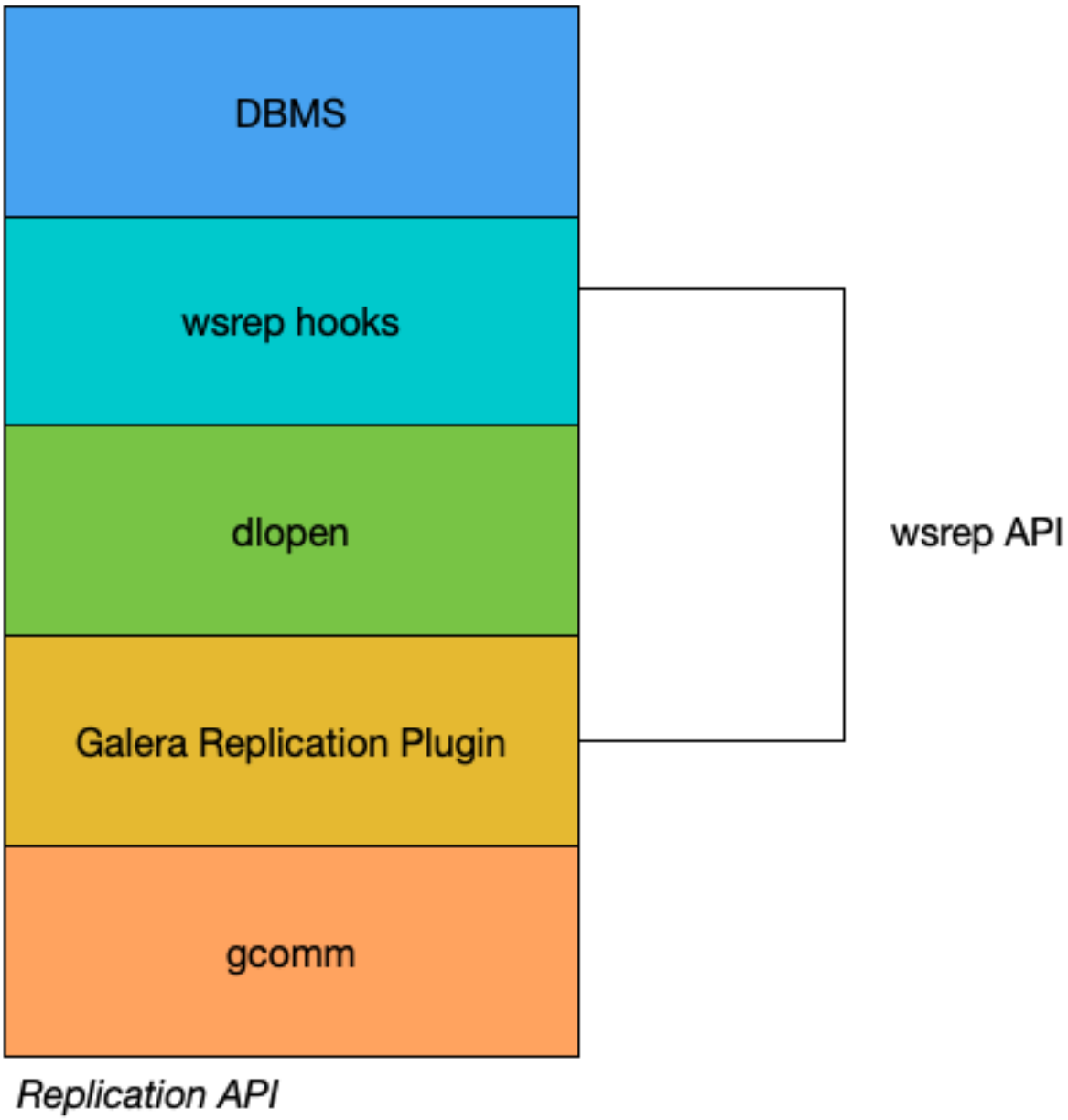


Fig. 4: *Replication API*

- *wsrep Hooks:* This integrates with the database server engine for write-set replication.
- *dlopen():* This function makes the wsrep provider available to the wsrep hooks.
- **Galera Replication Plugin:** This plugin enables write-set replication service functionality.

wsrep API

The *wsrep API* is a generic replication plugin interface for databases. It defines a set of application callbacks and replication plugin calls.

The wsrep API uses a replication model that considers the database server to have a state. That state refers to the contents of the database. When a database is in use and clients modify the database content, its state is changed. The wsrep API represents changes in the database state as a series of atomic changes, or transactions.

In a database cluster, all of the nodes always have the same state. They synchronize with each other by replicating and applying state changes in the same serial order.

From a more technical perspective, Galera Cluster handles state changes in the following way:

- On one node in the cluster, a state change occurs in the database.
- In the database, the wsrep hooks translate the changes to the write-set.
- `dlopen()` then makes the wsrep provider functions available to the wsrep hooks.
- The Galera Replication plugin handles write-set certification and replication to the cluster.

For each node in the cluster, the application process occurs by high-priority transactions.

Global Transaction ID

In order to keep the state identical across the cluster, the wsrep API uses a *Global Transaction ID*, or GTID. This allows it to identify state changes and to identify the current state in relation to the last state change. Below is an example of a GTID:

```
45eec521-2f34-11e0-0800-2a36050b826b:94530586304
```

The Global Transaction ID consists of the following components:

- **State UUID** This is a unique identifier for the state and the sequence of changes it undergoes.
- **Ordinal Sequence Number:** The seqno is a 64-bit signed integer used to denote the position of the change in the sequence.

The Global Transaction ID allows you to compare the application state and establish the order of state changes. You can use it to determine whether or not a change was applied and whether the change is applicable to a given state.

Galera Replication Plugin

The *Galera Replication Plugin* implements the *wsrep API*. It operates as the wsrep Provider. From a more technical perspective, the Galera Replication Plugin consists of the following components:

- **Certification Layer:** This layer prepares the write-sets and performs the certification checks on them, ensuring that they can be applied.
- **Replication Layer:** This layer manages the replication protocol and provides the total ordering capability.
- **Group Communication Framework:** This layer provides a plugin architecture for the various group communication systems that connect to Galera Cluster.

Group Communication Plugins

The Group Communication Framework provides a plugin architecture for the various gcomm systems.

Galera Cluster is built on top of a proprietary group communication system layer, which implements a virtual synchrony QoS (Quality of Service). Virtual synchrony unifies the data delivery and cluster membership services, providing clear formalism for message delivery semantics.

While virtual synchrony guarantees consistency, it does not guarantee temporal synchrony, which is necessary for smooth multi-primary operations. To address this, Galera Cluster implements its own runtime-configurable temporal flow control. Flow control keeps nodes synchronized to a fraction of a second.

Group Communication Framework also provides a total ordering of messages from multiple sources. It uses this to generate *Global Transaction ID*'s in a multi-primary cluster.

At the transport level, Galera Cluster is a symmetric undirected graph. All database nodes connect to each other over a TCP (Transmission Control Protocol) connection. By default, TCP is used for both message replication and the cluster membership services. However, you can also use UDP (User Datagram Protocol) multicast for replication in a LAN (Local Area Network).

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *READ-COMMITTED* (page 18)
- *READ-UNCOMMITTED* (page 18)
- *REPEATABLE-READ* (page 18)
- *SERIALIZABLE* (page 19)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

2.4 Isolation Levels

In a database system, concurrent transactions are processed in "isolation" from each other. The level of isolation determines how transactions can affect each other.

Intra-Node vs. Inter-Node Isolation in Galera Cluster

Before going into details about possible isolation levels which can be set for a client session in Galera Cluster it is important to make a distinction between single node and global cluster transaction isolation. Individual cluster nodes can provide any isolation level *to the extent* it is supported by MySQL/InnoDB. However isolation level *between* the nodes in the cluster is affected by replication protocol, so transactions issued on different nodes may not be isolated *identically* to transactions issued on the same node.

Overall isolation levels that are supported cluster-wide are

- *READ-UNCOMMITTED* (page 18)
- *READ-COMMITTED* (page 18)
- *REPEATABLE-READ* (page 18)

For transactions issued on different nodes, isolation is also strengthened by the "first committer wins" rule, which eliminates the "lost update anomaly" inherent to these levels, whereas for transactions issued on the same node this rule does not hold (as per original MySQL/InnoDB behavior). This makes for different outcomes depending on transaction origin (transaction issued on the same node may succeed, whereas the same transaction issued on another node would fail), but in either case it is no weaker than that isolation level on a standalone MySQL/InnoDB.

SERIALIZABLE (page 19) isolation level is honored only between transactions issued on the same node and thus should be avoided.

Data consistency between the nodes is always guaranteed regardless of the isolation level chosen by the client. However the client logic may break if it relies on an isolation level which is not not supported in the given configuration.

Understanding Isolation Levels

> **Warning:** When using Galera Cluster in primary-replica mode, all four levels are available to you, to the extent that MySQL supports it. In multi-primary mode, however, you can only use the `REPEATABLE-READ` level.

READ-UNCOMMITTED

Here transactions can see changes to data made by other transactions that are not yet committed.

In other words, transactions can read data that eventually may not exist, given that other transactions can always rollback the changes without commit. This is known as a dirty read. Effectively, `READ-UNCOMMITTED` has no real isolation at all.

READ-COMMITTED

Here dirty reads are not possible. Uncommitted changes remain invisible to other transactions until the transaction commits.

However, at this isolation level `SELECT` queries use their own snapshots of committed data, that is data committed before the `SELECT` query executed. As a result, `SELECT` queries, when run multiple times within the same transaction, can return different result sets. This is called a non-repeatable read.

REPEATABLE-READ

Here non-repeatable reads are not possible. Snapshots taken for the `SELECT` query are taken the first time the `SELECT` query runs during the transaction.

The snapshot remains in use throughout the entire transaction for the `SELECT` query. It always returns the same result set. This level does not take into account changes to data made by other transactions, regardless of whether or not they have been committed. In this way, reads remain repeatable.

SERIALIZABLE

Here all records accessed within a transaction are locked. The resource locks in a way that also prevents you from appending records to the table the transaction operates upon.

`SERIALIZABLE` prevents a phenomenon known as a phantom read. Phantom reads occur when, within a transaction, two identical queries execute, and the rows the second query returns differ from the first.

Related Documents

- *READ-COMMITTED* (page 18)
- *READ-UNCOMMITTED* (page 18)
- *REPEATABLE-READ* (page 18)
- *SERIALIZABLE* (page 19)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Galera Parameters* (page 271)
- *gcache.dir* (page 287)
- *gcache.recover* (page 288)
- *Incremental St. Transfr.* (page 20)
- *State Snapshot Transfers* (page 67)
- *State Snap. Transfr.* (page 20)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

2.5 State Transfers

The process of replicating data from the cluster to the individual node, bringing the node into sync with the cluster, is known as provisioning. There are two methods available in Galera Cluster to provision nodes:

- *State Snapshot Transfers (SST)* (page 20), where a snapshot of the entire node state transfers.
- *Incremental State Transfers (IST)* (page 20), where only the missing transactions transfer.

State Snapshot Transfer (SST)

In a *State Snapshot Transfer* (SST), the cluster provisions nodes by transferring a full data copy from one node to another. When a new node joins the cluster, the new node initiates a State Snapshot Transfer to synchronize its data with a node that is already part of the cluster.

You can choose from two conceptually different approaches in Galera Cluster to transfer a state from one database to another:

- **Logical** This method uses `mysqldump`. It requires that you fully initialize the receiving server and ready it to accept connections *before* the transfer.

  This is a blocking method. The *Donor Node* becomes `READ-ONLY` for the duration of the transfer. The State Snapshot Transfer applies the `FLUSH TABLES WITH READ LOCK` command on the donor node.

  `mysqldump` is the slowest method for State Snapshot Transfers. This can be an issue in a loaded cluster.

- **Physical** This method uses `rsync`, `rsync_wan`, `xtrabackup` and other methods, and copies the data files directly from server to server. It requires that you initialize the receiving server *after* the transfer.

  These methods are faster than `mysqldump`, but they have certain limitations. You can only use them on server startup. The receiving server requires very similar configurations to the donor, (for example, both servers must use the same `innodb_file_per_table` value. See innodb_file_per_table for version 8.0 or innodb_file_per_table for version 8.4.

  Some of these methods, such as `xtrabackup`, can be made non-blocking on the donor. They are supported through a scriptable SST interface.

For more information on the particular methods available for State Snapshot Transfers, see *State Snapshot Transfers* (page 67).

You can set which State Snapshot Transfer method a node uses from the confirmation file. For example:

```
wsrep_sst_method=rsync_wan
```

Incremental State Transfer (IST)

In an *Incremental State Transfer* (IST), the cluster provisions a node by identifying the missing transactions on the joiner and sends them only, instead of the entire state.

This provisioning method is only available under certain conditions:

- Where the *Joiner Node state UUID* is the same as that of the group.
- Where all missing write-sets are available in the donor's write-set cache.

When these conditions are met, the donor node transfers the missing transactions alone, replaying them in order until the joiner catches up with the cluster.

For example, say that you have a node in your cluster that falls behind the cluster. This node carries a node state that reads:

```
5a76ef62-30ec-11e1-0800-dba504cf2aab:197222
```

Meanwhile, the current node state on the cluster reads:

```
5a76ef62-30ec-11e1-0800-dba504cf2aab:201913
```

The donor node on the cluster receives the state transfer request from the joiner node. It checks its write-set cache for the *sequence number* `197223`. If that seqno is not available in the *write-set cache*, a State Snapshot Transfer initiates. If that seqno is available in the write-set cache, the donor node sends the commits from `197223` through to `201913` to the joiner, instead of the full state.

The advantage of Incremental State Transfers is that they can dramatically speed up the reemerging of a node to the cluster. Additionally, the process is non-blocking on the donor.

Note: The most important parameter for Incremental State Transfers is `gcache.size` on the donor node. This controls how much space you allocate in system memory for caching write-sets. The more space available the more write-sets you can store. The more write-sets you can store the wider the seqno gaps you can close through Incremental State Transfers.

On the other hand, if the write-set cache is much larger than the size of your database state, Incremental State Transfers become less efficient than sending a state snapshot.

Write-set Cache (GCache)

Galera Cluster stores write-sets in a special cache called the *Write-set Cache*, or GCache. GCache cache is a memory allocator for write-sets. Its primary purpose is to minimize the *write-set* footprint on the RAM (Random Access Memory). Galera Cluster improves upon this through the offload write-set storage to disk.

GCache employs three types of storage:

- **Permanent In-Memory Store** Here write-sets allocate using the default memory allocator for the operating system. This is useful in systems that have spare RAM. The store has a hard size limit.

  By default it is disabled.

- **Permanent Ring-Buffer File** Here write-sets pre-allocate to disk during cache initialization. This is intended as the main write-set store.

  By default, its size is 128 Mb.

- **On-Demand Page Store** Here write-sets allocate to memory-mapped page files during runtime as necessary.

  By default, its size is 128 Mb, but can be larger if it needs to store a larger write-set. The size of the page store is limited by the free disk space. By default, Galera Cluster deletes page files when not in use, but you can set a limit on the total size of the page files to keep.

  When all other stores are disabled, at least one page file remains present on disk.

For more information on parameters that control write-set caching, see the `gcache.*` parameters on *Galera Parameters* (page 271).

Galera Cluster uses an allocation algorithm that attempts to store write-sets in the above order. That is, first it attempts to use permanent in-memory store. If there is not enough space for the write-set, it attempts to store to the permanent ring-buffer file. The page store always succeeds, unless the write-set is larger than the available disk space.

By default, the write-set cache allocates files in the working directory of the process. You can specify a dedicated location for write-set caching, using the *gcache.dir* (page 287) parameter.

Note: Given that all cache files are memory-mapped, the write-set caching process may appear to use more memory than it actually does.

Note: If the *gcache.recover* (page 288) parameter is set to `yes`, an attempt to recover the gcache will be performed on startup, so that the node may continue to serve IST to other nodes. If set to `no`, gcache will be invalidated on startup and the node will only be able to serve SST.

Related Documents

- *Galera Parameters* (page 271)
- *gcache.dir* (page 287)
- *gcache.recover* (page 288)
- *Incremental St. Transfr.* (page 20)
- *State Snapshot Transfers* (page 67)
- *State Snap. Transfr.* (page 20)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Catching Up* (page 24)
- *Cluster Sync* (page 24)
- *gcs.fc_factor* (page 290)
- *gcs.fc_limit* (page 290)
- *gcs.max_throttle* (page 291)
- *gcs.recv_q_hard_limit* (page 291)
- *gcs.recv_q_soft_limit* (page 291)
- *No Flow Control* (page 23)
- *Write-set Caching* (page 23)
- *wsrep_ready* (page 331)
- Home
- *Docs* (page 1)

- KB
- Training
- FAQ

2.6 Flow Control

Galera Cluster manages the replication process using a feedback mechanism, called Flow Control. Flow Control allows a node to pause and resume replication according to its needs. This prevents any node from lagging too far behind the others in applying transactions.

How Flow Control Works

Galera Cluster achieves synchronous replication by ensuring that transactions copy to all nodes an execute according to a cluster-wide ordering. That said, the transaction applies and commits occur asynchronously as they replicate through the cluster.

Nodes receive write-sets and organize them into the global ordering. Transactions that the node receives from the cluster but which it has not applied and committed, are kept in the received queue.

When the received queue reaches a certain size the node triggers Flow Control. The node pauses replication, then works through the received queue. When it reduces the received queue to a more manageable size, the node resumes replication.

Understanding Node States

Galera Cluster implements several forms of Flow Control, depending on the node state. This ensures temporal synchrony and consistency—as opposed to logical, which virtual synchrony provides.

There are four primary kinds of Flow Control:

- *No Flow Control* (page 23)
- *Write-set Caching* (page 23)
- *Catching Up* (page 24)
- *Cluster Sync* (page 24)

No Flow Control

This Flow Control takes effect when nodes are in the `OPEN` or `PRIMARY` states.

When nodes hold these states, they are not considered part of the cluster. These nodes are not allowed to replicate, apply or cache any write-sets.

Write-set Caching

This Flow Control takes effect when nodes are in the `JOINER` and `DONOR` states.

Nodes cannot apply any write-sets while in this state and must cache them for later. There is no reasonable way to keep the node synchronized with the cluster, except for stopping all replication.

It is possible to limit the replication rate, ensuring that the write-set cache does not exceed the configured size. You can control the write-set cache with the following parameters:

- *gcs.recv_q_hard_limit* (page 291) Maximum write-set cache size (in bytes).
- *gcs.max_throttle* (page 291) Smallest fraction to the normal replication rate the node can tolerate in the cluster.
- *gcs.recv_q_soft_limit* (page 291) Estimate of the average replication rate for the node.

Catching Up

This Flow Control takes effect when nodes are in the `JOINED` state.

Nodes in this state can apply write-sets. Flow Control here ensures that the node can eventually catch up with the cluster. It specifically ensures that its write-set cache never grows. Because of this, the cluster wide replication rate remains limited by the rate at which a node in this state can apply write-sets. Since applying write-sets is usually several times faster than processing a transaction, nodes in this state hardly ever effect cluster performance.

The one occasion when nodes in the `JOINED` state do effect cluster performance is at the very beginning, when the buffer pool on the node in question is empty.

Note: You can significantly speed this up with parallel applying.

Cluster Sync

This Flow Control takes effect when nodes are in the `SYNCED` state.

When nodes enter this state Flow Control attempts to keep the replica queue to a minimum. You can configure how the node handles this using the following parameters:

- *gcs.fc_limit* (page 290) Used to determine the point where Flow Control engages.
- *gcs.fc_factor* (page 290) Used to determine the point where Flow Control disengages.

Changes in the Node State

The node state machine handles different state changes on different layers of Galera Cluster. These are the node state changes that occur at the top most layer:

1. The node starts and establishes a connection to the *Primary Component*.
2. When the node succeeds with a state transfer request, it begins to cache write-sets.
3. The node receives a *State Snapshot Transfer*. It now has all cluster data and begins to apply the cached write-sets.

   Here the node enables Flow Control to ensure an eventual decrease in the replica queue.
4. The node finishes catching up with the cluster. Its replica queue is now empty and it enables Flow Control to keep it empty.

   The node sets the MySQL status variable *wsrep_ready* (page 331) to the value `1`. The node is now allowed to process transactions.
5. The node receives a state transfer request. Flow Control relaxes to `DONOR`. The node caches all write-sets it cannot apply.
6. The node completes the state transfer to *Joiner Node*.

For the sake of legibility, certain transitions were omitted from the above description. Bear in mind the following points:

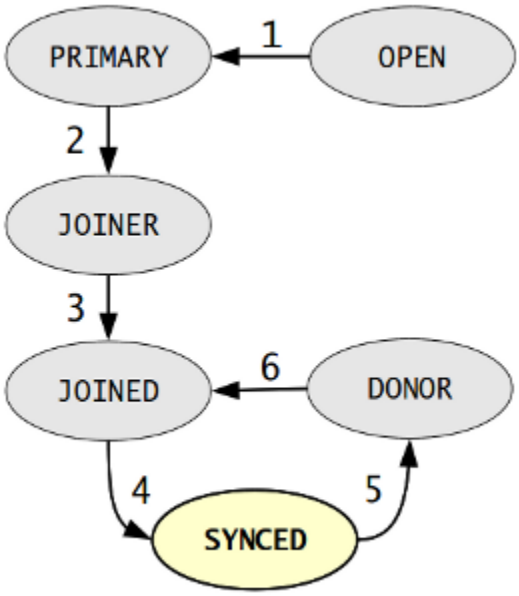


Fig. 5: *Galera Cluster Node State Changes*

- **Connectivity** Cluster configuration change events can send a node in any state to `PRIMARY` or `OPEN`. For instance, a node that is `SYNCED` reverts to `OPEN` when it loses its connection to the Primary Component due to network partition.
- **Missing Transitions** In the event that the joining node does not require a state transfer, the node state changes from the `PRIMARY` state directly to the `JOINED` state.

For more information on Flow Control see Galera Flow Control in Percona XtraDB Cluster.

Related Documents

- *Catching Up* (page 24)
- *Cluster Sync* (page 24)
- *gcs.fc_factor* (page 290)
- *gcs.fc_limit* (page 290)
- *gcs.max_throttle* (page 291)
- *gcs.recv_q_hard_limit* (page 291)
- *gcs.recv_q_soft_limit* (page 291)
- *No Flow Control* (page 23)
- *Write-set Caching* (page 23)
- *wsrep_ready* (page 331)

The Library

- *Documentation* (page 1)
- Knowledge Base

- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *evs.consensus_timeout* (page 280)
- *evs.inactive_check_period* (page 282)
- *evs.inactive_timeout* (page 282)
- *evs.keepalive_period* (page 284)
- *evs.suspect_timeout* (page 285)
- *Monitoring the Cluster* (page 139)
- *Notification Command* (page 201)
- *wsrep_local_state* (page 327)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

2.7 Node Failure & Recovery

Individual nodes fail to operate when they lose touch with the cluster. This can occur due to various reasons. For instance, in the event of hardware failure or software crash, the loss of network connectivity or the failure of a state transfer. Anything that prevents the node from communicating with the cluster is generalized behind the concept of node failure. Understanding how nodes fail will help in planning for their recovery.

Detecting Single Node Failures

When a node fails the only sign is the loss of connection to the node processes as seen by other nodes. Thus nodes are considered failed when they lose membership with the cluster's *Primary Component*. That is, from the perspective of the cluster when the nodes that form the Primary Component can no longer see the node, that node is failed. From the perspective of the failed node itself, assuming that it has not crashed, it has lost its connection with the Primary Component.

Although there are third-party tools for monitoring nodes—such as ping, Heartbeat, and Pacemaker—they can be grossly off in their estimates on node failures. These utilities do not participate in the Galera Cluster group communications and remain unaware of the Primary Component.

If you want to monitor the Galera Cluster node status poll the *wsrep_local_state* (page 327) status variable or through the *Notification Command* (page 201).

For more information on monitoring the state of cluster nodes, see the chapter on *Monitoring the Cluster* (page 139).

The cluster determines node connectivity from the last time it received a network packet from the node. You can configure how often the cluster checks this using the *evs.inactive_check_period* (page 282) parameter. During the check, if the cluster finds that the time since the last time it received a network packet from the node is greater than the value of the *evs.keepalive_period* (page 284) parameter, it begins to emit heartbeat beacons. If the cluster continues to receive no network packets from the node for the period of the *evs.suspect_timeout* (page 285) parameter, the node is declared suspect. Once all members of the Primary Component see the node as suspect, it is declared inactive—that is, failed.

If no messages were received from the node for a period greater than the *evs.inactive_timeout* (page 282) period, the node is declared failed regardless of the consensus. The failed node remains non-operational until all members agree on its membership. If the members cannot reach consensus on the liveness of a node, the network is too unstable for cluster operations.

The relationship between these option values is:

evs.keepalive_period (page 284)	<=	*evs.inactive_check_period* (page 282)
evs.inactive_check_period (page 282)	<=	*evs.suspect_timeout* (page 285)
evs.suspect_timeout (page 285)	<=	*evs.inactive_timeout* (page 282)
evs.inactive_timeout (page 282)	<=	*evs.consensus_timeout* (page 280)

Note: Unresponsive nodes that fail to send messages or heartbeat beacons on time—for instance, in the event of heavy swapping—may also be pronounced failed. This prevents them from locking up the operations of the rest of the cluster. If you find this behavior undesirable, increase the timeout parameters.

Cluster Availability vs. Partition Tolerance

Within the CAP theorem, Galera Cluster emphasizes data safety and consistency. This leads to a trade-off between cluster availability and partition tolerance. That is, when using unstable networks, such as WAN (Wide Area Network), low *evs.suspect_timeout* (page 285) and *evs.inactive_timeout* (page 282) values may result in false node failure detections, while higher values on these parameters may result in longer availability outages in the event of actual node failures.

Essentially what this means is that the *evs.suspect_timeout* (page 285) parameter defines the minimum time needed to detect a failed node. During this period, the cluster is unavailable due to the consistency constraint.

Recovering from Single Node Failures

If one node in the cluster fails, the other nodes continue to operate as usual. When the failed node comes back online, it automatically synchronizes with the other nodes before it is allowed back into the cluster.

No data is lost in single node failures.

State Transfer Failure

Single node failures can also occur when a *state snapshot transfer* fails. This failure renders the receiving node unusable, as the receiving node aborts when it detects a state transfer failure.

When the node fails while using `mysqldump`, restarting may require you to manually restore the administrative tables. For the `rsync` method in state transfers this is not an issue, given that it does not require the database server to be in an operational state to work.

Related Documents

- *evs.consensus_timeout* (page 280)
- *evs.inactive_check_period* (page 282)
- *evs.inactive_timeout* (page 282)
- *evs.keepalive_period* (page 284)
- *evs.suspect_timeout* (page 285)
- *Monitoring the Cluster* (page 139)
- *Notification Command* (page 201)
- *wsrep_local_state* (page 327)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *evs.suspect_timeout* (page 285)
- *Galera Arbitrator* (page 105)
- *pc.weight* (page 300)
- *Reset Quorum* (page 93)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

2.8 Quorum Components

In addition to single node failures, the cluster may split into several components due to network failure. A component is a set of nodes that are connected to each other, but not to the nodes that form other components. In these situations, only one component can continue to modify the database state to avoid history divergence. This component is called the *Primary Component*.

Under normal operations, your Primary Component is the cluster. When cluster partitioning occurs, Galera Cluster invokes a special *Quorum* algorithm to select one component as the Primary Component. This guarantees that there is never more than one Primary Component in the cluster.

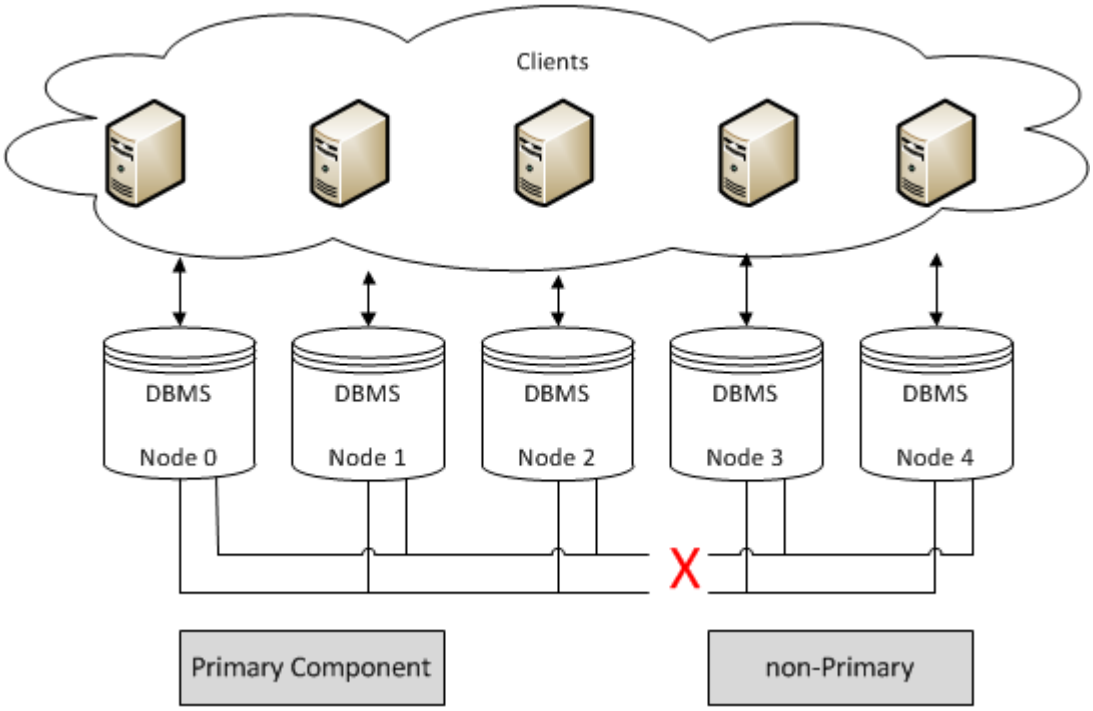
Note: In addition to the individual node, quorum calculations also take into account a separate process called `garbd`. For more information on its configuration and use, see *Galera Arbitrator* (page 105).

Weighted Quorum

The current number of nodes in the cluster defines the current cluster size. There is no configuration setting that allows you to define the list of all possible cluster nodes. Every time a node joins the cluster, the total cluster size increases. When a node leaves the cluster, gracefully, the cluster size decreases. Cluster size determines the number of votes required to achieve quorum.

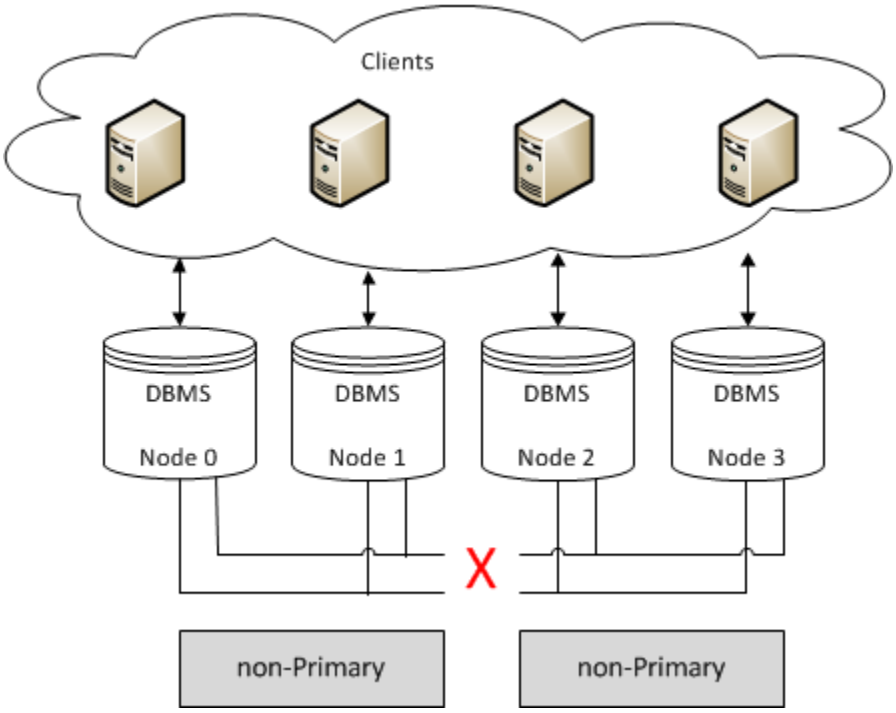
Galera Cluster takes a quorum vote whenever a node does not respond and is suspected of no longer being a part of the cluster. You can fine tune this no response timeout using the *evs.suspect_timeout* (page 285) parameter. The default setting is 5 seconds.

When the cluster takes a quorum vote, if the majority of the total nodes connected from before the disconnect remain, that partition stays up. When network partitions occur, there are nodes active on both sides of the disconnect. The component that has quorum alone continues to operate as the *Primary Component*, while those without quorum enter the non-primary state and begin attempt to connect with the Primary Component.



Quorum requires a majority, meaning that you cannot have automatic failover in a two node cluster. This is because the failure of one causes the remaining node automatically go into a non-primary state.

Clusters that have an even number of nodes risk split-brain conditions. If should you lose network connectivity somewhere between the partitions in a way that causes the number of nodes to split exactly in half, neither partition can retain quorum and both enter a non-primary state.



In order to enable automatic failovers, you need to use at least three nodes. Bear in mind that this scales out to other levels of infrastructure, for the same reasons.

- Single switch clusters should use a minimum of 3 nodes.
- Clusters spanning switches should use a minimum of 3 switches.
- Clusters spanning networks should use a minimum of 3 networks.
- Clusters spanning data centers should use a minimum of 3 data centers.

Split-Brain Condition

Cluster failures that result in database nodes operating autonomous of each other are called split-brain conditions. When this occurs, data can become irreparably corrupted, such as would occur when two database nodes independently update the same row on the same table. As is the case with any quorum-based system, Galera Cluster is subject to split-brain conditions when the quorum algorithm fails to select a *Primary Component*.

For example, this can occur if you have a cluster without a backup switch in the event that the main switch fails. Or, when a single node fails in a two node cluster.

By design, Galera Cluster avoids split-brain condition. In the event that a failure results in splitting the cluster into two partitions of equal size, (unless you explicitly configure it otherwise), neither partition becomes a Primary Component.

To minimize the risk of this happening in clusters that do have an even number of nodes, partition the cluster in a way that one component always forms the *Primary Cluster* section.

```
4 node cluster -> 3 (Primary) + 1 (Non-primary)
6 node cluster -> 4 (Primary) + 2 (Non-primary)
6 node cluster -> 5 (Primary) + 1 (Non-primary)
```

In these partitioning examples, it is very difficult for any outage or failure to cause the nodes to split exactly in half.

For more information on configuring and managing the quorum, see *Resetting the Quorum* (page 93).

Quorum Calculation

Galera Cluster supports a weighted quorum, where each node can be assigned a weight in the 0 to 255 range, with which it will participate in quorum calculations.

The quorum calculation formula is

$$\frac{\sum_{p_i \times w_i} - \sum_{l_i \times w_i}}{2} < \sum_{m_i \times w_i}$$

Where:

- p_i Members of the last seen primary component;
- l_i Members that are known to have left gracefully;
- m_i Current component members; and,
- w_i Member weights.

What this means is that 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.

You can customize node weight using the *pc.weight* (page 300) parameter. By default, node weight is `1`, which translates to the traditional node count behavior.

You can change the node weight during runtime by setting the *pc.weight* (page 300) parameter.

```
SET GLOBAL wsrep_provider_options="pc.weight=3";
```

Galera Cluster applies the new weight on the delivery of a message that carries a weight. At the moment, there is no mechanism to notify the application of a new weight, but will eventually happen when the message is delivered.

Warning: If a group partitions at the moment when the weight-change message is delivered, all partitioned components that deliver weight-change messages in the transitional view will become non-primary components. Partitions that deliver messages in the regular view, will go through quorum computation with the applied weight when the subsequential transitional view is delivered. In other words, there is a corner case where the entire cluster can become non-primary component, if the weight changing message is sent at the moment when partitioning takes place. Recovering from such a situation should be done either by waiting for a re-merge or by inspecting which partition is most advanced and by bootstrapping it as a new Primary Component.

Weighted Quorum Examples

Now that you understand how quorum weights work, here are some examples of deployment patterns and how to use them.

Weighted Quorum for Three Nodes

When configuring quorum weights for three nodes, use the following pattern:

```
node1: pc.weight = 2
node2: pc.weight = 1
node3: pc.weight = 0
```

Under this pattern, killing `node2` and `node3` simultaneously preserves the *Primary Component* on `node1`. Killing `node1` causes `node2` and `node3` to become non-primary components.

Weighted Quorum for a Simple Primary-Replica Scenario

When configuring quorum weights for a simple primary-replica scenario, use the following pattern:

```
node1: pc.weight = 1
node2: pc.weight = 0
```

Under this pattern, if the primary `node` dies, `node2` becomes a non-primary component. However, in the event that `node2` dies, `node1` continues as the *Primary Component*. If the network connection between the nodes fails, `node1` continues as the Primary Component while `node2` becomes a non-primary component.

Weighted Quorum for a Primary and Multiple Replicas Scenario

When configuring quorum weights for a primary-replica scenario that features multiple replica nodes, use the following pattern:

```
node1: pc.weight = 1
node2: pc.weight = 0
node3: pc.weight = 0
...
noden: pc.weight = 0
```

Under this pattern, if `node1` dies, all remaining nodes end up as non-primary components. If any other node dies, the *Primary Component* is preserved. In the case of network partitioning, `node1` always remains as the Primary Component.

Weighted Quorum for a Primary and Secondary Site Scenario

When configuring quorum weights for primary and secondary sites, use the following pattern:

```
Primary Site:
  node1: pc.weight = 2
  node2: pc.weight = 2

Secondary Site:
  node3: pc.weight = 1
  node4: pc.weight = 1
```

Under this pattern, some nodes are located at the primary site while others are at the secondary site. In the event that the secondary site goes down or if network connectivity is lost between the sites, the nodes at the primary site remain the *Primary Component*. Additionally, either `node1` or `node2` can crash without the rest of the nodes becoming non-primary components.

Related Documents

- *evs.suspect_timeout* (page 285)
- *Galera Arbitrator* (page 105)

- *pc.weight* (page 300)
- *Reset Quorum* (page 93)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Limitations* (page 34)
- *Streaming Replication* (page 104)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

2.9 Streaming Replication

Under normal operation, the node performs all replication and certification events when a transaction commits. When working with small transactions this is fine. However, it poses an issue with long-running writes and changes to large data-sets.

In *Streaming Replication*, the node breaks the transaction into fragments, then certifies and replicates them on the replicas while the transaction is still in progress. Once certified, the fragment can no longer be aborted by conflicting transactions.

Additionally, Streaming Replication allows the node to process transaction write-sets greater than 2Gb.

Note: Streaming Replication is a new feature introduced in version 4.0 of Galera Cluster. Older versions do not support these operations.

When to Use Streaming Replication

In most cases, the normal method Galera Cluster uses in replication is sufficient in transferring data from a node to a cluster. *Streaming Replication* provides you with an alternative for situations in which this is not the case. Keep in mind that there are some limitations to its use. It is recommended that you only enable it at a session-level, and then only on specific transactions that require the feature.

For more information on the limitations to Streaming Replication, see *Limitations* (page 34).

Long-Running Write Transactions

When using normal replication, you may occasionally encounter issues with long-running write transactions.

The longer it takes for a node to commit a transaction, the greater the likelihood that the cluster will apply a smaller, conflicting transaction before the longer one can replicate to the cluster. When this happens, the cluster aborts the long-running transaction.

Using *Streaming Replication* on long-running transactions mitigates this situation. Once the node replicates and certifies a fragment, it is no longer possible for other transactions to abort it.

Certification keys are generated from record locks, therefore they do not cover gap locks or next key locks. If the transaction takes a gap lock, it is possible that a transaction, which is executed on another node, will apply a write set which encounters the gap log and will abort the streaming transaction.

Large Data Write Transactions

When using normal replication, the node locally processes the transaction and does not replicate the data until you commit. This can create problems when updating a large volume of data, especially on nodes with slower network connections.

Additionally, while replica nodes apply a large transaction, they cannot commit other transactions they receive, which may result in Flow Control throttling of the entire cluster.

With *Streaming Replication*, the node begins to replicate the data with each transaction fragment, rather than waiting for the commit. This allows you to spread the replication over the lifetime of the transaction.

In the case of the replica nodes, after the replica applies a fragment, it is free to apply and commit other, concurrent transactions without blocking. This allows the replica node to process incrementally the entire large transaction with a minimal impact on the cluster.

Hot Records

In situations in which an application frequently updates one and the same records from the same table (for example, when implementing a locking scheme, a counter, or a job queue), you can use *Streaming Replication* to force critical updates to replicate to the entire cluster.

Running a transaction in this way effectively locks the hot record on all nodes, preventing other transactions from modifying the row. It also increases the chances that the transaction will commit successfully and that the client in turn will receive the desired outcome.

For more information and an example of how to implement Streaming Replication in situations such as this, see *Using Streaming Replication with Hot Records* (page 104).

Limitations

In deciding whether you want to use *Streaming Replication* with your application, consider the following limitations.

Performance During a Transaction

When you enable *Streaming Replication*, as of version 4 of Galera, each node in the cluster begins recording its write-sets to the `wsrep_streaming_log` table in the `mysql` database. Nodes do this to ensure the persistence of Streaming Replication updates in the event that they crash. However, this operation increases the load on the node, which may adversely affect its performance.

As such, it is recommended that you only enable Streaming Replication at a session-level and then only for transactions that would not run correctly without it.

Performance During Rollbacks

Occasionally, you may encounter situations in which the cluster needs to roll back a transaction while *Streaming Replication* is in use. In these situations, the rollback operation consumes system resources on all nodes.

When long-running write transactions frequently need to be rolled back, this can become a performance problem. Therefore, it is a good application design policy to use shorter transactions whenever possible. In the event that your application performs batch processing or scheduled housekeeping tasks, consider splitting these into smaller transactions in addition to using Streaming Replication.

Related Documents

- *Limitations* (page 34)
- *Streaming Replication* (page 104)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Articles

- Install Galera
- Install Galera on AWS

Other Resources

- Galera AWS (video)
- Galera MariaDB (video)
- Galera MySQL (video)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

CHAPTER

THREE

INSTALLING GALERA CLUSTER

Galera Cluster is essentially used to form a cluster among multiple database servers. It is widely used in conjunction with MySQL, MariaDB, and XtraDB database software systems. Galera Cluster is integral to these database systems. As a result, it may be installed together with one of them.

Note: FreeBSD is only available up to Galera Cluster for MySQL 5.7, not on 8.0 or later. As far as MariaDB is concerned, you can try the MariaDB port variants for FreeBSD, as the `wsrep` library actually runs on FreeBSD, but Codership does not maintain the patches for FreeBSD.

There are several methods available for installing the paired systems: you may use binary installation packages or install with the source files. Below is a list of the various pairs and links to how to use whichever method your prefer:

Installing MySQL Galera Cluster

MySQL the company and the database software was purchased quite a while ago by Oracle. They continue to support MySQL software and cooperate with Codership to deliver an excellent database cluster system.

MySQL Binary Installation

Click on the heading here to read this article on how to install MySQL using a binary installation package. Binary installation packages are available for Linux distributions using `apt-get` and `yum` package managers through the Codership repository.

MySQL Source Installation

If you are using a Linux distribution for which we do not have binary files that work with its package management system, or if your server uses a different unix-like operating system (for example, Solaris or FreeBSD), you will need to build Galera Cluster for MySQL from source files.

Installing MariaDB Galera Cluster

MariaDB the company and the database software is somewhat of a spinoff or fork of MySQL. The software is basically the same as MySQL; Some people who worked formerly at MySQL, founded MariaDB several years ago. Because of all of this, MariaDB software works well with Galera. In fact, starting with version 10.4 of MariaDB, Galera is included. Before that version, you will have to use one our binary installation packages or install from the source files.

MariaDB Binary Installation

This article provides information on how to install MariaDB using a binary installation package. They're available for Debian-based and RPM-based distributions of Linux, from the MariaDB Repository Generator.

MariaDB Source Installation

If there aren't a binary installation packages that are suited to the distribution of Linux your servers are using, or you are using a different unix-like operating system (for example, Solaris or FreeBSD), you will have to build MariaDB Galera Cluster from the source files.

Installing XtraDB Galera Cluster

Many years before MariaDB was formed and several years before MySQL was bought by Oracle, some key personnel at MySQL, who specialized in performance tuning MySQL software, left to form Percona—the name is an amalgamation of the words, *Performance* and *Consulting*. In their efforts to get the most out of MySQL software, they developed their own fork with some extra performance enhancements, called XtraDB. It also works well with Galera Cluster.

XtraDB Binary Installation

Binary packages for installing XtraDB with Galera Cluster are available for Debian-based and RPM-based distributions, but through the Percona repository. This article explains how to install and configure this pairing of software, as well as provides links to the repository.

XtraDB Source Installation

You may not be able to use one of the binary installation packages available because of your operating system. If so, you will have to use our source files. Actually, you may want to use the source files to make minor changes that will become part of the files you will build.

Related Articles

- Install Galera
- Install Galera on AWS

Other Resources

- Galera AWS (video)
- Galera MariaDB (video)
- Galera MySQL (video)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ

- search

Related Documents

- *Install Galera* (page 36)
- *Galera MySQL Source* (page 43)
- *MySQL Shared Compatibility Libraries* (page 41)
- *Galera MariaDB Binaries* (page 47)

Related Articles

- ../training/tutorials/migration

Other Resources

- Galera AWS (video)
- Galera MySQL (video)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

3.1 Galera Cluster for MySQL—Binary Installation

Galera Cluster for MySQL may be installed on Linux servers using binary packages. These files can be downloaded directly from the Codership repository, or by way of a package manager: `apt-get` or `yum`.

Enabling the Codership Repository

To install Galera Cluster for MySQL with a package manager, you first will have to enable the Codership repository on the server. There are a few ways to do this, depending on which Linux distribution and package manager you use. The sections below provide details on how to use each of the three main supported package managers to install Galera Cluster for MySQL.

Enabling the `apt` Repository

For Debian and Debian-based Linux distributions, the procedure for adding a repository requires that you first install the *Software Properties*. The package names vary depending on the distribution. For Debian, at the command-line, execute the following command:

```
apt-get install python-software-properties
```

For Ubuntu or a distribution derived from Ubuntu, you would execute instead the following:

```
apt-get install software-properties-common
```

If your server uses a different Debian-based distribution, and neither of these commands work on your system, try consulting your distribution's package listings for the appropriate package name.

Once you have the *Software Properties* installed, you can then enable the Codership repository for your system. Start by adding the GnuPG key for the repository. This is done by executing the following from the command-line:

```
apt-key adv --keyserver keyserver.ubuntu.com --recv 8DA84635
```

Note: For packages before MySQL 5.7.44 and 8.0.35, the signing key is BC19DDBA. Next, add the Codership repository to your sources list. Using a simple text editor, create file called, *galera.list* in the `/etc/apt/sources.list.d/` directory. Add these lines to that file, with the necessary adjustments for the version used:

```
# Codership Repository (Galera Cluster for MySQL)
deb https://releases.galeracluster.com/mysql-wsrep-VERSION/DIST RELEASE main
deb https://releases.galeracluster.com/galera-3/DIST RELEASE main
```

In the example above, you would change the repository addresses. The `VERSION` should be set to MySQL-wsrep version you want to install. For example, it might be something like, `8.4`. The `DIST` should be replaced with the name of the Linux distribution on the server. This could be `ubuntu`. Last, replace `RELEASE` with the distribution release (for example, `wheezy`).

If you do not know which release you have installed on your server, you can generally find this using the following command:

```
lsb_release -a
```

Version 4 of Galera was recently released. If you'd like to install it, the configuration lines in *galera.list* should read similar to the following:

```
# Codership Repository (Galera Cluster for MySQL)
deb https://releases.galeracluster.com/galera-4/ubuntu focal main
deb https://releases.galeracluster.com/mysql-wsrep-8.0/ubuntu focal main
```

Again, you may have to adust the version and release numbers, depending on which you want to install. Please note that this will require at least version 18.04 of Ubuntu.

To be assured the proper version is installed and updated, set which repository you prefer to the Codership repository (this is not only recommended, it is required). To do this, using a text editor, create a file called, *galera.pref* in the `/etc/apt/preferences.d/` directory. The contents should look like the following:

```
# Prefer Codership repository
Package: *
Pin: origin releases.galeracluster.com
Pin-Priority: 1001
```

This is needed to make sure the patched versions are preferred. This might be important, for instance, if a third-party program requires `libmysqlclient20` and the OS-version for the library is newer.

Finally, you should update the local cache of the repository. Do this by entering the following from the command-line:

```
apt-get update
```

Once you've done all of these tasks, the packages in the Codership repository will be available for installation. For information on installing them using `apt-get`, skip ahead on this page to the section entitled, *Installing Galera Cluster for MySQL* (page 40).

Enabling the `yum` Repository

For rpm-based distributions of Linux (for example, CentOS and Red Hat Enterprise Linux), you will need to enable the Codership repository. Using a simple text editor, create a file called, `galera.repo` in the `/etc/yum.repos.d/` directory. The contents of that file should look something like the following:

```
[galera]
name = Galera
baseurl = https://releases.galeracluster.com/galera-3/DIST/RELEASE/ARCH
gpgkey = https://releases.galeracluster.com/GPG-KEY-galeracluster.com
gpgcheck = 1

[mysql-wsrep]
name = MySQL-wsrep
baseurl =  https://releases.galeracluster.com/mysql-wsrep-VERSION/DIST/RELEASE/ARCH
gpgkey = https://releases.galeracluster.com/GPG-KEY-galeracluster.com
gpgcheck = 1
```

In this sample repository configuration file, you would change the repository addresses for the `baseurl`. The `VERSION` should be set to the whichever MySQL-wsrep version you want (for example, it might be `8.0`). The `DIST` should be changed to the name of the Linux distribution you are using on your server (for example, `redhat`). The `RELEASE` should be replaced with the distribution's release number. It might be `8` or `9` for CentOS and Red Hat Enterprise Linux. Last, the `ARCH` indicates the architecture of your hardware. This could be changed to `x86_64` for 64-bit systems.

Here is a sample repository configuration file for Red Hat 9 and Galera Cluster with MySQL 8.

```
[galera4]
name = Galera
baseurl = https://releases.galeracluster.com/galera-4/redhat/9/x86_64
gpgkey = https://releases.galeracluster.com/GPG-KEY-galeracluster.com
gpgcheck = 1

[mysql-wsrep8]
name = MySQL-wsrep
baseurl = https://releases.galeracluster.com/mysql-wsrep-8.0/redhat/9/x86_64
gpgkey = https://releases.galeracluster.com/GPG-KEY-galeracluster.com
gpgcheck = 1
```

After you've created, modified, and saved this repository file, you will be able to install the packages from the Codership repository using `yum`. For an explanation on installing, skip ahead on this page to the section entitled, *Installing Galera Cluster for MySQL* (page 40).

Installing Galera Cluster for MySQL

There are two packages involved in the installation of Galera Cluster for MySQL: the MySQL database server, but one that has been built to include the *wsrep API*; and the *Galera Replication Plugin*. The `yum` repositories include Galera Arbitrator with the Galera Replication Plugin, but for Debian-based distributions using `apt-get` you will need to include add it to your installation instruction.

Note: If SELinux (Security-Enhanced Linux) is enabled on the servers, disable it. See Disabling SELinux for mysqld.

Also, if AppArmor is enabled on the servers, disable it. See Disabling AppArmor. However, this is optional, and there are methods to enable the context files.

So, for Debian-based distributions using the `apt-get` package manager, execute the following from the command-line:

For Galera Cluster 8.0:

```
apt-get install galera-4 galera-arbitrator-4 mysql-wsrep-8.0
```

For Galera Cluster 8.4:

```
apt-get install galera-4 galera-arbitrator-4 mysql-wsrep-8.4
```

On servers using the `yum` package manager (that is, Red Hat Enterprise Linux and CentOS distributions), you would instead execute this command:

```
yum install galera-4 mysql-wsrep-8.4
```

For `mysql-wsrep-8.0`:

```
yum install galera-4 mysql-wsrep-8.0
```

Note: On Red Hat 9, this command may generate a transaction check error. For more information on that error and how to resolve it, see the section below on *MySQL Shared Compatibility Libraries* (page 41).

Please note that on Red Hat 8, you need to disable MySQL and MariaDB modules before installing Galera Cluster from a repository under https://releases.galeracluster.com/. In order to do this, execute the following from the command-line:

```
dnf module disable mysql mariadb
```

Once you've executed the line appropriate to your distribution and package manager, Galera Cluster for MySQL should be installed on your server. You will then have to repeat this process for each node in your cluster, including enabling the repository files mentioned earlier.

Incidentally, when deciding which packages from the Codership repository to install, the package manager may elect to install a newer major verion of Galera Cluster, newer than the one you intended to install. Before confirming the installation of packages, make sure that the package manager is planning to install the Galera Cluster version you want.

If you installed Galera Cluster for MySQL over an existing stand-alone instance of MySQL, there are some additional steps that you will need to take to update your system to the new database server. For more information, see ../training/tutorials/migration.

MySQL Shared Compatibility Libraries

When installing Galera Cluster for MySQL on CentOS 8 or 9, you may encounter a transaction check-error that blocks the installation. The error message may look something like this:

```
Transaction Check Error:
file /usr/share/mysql/czech/errmsg.sys from install
mysql-wsrep-server-5.6-5.6.23-25.10.e16.x86_64 conflicts
with file from package mysql-libs-5.1.73-.3.e16_5.x86_64
```

This relates to a dependency problem between the version of the MySQL shared compatibility libraries that CentOS uses, and the one that Galera Cluster requires. To resolve this, you will have to upgrade, which can be done with the Codership repository using `yum`.

There are two versions available for this package. Which version you will need will depend on which version of the MySQL wsrep database server you want to install.

For CentOS, you would enter something like the following from the command-line:

```
yum upgrade -y mysql-wsrep-libs-compat-VERSION
```

You would, of course, replace `VERSION` here with `8.0` or `8.1`, depending on the version of MySQL you want to use. For CentOS 8 or 9, to install MySQL version 8.0, you would execute the following from the command-line:

```
yum upgrade mysql-wsrep-shared-8.0
```

For CentOS 8 or 9, to install MySQL version 8.0, you will also need to disable the 8.0 upgrade. To do this, enter the following from the command-line:

```
yum upgrade -y mysql-wsrep-shared-8.0 -x mysql-wsrep-shared-8.0
```

When `yum` finishes the upgrade, you can then install the MySQL wsrep database server and the Galera Replication Plugin as described above.

Related Documents

- *Install Galera* (page 36)
- *Galera MySQL Source* (page 43)
- *MySQL Shared Compatibility Libraries* (page 41)
- *Galera MariaDB Binaries* (page 47)

Related Articles

- ../training/tutorials/migration

Other Resources

- Galera AWS (video)
- Galera MySQL (video)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Install Galera* (page 36)
- *Galera MySQL Binaries* (page 38)

- *Galera MariaDB Source* (page 50)
- *wsrep_provider* (page 254)

Related Articles

- ../training/tutorials/migration
- System Configuration
- Replication Configuration
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

3.2 Galera Cluster for MySQL - Source Installation

Galera Cluster for MySQL is the reference implementation from Codership Oy. Binary installation packages are available for Debian- and RPM-based distributions of Linux. If your Linux distribution is based upon a different package management system, if your server uses a different unix-like operating system, such as Solaris or FreeBSD, you will need to build Galera Cluster for MySQL from source.

Note: If you built Galera Cluster for MySQL over an existing standalone instance of MySQL, there are some additional steps that you need to take in order to update your system to the new database server. For more information, see ../training/tutorials/migration.

Installing Build Dependencies

When building from source code, `make` cannot manage or install dependencies for either Galera Cluster or the build process itself. You need to install these first. For Debian-based systems, run the following command:

```
# apt-get build-dep mysql-server
```

For RPM-based distributions, instead run this command:

```
# yum-builddep MySQL-server
```

If neither command works on your system or that you use a different Linux distribution or FreeBSD (FreeBSD is available up to Galera Cluster for MySQL 5.7), the following packages are required:

- **MySQL Database Server with wsrep API**: Git, CMake, GCC and GCC-C++, Automake, Autoconf, and Bison, as well as development releases of libaio and ncurses.
- **Galera Replication Plugin**: SCons, as well as development releases of Boost, Check and OpenSSL.

Check with the repositories for your distribution or system for the appropriate package names to use during installation. Bear in mind that different systems may use different names and that some may require additional packages to run. For instance, to run CMake on Fedora you need both `cmake` and `cmake-fedora`.

Building Galera Cluster for MySQL

The source code for Galera Cluster for MySQL is available through GitHub. You can download the source code from the website or directly using `git`. In order to build Galera Cluster, you need to download both the database server with the wsrep API patch and the *Galera Replication Plugin*.

To download the database server, complete the following steps:

1. Clone the Galera Cluster for MySQL database server source code.

   ```
   # git clone https://github.com/codership/mysql-wsrep
   ```

2. Checkout the branch for the version that you want to use.

   ```
   # git checkout 8.4
   ```

   The main branches available for Galera Cluster for MySQL are:

   - `8.0`
   - `8.4`

You now have the source files for the MySQL database server, including the wsrep API patch needed for it to function as a Galera Cluster node.

In addition to the database server, you need the wsrep Provider, also known as the Galera Replication Plugin. In a separator directory, run the following command:

```
# cd ..
# git clone https://github.com/codership/galera.git
```

Once Git finishes downloading the source files, you can start building the database server and the Galera Replication Plugin. The above procedures created two directories: `mysql-wsrep/` for the database server source and for the Galera source `galera/`

Building the Database Server

The database server for Galera Cluster is the same as that of the standard database servers for standalone instances of MySQL, with the addition of a patch for the wsrep API, which is packaged in the version downloaded from GitHub. You can enable the patch through the wsrep API, requires that you enable it through the `WITH_WSREP` and `WITH_INNODB_DISALLOW_WRITES` CMake configuration options.

To build the database server, `cd` into the `mysql-wsrep/` directory and run the following commands:

```
# cmake -DWITH_WSREP=ON -DWITH_INNODB_DISALLOW_WRITES=ON ./
# make
# make install
```

Building the wsrep Provider

The *Galera Replication Plugin* implements the *wsrep API* and operates as the wsrep Provider for the database server. What it provides is a certification layer to prepare write-sets and perform certification checks, a replication layer and a group communication framework.

To build the Galera Replicator plugin, `cd` into the `galera/` directory and run SCons:

```
# scons
```

This process creates the Galera Replication Plugin, (that is, the `libgalera_smm.so` file). In your `my.cnf` configuration file, you need to define the path to this file for the *wsrep_provider* (page 254) parameter.

Note: For FreeBSD users, building the Galera Replicator Plugin from source raises certain Linux compatibility issues. You can mitigate these by using the ports build at `/usr/ports/databases/galera`.

Post-installation Configuration

After the build completes, there are some additional steps that you must take in order to finish installing the database server on your system. This is over and beyond the standard configurations listed in System Configuration and Replication Configuration.

Note: Unless you defined the `CMAKE_INSTALL_PREFIX` configuration variable when you ran `cmake` above, by default the database server installed to the path `/usr/local/mysql/`. If you chose a custom path, adjust the commands below to accommodate the change.

1. Create the user and group for the database server.

   ```
   # groupadd mysql
   # useradd -g mysql mysql
   ```

2. Install the database.

   ```
   # cd /usr/local/mysql
   # ./scripts/mysql_install_db --user=mysql
   ```

   This installs the database in the working directory. That is, at `/usr/local/mysql/data/`. If you would like to install it elsewhere or run it from a different directory, specify the desired path with the `--basedir` and `--datadir` options.

3. Change the user and group for the directory.

   ```
   # chown -R mysql /usr/local/mysql
   # chgrp -R mysql /usr/local/mysql
   ```

4. Create a system unit.

   ```
   # cp /usr/local/mysql/supported-files/mysql.server \
        /etc/init.d/mysql
   # chmod +x /etc/init.d/mysql
   # chkconfig --add mysql
   ```

   This allows you to start Galera Cluster using the `service` command. It also sets the database server to start during boot.

In addition to this procedure, bear in mind that any custom variables you enabled during the build process, such as a nonstandard base or data directory, requires that you add parameters to cover this in the configuration file, (that is, `my.cnf`).

Note: This tutorial omits MySQL authentication options for brevity.

Related Documents

- *Install Galera* (page 36)
- *Galera MySQL Binaries* (page 38)
- *Galera MariaDB Source* (page 50)
- *wsrep_provider* (page 254)

Related Articles

- System Migration
- System Configuration
- Replication Configuration

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Install Galera* (page 36)
- *MariaDB Galera Source* (page 50)
- *Galera MySQL* (page 38)

Related Articles

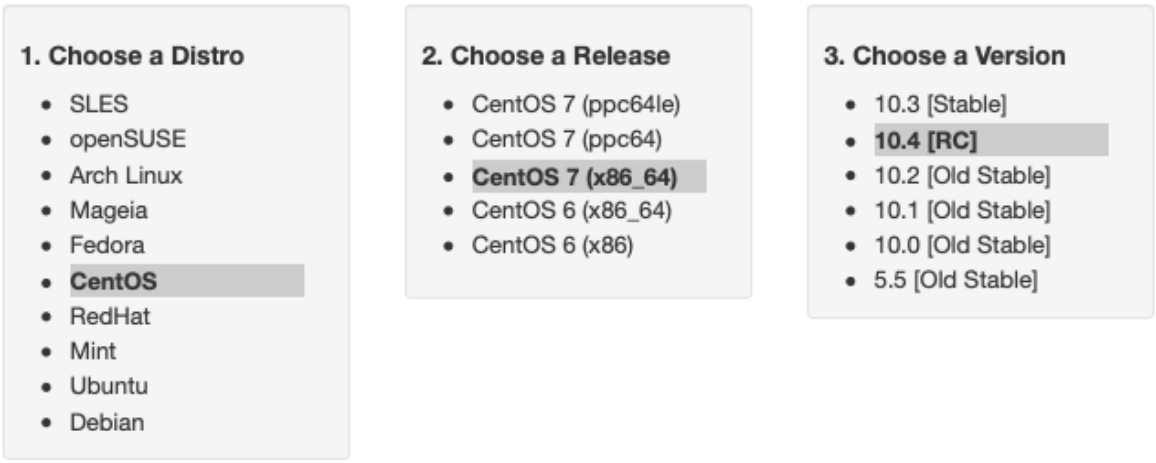
- ../training/tutorials/migration

Other Resources

- Galera AWS (video)
- Galera MariaDB (video)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

3.3 MariaDB Galera Cluster - Binary Installation

MariaDB Galera Cluster is the MariaDB implementation of Galera Cluster. Binary installation packages are available for Debian-based and RPM-based distributions of Linux through the MariaDB repository (MariaDB Repository Generator).



Here is your custom MariaDB YUM repository entry for CentOS. Copy and paste it into a file under /etc/yum.rep MariaDB.repo or something similar).

```
# MariaDB 10.4 CentOS repository list - created 2019-06-11 12:18 UTC
# http://downloads.mariadb.org/mariadb/repositories/
[mariadb]
name = MariaDB
baseurl = http://yum.mariadb.org/10.4/centos7-amd64
gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
gpgcheck=1
```

After the file is in place, install MariaDB with:

```
sudo yum install MariaDB-server MariaDB-client
```

Enabling the MariaDB Repository

In order to install MariaDB Galera Cluster through your package manager, you need to enable the MariaDB repository on your server. There are two different ways to accomplish this, depending on which Linux distribution you use.

Enabling the `apt` Repository

For Debian and Debian-based Linux distributions, the procedure for adding a repository requires that you first install the software properties. The package names vary depending on your distribution. For Debian, at the command-line execute the following:

```
# apt-get install python-software-properties
```

For Ubuntu or a distribution derived from Ubuntu, execute instead this command:

```
$ sudo apt-get install software-properties-common
```

If you are use a different Debian-based distribution and neither of these lines above work, consult your distribution's package listings for the appropriate package name.

Once you have the software properties installed, you can enable the MariaDB repository for your server.

First, add the GnuPG key for the MariaDB repository by executing the following from the command-line:

```
# apt-key adv --recv-keys --keyserver \
      keyserver.ubuntu.com 0xcbcb082a1bb943db
```

Next, add the MariaDB repository to your sources list. You can do this by entering something like the following from the command-line:

```
# add-apt-repository 'deb https://mirror.jmu.edu/pub/mariadb/repo/version/distro␣
↪release main'
```

You wouldn't enter exactly the line above. You'll have to adjust the repository address:

- `version` indicates the version number of MariaDB that you want to use. (for example, `10.4`).
- `distro` is the name of the Linux distribution you are using' (for example, `ubuntu`).
- `release` should be changed to your distribution release (for example, `wheezy`).

If you do not know which release is installed on your server, you can determine this by using the entering the following from the command-line:

```
$ lsb_release -a
```

1. You should also update the local cache on the server. You can do this by entering the following:

```
# apt-get update
```

For more information on the MariaDB repository, package names and available mirrors, see the MariaDB Repository Generator.

Packages in the MariaDB repository are now available for installation through `apt-get`.

Enabling the `yum` Repository

For RPM-based distributions (for example, CentOS and Red Hat Enterprise Linux), you can enable the MariaDB repository by creating a text file with `.repo` as the file extension to the `/etc/yum/repos.d/` directory.

Using a simple text editor, create a new `.repo` file containing something like the following:

```
# MariaDB.repo

[mariadb]
name = MariaDB
baseurl = https://yum.mariadb.org/version/package
gpgkey = httpss://yum.mariadb.org/RPM-GPG-KEY-MariaDB
gpgcheck = 1
```

For the value of `baseurl`, you will have to adjust the web address:

- `version` should be changed to the version of MariaDB you want to use (for example, `10.4`).

- `package` will have to be changed to the package name for your operating system distribution, release and architecture. For example, `rhel8-amd64` would reference packages for a Red Hat Enterprise Linux 8 server running on 64-bit hardware.

For more information on the repository, package names or available mirrors, see the MariaDB Repository Generator. It will generate the actual text you will need to put in your repository configuration file. In fact, by clicking through the choices presented, you can just copy the results and paste them into your configuration file without any modification.

Installing MariaDB Galera Cluster

There are three packages involved in the installation of MariaDB Galera Cluster: the MariaDB database client, a command-line tool for accessing the database; the MariaDB database server, built to include the *wsrep API* patch; and the *Galera Replication Plugin*.

For Debian-based distributions, from the command-line run the following commands:

```
# apt-get update
# apt-get install mariadb-server mariadb-client galera-4
```

Note: For MariaDB 10.3 and before, replace `galera-4` with `galera-3`.

For RPM-based distributions, first install the EPEL repository and the PV utility:

- For CentOS:

  ```
  # yum install epel-release
  # yum install pv
  ```

- For Red Hat Enterprise Linux:

  ```
  # yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-
  ↪$(rpm -E '%{rhel}').noarch.rpm
  # yum install pv
  ```

Then, to install MariaDB Galera Cluster, execute from the command line the following:

```
# yum install MariaDB-server MariaDB-client galera-4
```

Note: For MariaDB 10.3 and before, replace `galera-4` with `galera-3`.

Once you've done this, MariaDB Galera Cluster will be installed on your server. You'll need to repeat this process for each node in your cluster.

Note: If you installed MariaDB Galera Cluster over an existing stand-alone instance of MariaDB, there are some additional steps that you will need to take to update your system to the new database server. For more information on this, see ../training/tutorials/migration.

Related Documents

- *Install Galera* (page 36)
- *MariaDB Galera Source* (page 50)
- *Galera MySQL* (page 38)

Related Articles

- ../training/tutorials/migration

Other Resources

- Galera AWS (video)
- Galera MariaDB (video)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Install Galera* (page 36)
- *MariaDB Galera Binaries* (page 47)
- *Galera MySQL Source* (page 43)
- *wsrep_provider* (page 254)

Related Articles

- Migration
- System Configuration
- Replication Configuration
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

3.4 MariaDB Galera Cluster - Source Installation

MariaDB Galera Cluster is the MariaDB implementation of Galera Cluster for MySQL. Binary installation packages are available for Debian and RPM-based distributions of Linux. If your Linux distribution is based on a different package management system, or if it runs on a different Unix-like operating system where binary installation packages are not available, such as Solaris or FreeBSD (FreeBSD is available up to Galera Cluster for MySQL 5.7), you will need to build MariaDB Galera Cluster from source.

Note: If you built MariaDB Galera Cluster over an existing standalone instance of MariaDB, there are some additional steps that you need to take in order to update your system to the new database server. For more information, see Migration.

Preparing the Server

When building from source code, `make` cannot manage or install dependencies for either Galera Cluster or the build process itself. You need to install these packages first.

- For Debian-based distributions of Linux, if MariaDB is available in your repositories, you can run the following command:

```
# apt-get build-dep mariadb-server
```

- For RPM-based distributions, instead run this command:

```
# yum-builddep MariaDB-server
```

In the event that neither command works for your system or that you use a different Linux distribution or FreeBSD, the following packages are required:

- **MariaDB Database Server with wsrep API**: Git, CMake, GCC and GCC-C++, Automake, Autoconf, and Bison, as well as development releases of libaio and ncurses.
- **Galera Replication Plugin**: SCons, as well as development releases of Boost, Check and OpenSSL.

Check with the repositories for your distribution or system for the appropriate package names to use during installation. Bear in mind that different systems may use different names and that some may require additional packages to run. For instance, to run CMake on Fedora you need both `cmake` and `cmake-fedora`.

Building MariaDB Galera Cluster

The source code for MariaDB Galera Cluster is available through GitHub. Using Git you can download the source code to build MariaDB and the Galera Replicator Plugin locally on your system.

1. Clone the MariaDB database server repository.

```
# git clone https://github.com/mariadb/server
```

2. Checkout the branch for the version that you want to use.

```
# git checkout 10.0-galera
```

   The main branches available for MariaDB Galera Cluster are:

   - `10.1`
   - `10.0-galera`

   Starting with version 10.1, MariaDB includes the wsrep API for Galera Cluster by default.

You now have the source files for the MariaDB database server with the wsrep API needed to function as a Galera Cluster node.

In addition to the database server, you also need the wsrep Provider, also known as the Galera Replicator Plugin. In a separate directory run the following command:

```
# cd ..
# git clone https://github.com/codership/galera.git
```

Once Git finishes downloading the source files, you can start building the database server and the Galera Replicator Plugin. You now have the source files for the database server in a `server/` directory and the Galera source files in `galera/`.

Building the Database Server

The database server for Galera Cluster is the same as that of the standard database servers for standalone instances of MariaDB, with the addition of a patch for the wsrep API, which is packaged in the version downloaded from GitHub. You can enable the patch through the `WITH_WSREP` and `WITH_INNODB_DISALLOW_WRITES` CMake configuration options.

To build the database server, `cd` into the `server/` directory and run the following commands:

```
# cmake -DWITH_WSREP=ON -DWITH_INNODB_DISALLOW_WRITES=ON ./
# make
# make install
```

Note: In addition to compiling through `cmake` and `make`, there are also a number of build scripts in the `BUILD/` directory, which you may find more convenient to use. For example,

```
# ./BUILD/compile-pentium64-wsrep
```

This has the same effect as running the above commands with various build options pre-configured. There are several build scripts available in the directory, select the one that best suits your needs.

Building the wsrep Provider

The *Galera Replication Plugin* implements the *wsrep API* and operates as the wsrep Provider for the database server. What it provides is a certification layer to prepare write-sets and perform certification checks, a replication layer and a group communication framework.

To build the Galera Replication Plugin, `cd` into the `galera/` directory and run SCons.

```
# scons
```

This process creates the Galera Replication Pluigin, (that is, the `libgalera_smm.so` file). In your `my.cnf` configuration file, you need to define the path to this file for the *wsrep_provider* (page 254) parameter.

Note: For FreeBSD users, building the Galera Replication Plugin from source raises certain issues due to Linux dependencies. You can mitigate these by using the ports build available at `/usr/ports/databases/galera` or by installing the binary package:

```
# pkg install galera
```

Post-installation Configuration

After the build completes, there are some additional steps that you must take in order to finish installing the database server on your system. This is over and beyond the standard configuration process listed in System Configuration and Replication Configuration.

Note: Unless you defined the `CMAKE_INSTALL_PREFIX` configuration variable when you ran `cmake` above, by default the database is installed to the path `/usr/local/mysql/`. If you chose a custom path, adjust the commands below to accommodate the change.

---

1. Create the user and group for the database server.

   ```
   # groupadd mysql
   # useradd -g mysql mysql
   ```

2. Install the database.

   ```
   # cd /usr/local/mysql
   # ./scripts/mysql_install_db --user=mysql
   ```

   This installs the database in the working directory, (that is, at `/usr/local/mysql/data`). If you would like to install it elsewhere or run the script from a different directory, specify the desired paths with the `--basedir` and `--datadir` options.

3. Change the user and group permissions for the base directory.

   ```
   # chown -R mysql /usr/local/mysql
   # chgrp -R mysql /usr/local/mysql
   ```

4. Create a system unit for the database server.

   ```
   # cp /usr/local/mysql/supported-files/mysql.server \
        /etc/init.d/mysql
   # chmod +x /etc/init.d/mysql
   # chkconfig --add mysql
   ```

   This allows you to start Galera Cluster using the `service` command. It also sets the database server to start during boot.

In addition to this procedure, bear in mind that any further customization variables you enabled during the build process, such as a nonstandard base or data directory, may require you to define additional parameters in the configuration file, (that is, `my.cnf`).

Note: This tutorial omits MariaDB authentication options for brevity.

Related Documents

- *Install Galera* (page 36)
- *MariaDB Galera Binaries* (page 47)
- *Galera MySQL Source* (page 43)
- *wsrep_provider* (page 254)

Related Articles

- Migration

- System Configuration
- Replication Configuration

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Articles

- ../training/tutorials/migration
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

3.5 Percona XtraDB Cluster - Binary Installation

Percona XtraDB Cluster is the Percona implementation of Galera Cluster for MySQL. Binary installation packages are available for Debian- and RPM-based distributions through the Percona repository.

Enabling the Percona Repository

In order to install Percona XtraDB Cluster through your package manager, you need to first enable the Percona repository on your system. There are two different ways to accomplish this, depending upon which Linux distribution you use.

Enabling the `apt` Repository

For Debian and Debian-based Linux distributions, the procedure for adding the Percona repository requires that you first install Software Properties on your system. The package names vary depending upon which distribution you use. For Debian, in the terminal run the following command:

```
# apt-get install python-software-properties
```

For Ubuntu, instead run this command:

```
$ sudo apt-get install software-properties-common
```

In the event that you use a different Debian-based distribution and neither of these commands work, consult your distribution's package listings for the appropriate package name.

Once you have Software Properties installed, you can enable the Percona repository for your system.

1. Add the GnuPG key for the Percona repository:

```
# add-key adv --recv-keys --keyserver \
      keyserver.ubuntu.com 1C4CBDCDCD2EFD2A
```

2. Add the Percona repository to your sources list:

```
# add-apt-repository 'deb https://repo.percona.com/apt release main'
```

   For the repository address, make the following changes:

   - `release` Indicates the release name for the distribution you are using. For example, `wheezy`.

     In the event that you do not know which release you have installed on your server, you can find out using the following command:

```
$ lsb_release -a
```

3. Update the local cache.

```
# apt-get update
```

For more information on the repository, available packages and mirrors, see Percona Software Repositories Documentation.

Packages in the Percona repository are now available for installation on your server through `apt-get`.

Enabling the `yum` Repository

For RPM-based distributions, you can enable the Percona repository through `yum` using the following command:

```
# yum install https://repo.percona.com/yum/percona-release-latest.noarch.rpm
```

For more information on the repository, package names or available mirrors, see Percona Software Repositories Documentation.

Packages in the Percona repository are now available for installation on your server through `yum`.

Installing Percona XtraDB Cluster

Note: Telemetry is enabled by default. To disable it, see https://docs.percona.com/percona-xtradb-cluster/8.0/telemetry.html.

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.

Starting with Percona XtraDB Cluster 8.0.20-11, the `Linux - Generic` section lists only full or minimal tar files. Each tarball file replaces the multiple tar file listing used in earlier versions and supports all distributions.

The version number in the tarball name must be substituted with the appropriate version number for your system. To indicate that such a substitution is needed in statements, we use `<version-number>`.

For installations before Percona XtraDB Cluster 8.0.20-11, the `Linux - Generic` section contains multiple tarballs based on the operating system names:

```
Percona-XtraDB-Cluster_8.0.18-9.3_Linux.x86_64.bionic.tar.gz
Percona-XtraDB-Cluster_8.0.18-9.3_Linux.x86_64.buster.tar.gz
...
```

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

```
curl -O https://downloads.percona.com/downloads/Percona-XtraDB-Cluster-
↪LATEST/Percona-XtraDB-Cluster-8.0.27/binary/tarball/Percona-XtraDB-Cluster_
↪8.0.27-18.1_Linux.x86_64.glibc2.17-minimal.tar.gz
```

Check your system to make sure the packages that the PXC version requires are installed.

For Debian and Ubuntu-based distributions, run the following commands in the terminal:

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

For Ubuntu and distributions that derive from Ubuntu, however, you will need to specify the meta package. In the terminal, run this command instead:

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

For RPM-based distributions, instead run this command:

```
# sudo yum install -y openssl socat  \
      procps-ng chkconfig procps-ng coreutils shadow-utils \
```

Percona XtraDB Cluster is now installed on your server.

Note: If you installed Percona XtraDB Cluster over an existing standalone instance of Percona XtraDB, there are some additional steps that you need to take in order to update your system to the new database server. For more information, see ../training/tutorials/migration.

Note: Telemetry is enabled by default. To disable it, see Telemetry on Percona XtraDB Cluster.

Related Articles

- ../training/tutorials/migration

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses

- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *wsrep_provider* (page 254)

Related Articles

- ../training/tutorials/migration
- System Configuration
- Replication Configuration
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

3.6 Percona XtraDB Cluster - Source Installation

Percona XtraDB Cluster is the Percona implementation of Galera Cluster for MySQL. Binary installation packages are available for Debian- and RPM-based distributions of Linux. If your Linux distribution is based on a different package management system or if it runs on a different unix-like operating system where binary installation packages are unavailable, such as Solaris or FreeBSD, you will need to build Percona XtraDB Cluster from source.

Note: In the event that you built Percona XtraDB Cluster over an existing standalone instance of Percona XtraDB, there are some additional steps that you need to take in order to update your system to the new database server. For more information, see ../training/tutorials/migration.

Preparing the Server

When building from source code, `make` cannot manage or install dependencies necessary for either Galera Cluster itself or the build process. You need to install these packages first.

- For Debian-based distributions of Linux, if Percona is available in your repositories, you can run the following command:

```
# apt-get build-dep percona-xtradb-cluster
```

- For RPM-based distributions, instead run this command:

```
# yum-builddep percona-xtradb-cluster
```

In the event that neither command works for your system or that you use a different Linux distribution or FreeBSD (FreeBSD is available up to Galera Cluster for MySQL 5.7), the following packages are required:

- **Percona XtraDB Database Server with wsrep API**: Git, CMake, GCC and GCC-C++, Automake, Autoconf, and Bison, as well as development releases of libaio and ncurses.
- **Galera Replication Plugin**: SCons, as well as development releases of Boost, Check and OpenSSL.

Check with the repositories for your distribution or system for the appropriate package names to use during installation. Bear in mind that different systems may use different names and that some may require additional packages to run. For instance, to run CMake on Fedora you need both `cmake` and `cmake-fedora`.

Building Percona XtraDB Cluster

The source code for Percona XtraDB Cluster is available through GitHub. Using Git you can download the source to build both Percona XtraDB Cluster and the Galera Replication Plugin locally on your system.

1. Clone the Percona XtraDB Cluster database server.

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

2. Checkout the 8.0 branch and initialize submodules:

```
# cd percona-xtradb-cluster
# git checkout 8.0
# git submodule update --init --recursive
```

You now have the source files for the Percona XtraDB Cluster database server, set to the branch of development that you want to build.

Download the matching Percona XtraBackup 8.0 tarball (*.tar.gz) for your operating system from Percona Downloads. The following example extracts the Percona XtraBackup 8.0.32-25 tar.gz file to the target directory `./pxc-build`:

```
```{.bash data-prompt="$"}
# tar -xvf percona-xtrabackup-8.0.32-25-Linux-x86_64.glibc2.17.tar.gz -C ./
↪pxc-build
```
```

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.

In addition to the database server, you also need the wsrep Provider, also known as the Galera Replication Plugin. In a separate directory, run the following command:

```
# cd ..
# git clone https://github.com/codership/galera.git
```

Once Git finishes downloading the source file,s you can start building the database server and the Galera Replication Plugin. You now have the source file for the database server in a `percona-xtradb-cluster/` and the Galera source files in `galera/`.

Building the Database Server

The database server for Galera Cluster is the same as that of the standard database servers for standalone instances of Percona XtraDB, with the addition of a patch for the wsrep API, which is packaged in the version downloaded from GitHub. You can enable the patch through the wsrep API, requires that you enable it through the `WITH_WSREP` and `WITH_INNODB_DISALLOW_WRITES` CMake configuration options.

To build the database server, `cd` into the `percona-xtradb-cluster` directory and run the following commands:

```
# cmake -DWITH_WSREP=ON -DWITH_INNDOB_DISALLOW_WRITES=ON ./
# make
# make install
```

Note: In addition to compiling through `cmake` and `make`, there are also a number of build scripts available in the `BUILD/` directory, which you may find more convenient to use. For example:

```
# ./BUILD/compile-pentium64
```

This has the same effect as running the above commands with various build options pre-configured. There are several build scripts available in the `BUILD/` directory. Select the one that best suits your nees.

Building the wsrep Provider

The *Galera Replication Plugin* implements the *wsrep API* and operates as the wsrep Provider for the database server. What it provides is a certification layer to prepare write-sets and perform certification checks, a replication layer and a group communication framework.

To build the Galera Replication Plugin, `cd` into the `galera/` directory and run SCons.

```
# scons
```

This process creates the Galera Replication Plugin, (that is, the `libgalera_smm.so` file). In your `my.cnf` configuration file, you need to define the path to this file for the *wsrep_provider* (page 254) parameter.

Note: For FreeBSD users, building the Galera Replication Plugin from sources raises certain Linux compatibility issues. You can mitigate these by using the ports build available at `/usr/ports/databases/galera` or by install the binary package:

```
# pkg install galera
```

Post-installation Configuration

After the build completes, there are some additional steps that you must take in order to finish installing the database server on your system. This is over and beyond the standard configuration process listed in System Configuration and Replication Configuration.

Note: Unless you defined the `CMAKE_INSTALL_PREFIX` configuration varaible when you ran `cmake` above, by default the database is installed to the path `/usr/local/mysql/`. If you chose a custom path, adjust the commands below to accommodate this change.

---

1. Create the user and group for the database server.

```
# groupadd mysql
# useradd -g mysql mysql
```

2. Install the database.

```
# cd /usr/local/mysql
# ./scripts/mysql_install_db --user=mysql
```

   This installs the database in the working directory, (that is, at `/usr/local/mysql/data`). If you would like to install it elsewhere or run the script from a different directory, specify the desired paths with the `--basedir` and `--datadir` options.

3. Change the user and group permissions for the base directory.

```
# chown -R mysql /usr/local/mysql
# chgrp -R mysql /usr/local/mysql
```

4. Create a system unit for the database server.

```
# cp /usr/local/mysql/supported-files/mysql.server \
     /etc/init.d/mysql
# chmod +x /etc/init.d/mysql
# chkconfig --add mysql
```

   This allows you to start Galera Cluster using the `service` command. It also sets the database server to start during boot.

In addition to this procedure, bear in mind that any further customization variables that you enabled during the build process through `cmake`, (such as nonstandard base or data directories), may require you to define addition parameters in the configuration file, (that is, the `my.cnf`).

Related Documents

- *wsrep_provider* (page 254)

Related Articles

- ../training/tutorials/migration
- System Configuration
- Replication Configuration

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Arbitrator* (page 105)
- *Auto-Eviction* (page 100)
- *Backups* (page 109)
- *Cluster Monitoring* (page 138)
- *Deployment* (page 111)
- *Flow Control* (page 96)
- *Galera System Tables* (page 77)
- *Node Provisioning* (page 65)
- *Recover Primary* (page 91)
- *Reset Quorum* (page 93)
- *Schema Upgrades* (page 83)
- *Scriptable SST* (page 74)
- *Security* (page 207)
- *State Snapshot Transfers* (page 67)
- *Streaming Replication* (page 103)
- *Upgrading Galera* (page 86)

Related Articles

- ../training/tutorials/migrate
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

CHAPTER

FOUR

GALERA CLUSTER ADMINISTRATION

With the basics of how the cluster works and how to install and initialize it covered, this part begins a five part series on the administration and management of Galera Cluster.

The sections in this part relate to the administration of nodes and the cluster. *Deployment* (page 111), covers how to use Galera Cluster in relation to your wider infrastructure, how to configure load balancers to work with the cluster and edge case deployments, such as running nodes in containers. The pages in *Cluster Monitoring* (page 138) show how to keep tabs on the status of the cluster and automate reporting. *Security* (page 207) covers configuring Galera Cluster to work with firewalls, SELinux and SSL encryption. ../training/tutorials/migrate how to transition from a standalone instance of MySQL, MariaDB or Percona XtraDB to Galera Cluster.

Node Administration

Managing and administering nodes in Galera Cluster is similar to the administration and management of the standard standalone MySQL, MariaDB and Percona XtraDB database servers, with some additional features used to manage its interaction with the cluster. These pages cover the administration of individual nodes, how they handle write-set replication and schema updates, and the procedure for upgrading Galera Cluster software.

- *Node Provisioning* (page 65)

  The complete process of replicating data into a node so that it can operate as part of the Primary Component is called 'provisioning' the node. It ensures that the nodes update the local data, keeping it consistent with the state of the cluster. This section provides an overview to how nodes join the cluster and maintain their state through state transfers.

- *State Snapshot Transfers* (page 67)

  When a node falls too far behind the cluster, they request State Snapshot Transfers from another node in order to bring its local database up to date with the cluster. This section provides a guide to each state transfer method Galera Cluster supports.

- *Scriptable State Snapshot Transfers* (page 74)

  When nodes send and receive State Snapshot Transfers, they manage the process through external scripts that call the standard state transfer methods. If you require additional functionality than what is available by default, you can create a script to implement your own custom state snapshot transfer methods.

- *Galera System Tables* (page 77)

  When you install Galera Cluster, it creates a set of system tables in the `mysql` database, which it uses to store configuration information. Similar to how the underlying database server uses the `performance_schema` and `information_schema`, Galera Cluster uses these tables to record information relevant to replication. This section provides a guide to what you will find in these tables and how you might query them for useful information about the status of the node and the cluster.

- *Schema Upgrades* (page 83)

  Statements that update the database schema, (that is, DDL statements), are non-transactional and as such won't replicate to the cluster through write-sets. This section covers different methods for online schema upgrades and how to implement them in your deployment.

- *Upgrading Galera Cluster* (page 86)

  In order to upgrade Galera Cluster to a new version or increment of the software, there are a few additional steps you need to take in order to maintain the cluster during the upgrade. This section provides guides to different methods in handling this process.

Cluster Administration

In addition to node administration, Galera Cluster also provides interfaces for managing and administering the cluster. These sections cover Primary Component recovery, managing Flow Control and Auto Eviction, as well as Galera Arbitrator and how to handle backups.

- *Recovering Primary Component* (page 91)

  When nodes establish connections with each other, they form components. The operational component in the cluster is called the Primary Component. This section covers a new feature in version 3.6 of Galera Cluster, which sets the nodes to save the Primary Component state to disk. In the event of an outage, once all the nodes that previously formed the Primary Component reestablish network connectivity, they automatically restore themselves as the new Primary Component.

- *Resetting the Quorum* (page 93)

  The Primary Component maintains *Quorum* when most of the nodes in the cluster are connected to it. This section provides a guide to resetting the quroum in the event that the cluster becomes non-operational due to a major network outage, the failure of more than half the nodes, or a split-brain situation.

- *Managing Flow Control* (page 96)

  When nodes fall too far behind, Galera Cluster uses a feedback mechanism called Flow Control, pausing replication to give the node to process transactions and catch up with the cluster. This section covers the monitoring and configuration of Flow Control, in order to improve node performance.

- *Auto-Eviction* (page 100)

  When Galera Cluster notices erratic behavior from a node, such as in the case of unusually delayed response times, it can initiate a process to remove the node permanently from the cluster. This section covers the configuration and management of how the cluster handles these Auto Evictions.

- *Using Streaming Replication* (page 103)

  When the node uses Streaming Replication, instead of waiting for the commit to replicate and apply transactions to the cluster, it breaks the transaction down into replication units, transferring and applying these on the replica nodes while the transaction is still open. This section provides a guide to how to enable, configure and utilize Streaming Replication.

- *Galera Arbitrator* (page 105)

  Galera Arbitrator is a separate application from Galera Cluster. It functions as an additional node in quorum calculations, receives the same data as other node, but does not participate in replications. You can use it to provide an odd node to help avoid split-brain situations, or use it in generating consistent application state snapshots, in generating backups.

- *Backing Up Cluster Data* (page 109)

Standard backup methods available to MySQL database servers fail to preserve Global Transaction ID's used by Galera Cluster. You can recover data from these backups, but they're insufficient in restoring nodes to a well-defined state. This section shows how to use state transfers to properly perform backups in Galera Cluster.

Related Documents

- *Arbitrator* (page 105)
- *Auto-Eviction* (page 100)
- *Backups* (page 109)
- *Cluster Monitoring* (page 138)
- *Deployment* (page 111)
- *Flow Control* (page 96)
- *Galera System Tables* (page 77)
- *Node Provisioning* (page 65)
- *Recover Primary* (page 91)
- *Reset Quorum* (page 93)
- *Schema Upgrades* (page 83)
- *Scriptable SST* (page 74)
- *Security* (page 207)
- *State Snapshot Transfers* (page 67)
- *Streaming Replication* (page 103)
- *Upgrading Galera* (page 86)

Related Articles

- ../training/tutorials/migrate

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *wsrep_sst_donor* (page 260)
- *wsrep_node_name* (page 250)
- Home
- *Docs* (page 1)
- KB

- Training
- FAQ

4.1 Node Provisioning

When the state of a new or failed node differs from that of the cluster's *Primary Component*, the new or failed node must be synchronized with the cluster. Because of this, the provisioning of new nodes and the recover of failed nodes are essentially the same process as that of joining a node to the cluster Primary Component.

Galera reads the initial node state ID from the **grastate.dat** file, found in the directory assigned by the `wsrep_data_dir` parameter. Each time the node gracefully shuts down, Galera saves to this file.

In the event that the node crashes while in *Total Order Isolation* mode, its database state is unknown and its initial node state remains undefined:

```
00000000-0000-0000-0000-000000000000:-1
```

Note: In normal transaction processing, only the seqno part of the GTID remains undefined, (that is, with a value of `-1`. The UUID, (that is, the remainder of the node state), remains valid. In such cases, you can recover the node through an *Incremental State Transfer*.

How Nodes Join the Cluster

When a node joins the cluster, it compares its own *state UUID* to that of the *Primary Component*. If the state UUID does not match, the joining node requests a state transfer from the cluster.

There are two options available to determining the state transfer donor:

- **Automatic** When the node attempts to join the cluster, the group communication layer determines the state donor it should use from those members available in the Primary Component.
- **Manual** When the node attempts to join the cluster, it uses the *wsrep_sst_donor* (page 260) parameter to determine which state donor it should use. If it finds that the state donor it is looking for is not part of the Primary Component, the state transfer fails and the joining node aborts. For *wsrep_sst_donor* (page 260), use the same name as you use on the *Donor Node* for the *wsrep_node_name* (page 250) parameter.

Note: A state transfer is a heavy operation. This is true not only for the joining node, but also for the donor. In fact, a state donor may not be able to serve client requests.

Thus, whenever possible: manually select the state donor, based on network proximity and configure the load balancer to transfer client connections to other nodes in the cluster for the duration of the state transfer.

When a state transfer is in process, the joining node caches write-sets that it receives from other nodes in a replica queue. Once the state transfer is complete, it applies the write-sets from the replica queue to catch up with the current Primary Component state. Since the state snapshot carries a state UUID, it is easy to determine which write-sets the snapshot contains and which it should discard.

During the catch-up phase, flow control ensures that the replica queue shortens, (that is, it limits the *Cluster Replication* rates to the write-set application rate on the node that is catching up).

While there is no guarantee on how soon a node will catch up, when it does the node status updates to `SYNCED` and it begins to accept client connections.

State Transfers

There are two types of state transfers available to bring the node up to date with the cluster:

- *State Snapshot Transfer* (SST) Where donor transfers to the joining node a snapshot of the entire node state as it stands.
- *Incremental State Transfer* (IST) Where the donor only transfers the results of transactions missing from the joining node.

When using automatic donor selection, starting in Galera Cluster version 3.6, the cluster decides which state transfer method to use based on availability.

- If there are no nodes available that can safely perform an incremental state transfer, the cluster defaults to a state snapshot transfer.
- If there are nodes available that can safely perform an incremental state transfer, the cluster prefers a local node over remote nodes to serve as the donor.
- If there are no local nodes available that can safely perform an incremental state transfer, the cluster chooses a remote node to serve as the donor.
- Where there are several local or remote nodes available that can safely perform an incremental state transfer, the cluster chooses the node with the highest seqno to serve as the donor.

Related Documents

- *wsrep_sst_donor* (page 260)
- *wsrep_node_name* (page 250)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *wsrep_sst_donor* (page 260)
- *wsrep_sst_method* (page 262)
- *gmcast.segment* (page 294)
- *mysqldump* (page 70)
- *rsync* (page 72)
- *clone* (page 73)
- *xtrabackup* (page 72)
- *Logical* (page 68)
- *Physical* (page 71)

- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

4.2 State Snapshot Transfers

When a new node joins a cluster, it will request data from the cluster. One node, known as a donor, will use a *State Snapshot Transfer* (SST) method to provide a full data copy to the new node, known as the joiner.

You can designate in advance which node should be the donor with the *wsrep_sst_donor* (page 260) parameter. If you do not set the *Donor Node*, the *Group Communication* module will select a donor based on the information available about the node states.

Group Communication monitors node states for the purposes of flow control, state transfers and *Quorum* calculations. It ensures that a node that shows as `JOINING` does not count towards flow control and quorum.

A node can serve as a donor when it is in the `SYNCED` state. The *Joiner Node* selects a donor from the available synced nodes. It shows preference to synced nodes that have the same *gmcast.segment* (page 294) wsrep Provider option, or it selects the first in the index. When a donor node is chosen, its state changes immediately to `DONOR`. It is no longer available for requests.

SST Methods

Galera supports several back-end methods for use in state snapshot transfers. There are two types: Logical State Snapshots, which interface through the database server and client; and Physical State Snapshots, which directly copy the data files from node to node.

Method	Speed	Blocks Donor	Live Node Availability	Type	DB Root Access
mysqldump (page 70)	Slow	Blocks	Available	*Logical* (page 68)	Donor and Joiner
rsync (page 72)	Faster	Blocks	Unavailable	*Physical* (page 71)	None
clone (page 73)	Fastest	On DDLs	Unavailable	*Physical* (page 71)	Only Donor
xtrabackup (page 72)	Fast	Briefly	Unavailable	*Physical* (page 71)	Only Donor
xtrabackup-2 (page 72)	Fast	Briefly	Unavailable	*Physical* (page 71)	Only Donor

To set the State Snapshot Transfer method, use the *wsrep_sst_method* (page 262) parameter. In the example below, the method is set to use `rsync`, along with the default donors:

```
wsrep_sst_method = rsync
wsrep_sst_donor  = "node1, node2"
```

There is no single best method for State Snapshot Transfers. You must decide which suits your particular needs and cluster deployment. Fortunately, you need only set the method on the receiving node. So long as the donor has support, it serves the transfer in whatever method the joiner requests.

Related Documents

- *wsrep_sst_donor* (page 260)
- *wsrep_sst_method* (page 262)
- *gmcast.segment* (page 294)
- *mysqldump* (page 70)
- *rsync* (page 72)
- *clone* (page 73)
- *xtrabackup* (page 72)
- *Logical* (page 68)
- *Physical* (page 71)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *wsrep_sst_method* (page 262)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

4.2.1 Logical State Snapshot

There is one back-end method available for a Logical State Snapshots: `mysqldump`.

The *Logical State Transfer Method* has the following advantages:

- These transfers are available on live servers. In fact, only a fully initialized server can receive a Logical State Snapshot.
- These transfers do not require the receptor node to have the same configuration as the *Donor Node*. This allows you to upgrade storage engine options.

  For example, when using this transfer method you can migrate from the Antelope to the Barracuda file format, use compression resize, or move iblog* files from one partition into another.

The Logical State Transfer Method has the following disadvantages:

- These transfers are as slow as `mysqldump`.
- These transfers require that you configure the receiving database server to accept root connections from potential donor nodes.
- The receiving server must have a non-corrupted database.

mysqldump

The main advantage of `mysqldump` is that you can transfer a state snapshot to a working server. That is, you start the server standalone and then instruct it to join a cluster from within the database client command line. You can also use it to migrate from an older database format to a newer one.

`mysqldump` requires that the receiving node have a fully functional database, which can be empty. It also requires the same root credentials as the donor and root access from the other nodes.

This transfer method is several times slower than the others on sizable databases, but it may prove faster in cases of very small databases. For instance, on a database that is smaller than the log files.

Warning: This transfer method is sensitive to the version of `mysqldump` each node uses. It is not uncommon for a given cluster to have installed several versions. A State Snapshot Transfer can fail if the version one node uses is older and incompatible with the newer server.

On occasion, `mysqldump` is the only option available. For instance, if you upgrade from a cluster using MySQL 5.1 with the built-in InnoDB support to MySQL 5.5, which uses the InnoDB plugin.

The `mysqldump` script only runs on the sending node. The output from the script gets piped to the MySQL client that connects to the *Joiner Node*.

Because `mysqldump` interfaces through the database client, configuring it requires several steps beyond setting the *wsrep_sst_method* (page 262) parameter. For more information on its configuration, see:

For more information on `mysqldump`, see mysqldump Documentation.

Related Documents

- *wsrep_sst_method* (page 262)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *wsrep_sst_method* (page 262)
- *wsrep_sst_auth* (page 260)
- Home

- *Docs* (page 1)
- KB
- Training
- FAQ

Enabling `mysqldump`

The *Logical State Transfer Method*, `mysqldump` works by interfacing through the database server rather than the physical data. As such, it requires some additional configuration, besides setting the *wsrep_sst_method* (page 262) parameter.

Configuring SST Privileges

In order for `mysqldump` to interface with the database server, it requires root connections for both the donor and joiner nodes. You can enable this through the *wsrep_sst_auth* (page 260) parameter.

Using a text editor, open the `wsrep.cnf` file–it should be in the `/etc/mysql/conf.d/` directory. Add a line like the following to that file:

```
# wsrep SST Authentication
wsrep_sst_auth = wsrep_sst_username:password
```

You would use your own authentication parameters in place of `wsrep_sst_user` and `password`. This line will provide authentication information that the node will need to establish connections. Use the same values for every node in the cluster.

Granting SST Privileges

When the database server starts, it will read from the `wsrep.cnf` file to get the authentication information it needs to access another database server. In order for the node to accept connections from the cluster, you must also create and configure the State Snapshot Transfer user through the database client.

In order to do this, you need to start the database server. If you haven't used this node on the cluster before, start it with replication disabled. For servers that use `init`, execute the following from the command-line:

```
# service mysql start --wsrep-on=off
```

For servers that use `systemd`, instead execute this from the command-line:

```
# systemctl start mysql --wsrep-on=OFF
```

When the database server is running, log into the database using a client and execute the `GRANT ALL` statement for the IP address of each node in the cluster. You would do this like so:

```
GRANT ALL ON *.* TO 'wsrep_sst_user'@'node1_IP_address'
     IDENTIFIED BY 'password';
GRANT ALL ON *.* TO 'wsrep_sst_user'@'node2_IP_address'
     IDENTIFIED BY 'password';
GRANT ALL ON *.* TO 'wsrep_sst_user'@'node3_IP_address'
     IDENTIFIED BY 'password';
```

You would, of course, modify the text above to use your user names, IP addresses, and passwords. These SQL statements will grant each node in the cluster access to the database server on this node. You need to run these SQL statements on each node to allow `mysqldump` in state transfers among them.

If you have not yet created the cluster, you can stop the database server while you configure the other nodes. To stop MySQL on servers that use `init`, run the execute the following from the command-line:

```
service mysql stop
```

For servers that use `systemd`, you would execute the following from the command-line to shutdown MySQL:

```
systemctl stop mysql
```

Related Documents

- *wsrep_sst_method* (page 262)
- *wsrep_sst_auth* (page 260)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

4.2.2 Physical State Snapshot

There are two back-end methods available for Physical State Snapshots: `rsync` and `xtrabackup`. Starting with version 8.0.22 also `clone` method is available for Galera Cluster for MySQL

The *Physical State Transfer Method* has the following advantages:

- These transfers physically copy the data from one node to the disk of the other, and as such do not need to interact with the database server at either end.
- These transfers do not require the database to be in working condition, as the *Donor Node* overwrites what was previously on the joining node disk.
- These transfers are faster.

The *Physical State Transfer Method* has the following disadvantages:

- These transfers require the joining node to have the same data directory layout and the same storage engine configuration as the donor node. For example, you must use the same file-per-table, compression, log file size and similar settings for InnoDB.
- These transfers are not accepted by servers with initialized storage engines.

  What this means is that when your node requires a state snapshot transfer, the database server must restart to apply the changes. The database server remains inaccessible to the client until the state snapshot transfer is complete, since it cannot perform authentication without the storage engines.

rsync

The fastest back-end method for State Snapshot Transfers is `rsync`. It carries all the advantages and disadvantages of of the Physical Snapshot Transfer. While it does block the donor node during transfer, `rsync` does not require database configuration or root access, which makes it easier to configure.

When using terabyte-scale databases, `rsync` is considerably faster, (1.5 to 2 times faster), than `xtrabackup`. This translates to a reduction in transfer times by several hours.

`rsync` also features the rsync-wan modification, which engages the `rsync` delta transfer algorithm. However, given that this makes it more I/O intensive, you should only use it when the network throughput is the bottleneck, which is usually the case in WAN deployments.

Note: The most common issue encountered with this method is due to incompatibilities between the various versions of `rsync` on the donor and joining nodes.

The `rsync` script runs on both donor and joining nodes. On the joiner, it starts `rsync` in server-mode and waits for a connection from the donor. On the donor, it starts `rsync` in client-mode and sends the contents of the data directory to the joining node.

```
wsrep_sst_method = rsync
```

For more information about `rsync`, see the rsync Documentation.

xtrabackup

The most popular back-end method for State Snapshot Transfers is `xtrabackup`. It carries all the advantages and disadvantages of a Physical State Snapshot, but is virtually non-blocking on the donor node.

`xtrabackup` only blocks the donor for the short period of time it takes to copy the MyISAM tables, (for instance, the system tables). If these tables are small, the blocking time remains very short. However, this comes at the cost of speed: a state snapshot transfer that uses `xtrabackup` can be considerably slower than one that uses `rsync`.

Given that `xtrabackup` copies a large amount of data in the shortest possible time, it may also noticeably degrade donor performance.

Note: The most common issue encountered with this method is due to its configuration. `xtrabackup` requires that you set certain options in the configuration file, which means having local root access to the donor server.

```
[mysqld]
wsrep_sst_auth = <SST user>:<SST password>
wsrep_sst_method = xtrabackup-v2
datadir = /path/to/datadir
```

Minimal setup for the `xtrabackup` SST user:

```
mysql> CREATE USER '<SST user>'@'localhost' IDENTIFIED BY '<SST password>';
mysql> GRANT BACKUP_ADMIN, PROCESS, RELOAD ON *.* TO '<SST user>'@'localhost';
mysql> GRANT SELECT ON performance_schema.keyring_component_status TO '<SST user>'@
↪'localhost' ;
mysql> GRANT SELECT ON performance_schema.log_status TO '<SST user>'@'localhost' ;
```

For more information on `xtrabackup`, see the Percona XtraBackup User Manual and XtraBackup SST Configuration.

clone

Starting with version 8.0.22 `clone` SST method is available for Galera Cluster for MySQL. It is based on the native MySQL clone plugin. As of 8.4.2, it is the default value for SST methods. It proved to be much faster than `xtrabackup`, however it will block Donor node on DDL execution if that happens during the transfer.

As of MySQL-wsrep 8.0.27-26.9, progress reporting is also available for the `clone` SST method. See also *Scriptable State Snapshot Transfers* (page 74).

Basic configuration for `clone` SST on Joiner:

```
[mysqld]
wsrep_sst_method=clone
```

Basic server configuration for `clone` SST on Donor:

```
[mysqld]
wsrep_sst_auth=<SST user>:<SST password>
```

Minimal setup for the `clone` SST user:

```
mysql> CREATE USER '<SST user>'@'localhost' IDENTIFIED BY '<SST password>';
mysql> GRANT CREATE USER, SUPER ON *.* TO '<SST user>'@'localhost';
mysql> GRANT INSERT, DELETE ON mysql.plugin TO '<SST user>'@'localhost';
mysql> GRANT UPDATE ON performance_schema.setup_instruments TO '<SST user>'@'localhost
↪';
mysql> GRANT UPDATE ON performance_schema.setup_consumers TO '<SST user>'@'localhost';
mysql> GRANT BACKUP_ADMIN ON *.* TO '<SST user>'@'localhost' WITH GRANT OPTION;
mysql> GRANT EXECUTE ON *.* TO '<SST user>'@'localhost' WITH GRANT OPTION;
mysql> GRANT SELECT ON performance_schema.* TO '<SST user>'@'localhost' WITH GRANT␣
↪OPTION;
```

Optionally *plugin_dir* variable needs to be configured if MySQL plugins are not in the default location.

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ

- search

Related Documents

- *wsrep_sst_receive_address* (page 263)
- *wsrep_sst_auth* (page 260)
- *wsrep_sst_method* (page 262)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

4.3 Scriptable State Snapshot Transfers

When a node sends and receives a *State Snapshot Transfer*, it manage it through processes that run external to the database server. If you need more from these processes than the default behavior provides, Galera Cluster provides an interface for custom shell scripts to manage state snapshot transfers on the node.

Using the Common SST Script

Galera Cluster includes a common script for managing a *State Snapshot Transfer*, which you can use as a starting point in building your own custom script. The filename is `wsrep_sst_common`. For Linux users, the package manager typically installs it for you in `/usr/bin`.

The common SST script provides ready functions for parsing argument lists, logging errors, and so on. There are no constraints on the order or number of parameters it takes. You can add new parameters and ignore any of the existing ones as you prefer.

It assumes that the storage engine initialization on the receiving node takes place only after the state transfer is complete. Meaning that it copies the contents of the source data directory to the destination data directory (with possible variations).

State Transfer Script Parameters

When Galera Cluster starts an external process for state snapshot transfers, it passes a number of parameters to the script, which you can use in configuring your own state transfer script.

General Parameters

These parameters are passed to all state transfer scripts, regardless of method or whether the node is sending or receiving:

`--role` The script is given a string, either `donor` or `joiner`, to indicate whether the node is using it to send or receive a state snapshot transfer.

`--address` The script is given the IP address of the *Joiner Node*.

When the script is run by the joiner, the node uses the value of either the *wsrep_sst_receive_address* (page 263) parameter or a sensible default formatted as `<ip_address>:<port>`. When the script is run by the donor, the node uses the value from the state transfer request.

`--auth` The script is given the node authentication information.

When the script is run by the joiner, the node uses the value given to the *wsrep_sst_auth* (page 260) parameter. When the script is run by the donor, it uses the value given by the state transfer request.

`--datadir` The script is given the path to the data directory. The value is drawn from the `mysql_real_data_home` parameter.

`--defaults-file` The script is given the path to the `my.cnf` configuration file.

The values the node passes to these parameters varies depending on whether the node calls the script to send or receive a state snapshot transfer. For more information, see *Calling Conventions* (page 75) below.

Donor-specific Parameters

These parameters are passed only to state transfer scripts initiated by a node serving as the *Donor Node*, regardless of the method being used:

`--gtid` The node gives the *Global Transaction ID*, which it forms from the state UUID and the sequence number, or seqno, of the last committed transaction.

`--socket` The node gives the local server socket for communications, if required.

`--bypass` The node specifies whether the script should skip the actual data transfer and only pass the Global Transaction ID to the receiving node. That is, whether the node should initiate an *Incremental State Transfer*.

Logical State Transfer-specific Parameters

These parameters are passed only to the `wsrep_sst_mysqldump` state transfer script by both the sending and receiving nodes:

`--user` The node gives to the script the database user, which the script then uses to connect to both donor and joiner database servers. Meaning, this user must be the same on both servers, as defined by the *wsrep_sst_auth* (page 260) parameter.

`--password` The node gives to the script the password for the database user, as configured by the *wsrep_sst_auth* (page 260) parameter.

`--host` The node gives to the script the IP address of the joiner node.

`--port` The node gives to the script the port number to use with the joiner node.

`--local-port` The node gives to the script the port number to use in sending the state transfer.

Calling Conventions

In writing your own custom script for state snapshot transfers, there are certain conventions that you need to follow in order to accommodate how Galera Cluster calls the script.

Receiver

When the node calls for a state snapshot transfer as a joiner, it begins by passing a number of arguments to the state transfer script, as defined in *General Parameters* (page 74) above. For your own script you can choose to use or ignore these arguments as suits your needs.

After the script receives these arguments, prepare the node to accept a state snapshot transfer. For example, in the case of `wsrep_sst_rsync`, the script starts `rsync` in server mode.

To signal that the node is ready to receive the state transfer, print the following string to standard output: `ready <address>:port\n`. Use the IP address and port at which the node is waiting for the state snapshot. For example:

```
ready 192.168.1.1:4444
```

The node responds by sending a state transfer request to the donor node. The node forms the request with the address and port number of the joiner node, the values given to *wsrep_sst_auth* (page 260), and the name of your script. The donor receives the request and uses these values as input parameters in running your script on that node to send back the state transfer.

When the joiner node receives the state transfer and finishes applying it, print to standard output the *Global Transaction ID* of the received state. For example:

```
e2c9a15e-5485-11e0-0800-6bbb637e7211:8823450456
```

Then exit the script with a `0` status, to indicate that the state transfer was successful.

Sender

When the node calls for a state snapshot transfer as a donor, it begins by passing a number of arguments to the state transfer script, as defined in *General Parameters* (page 74) above. For your own script, you can choose to use or ignore these arguments as suits your needs.

While your script runs, Galera Cluster accepts the following signals. You can trigger them by printing to standard output:

`flush tables\n` Optional signal that asks the database server to run `FLUSH TABLES`. When complete, the database server creates a `tables_flushed` file in the data directory.

`continue\n` Optional signal that tells the database server that it can continue to commit transactions.

Progress reporting is also enabled for the `clone` SST method.

`done\n` Mandatory signal that tells the database server that the state transfer is complete and successful.

After your script sends the `done\n` signal, exit with a `0` return code.

In the event of failure, Galera Cluster expects your script to return a code that corresponds to the error it encountered. The donor node returns this code to the joiner through group communication. Given that its data directory now holds an inconsistent state, the joiner node then leaves the cluster and aborts the state transfer.

Note: Without the `continue\n` signal, your script runs in Total Order Isolation, which guarantees that no further commits occur until the script exits.

The script outputs control messages to the standard output, from where they are read by the parent `mysqld` process. These are:

- `total\n` This progress reporting parameter indicates the new SST stage and repor-ts the estimated total work.
- `complete\n` This progress reporting parameter reports the work completed so far.

Enabling Scriptable SST's

Whether you use `wsrep_sst_common` directly or decide to write a script of your own from scratch, the process for enabling it remains the same. The filename must follow the convention of `wsrep_sst_<name>`, with `<name>` being the value that you give for the *wsrep_sst_method* (page 262) parameter in the configuration file.

For example, if you write a script with the filename `wsrep_sst_galera-sst`, you would add the following line to your `my.cnf`:

```
wsrep_sst_method = galera-sst
```

When the node starts, it uses your custom script for state snapshot transfers.

Related Documents

- *wsrep_sst_receive_address* (page 263)
- *wsrep_sst_auth* (page 260)
- *wsrep_sst_method* (page 262)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

4.4 Galera System Tables

Starting with version 4 of Galera, three system tables related to Galera replication were added to the `mysql` database: `wsrep_cluster`, `wsrep_cluster_members`, and `wsrep_streaming_log`. As of MariaDB Server 10.10, and MySQL-wsrep 8.4.2, there is yet another, `wsrep_allowlist`. These system tables may be used by database administrators to get a sense of the current layout of the nodes in a cluster.

To see these tables on your server, execute the following SQL statement one of them using the `mysql` client or a similar client:

```
SHOW TABLES FROM mysql LIKE 'wsrep%';

+----------------------------+
| Tables_in_mysql (wsrep%)   |
+----------------------------+
| wsrep_allowlist            |
| wsrep_cluster              |
| wsrep_cluster_members      |
| wsrep_streaming_log        |
+----------------------------+
```

Database administrators and clients with the access to the `mysql` database may read these tables, but they may not modify them: the database itself will make modifications, as needed. If your server does not have these tables, it may be that your server is using an older version of Galera.

Allowlist

The `wsrep_allowlist` table stores the allowed IP addresses that can perform an IST/SST, in a comma delimited format. Before the introduction of "wsrep_allowlist", as long as a node has access to Galera Cluster TCP ports, it can make an SST/IST request, without authentication being performed; some users prefer to have a method to make this more robust, and secure, hence with `wsrep_allowlist` only if the JOINER node is in the IP list, will it be allowed to join the cluster.

You can either have IPv4 or IPv6 addresses for `wsrep_allowlist`, but it does not allow wildcard IPs or hostnames.

```
MariaDB [mysql]> describe wsrep_allowlist\G
*************************** 1. row ***************************
Field: ip
Type: char(64)
Null: NO
Key: PRI
Default: NULL
Extra:
1 row in set (0.001 sec)
```

To alter the allowlist, execute command:

```
insert into mysql.wsrep_allowlist(ip) values('18.193.102.155');
```

and the result will look like:

```
MariaDB [mysql]> select * from wsrep_allowlist;
+----------------+
| ip             |
+----------------+
| 18.193.102.155 |
| 18.194.147.243 |
+----------------+
2 rows in set (0.000 sec)
```

When another node tries to get connected, the potential DONOR nodes will see this in the `error.log`:

```
2024-03-18  8:19:02 0 [Warning] WSREP: Connection not allowed, IP 3.70.155.51 not␣
↪found in allowlist.
```

On the node trying to be the JOINER not in the allowlist, an error such as the one below should be easily notable:

```
2024-03-18  8:19:14 0 [ERROR] WSREP: failed to open gcomm backend connection: 110:␣
↪failed to reach primary view: 110 (Connection timed out) at ./gcomm/src/pc.
↪cpp:connect():160
2024-03-18  8:19:14 0 [ERROR] WSREP: ./gcs/src/gcs_core.cpp:gcs_core_open():221:␣
↪Failed to open backend connection: -110 (Connection timed out)
2024-03-18  8:19:15 0 [ERROR] WSREP: ./gcs/src/gcs.cpp:gcs_open():1674: Failed to␣
↪open channel 'mariadb' at 'gcomm://18.194.147.243,18.193.102.155': -110 (Connection␣
↪timed out)
2024-03-18  8:19:15 0 [ERROR] WSREP: gcs connect failed: Connection timed out
```

Add the remaining node to the allowlist to fix this:

```
MariaDB [mysql]> insert into mysql.wsrep_allowlist(ip) values('3.70.155.51');
Query OK, 1 row affected (0.002 sec)

MariaDB [mysql]> select * from wsrep_allowlist;
+----------------+
| ip             |
+----------------+
| 18.193.102.155 |
| 18.194.147.243 |
| 3.70.155.51    |
+----------------+
3 rows in set (0.000 sec)
```

And now we are back to having a three-node Galera Cluster.

Cluster View

One of the new Galera related system tables is the `wsrep_cluster` table. This new table, starting in version 4 of Galera, contains a current view of the cluster. That is to say, it stores the UUID of the cluster and some other identification information, as well as the cluster's capabilities.

To see the names of the columns in this table, either use the `DESCRIBE` statement or execute the following SQL statement from the `mysql` client on one of the nodes in the cluster:

```
SELECT COLUMN_NAME FROM information_schema.columns
WHERE table_schema='mysql'
AND table_name='wsrep_cluster';

+------------------+
| COLUMN_NAME      |
+------------------+
| cluster_uuid     |
| view_id          |
| view_seqno       |
| protocol_version |
| capabilities     |
+------------------+
```

The `cluster_uuid` contains the UUID of the cluster.

The `view_id` corresponds to the status value of the `wsrep_cluster_conf_id`, the number of cluster configuration changes which have occurred in the cluster. The `view_seqno` on the other hand, corresponds to Galera sequence number associated with the cluster view. The protocol version is the same value as contained in the `wsrep_protocol_version` variable. It is the protocol version of the MySQL-wsrep or the MariaDB wsrep

patch. Last, the `capabilities` column contains the capabilities bitmask provided by the Galera library. It is metadata that is needed to recover node state during crash recovery.

If you execute the following SQL statement from any node in a cluster, you can see the contents of this table:

```
SELECT * FROM mysql.wsrep_cluster \G

*************************** 1. row ***************************
    cluster_uuid: bd5fe1c3-7d80-11e9-8913-4f209d688a15
         view_id: 3
      view_seqno: 2956
protocol_version: 4
    capabilities: 184703
```

In the results here, you can see the cluster UUID. This can also be found by using the SQL statement, `SHOW STATUS` for the variable, `wsrep_local_state_uuid`.

Cluster Members

Another Galera related system tables is the `wsrep_cluster_members` table. This system table will provide the current membership of the cluster; it will contain a row for each node in the cluster. That is to say, each node in the cluster known to the node upon which the table is queried.

To see the names of columns in this table, either use the `DESCRIBE` statement or execute the following SQL statement from the `mysql` client on one of the nodes in the cluster:

```
SELECT COLUMN_NAME FROM information_schema.columns
WHERE table_schema='mysql'
AND table_name='wsrep_cluster_members';

+-----------------------+
| COLUMN_NAME           |
+-----------------------+
| node_uuid             |
| cluster_uuid          |
| node_name             |
| node_incoming_address |
+-----------------------+
```

The `node_uuid` records the UUID of each node in the cluster. The `cluster_uuid` is the UUID of the cluster for which the node belongs–the one on which the table has been queried. This is currently the same as what's contained in the `wsrep_cluster table`. The `node_name` contains the human readable name of each node, Last, the `node_incoming_address` stores the IP address and port on which each node is listening for client connections.

If you execute the following SQL statement from any node in a cluster, you can see the contents of this table:

```
SELECT * FROM mysql.wsrep_cluster_members ORDER BY node_name \G

*************************** 1. row ***************************
           node_uuid: e39d1774-7e2b-11e9-b5b2-7696f81d30fb
        cluster_uuid: bd5fe1c3-7d80-11e9-8913-4f209d688a15
           node_name: galera1
node_incoming_address: AUTO
*************************** 2. row ***************************
           node_uuid: eb8fc512-7e2b-11e9-bb74-3281cf207f60
        cluster_uuid: bd5fe1c3-7d80-11e9-8913-4f209d688a15
           node_name: galera2
```

(continues on next page)

(continued from previous page)

```
node_incoming_address: AUTO
*************************** 3. row ***************************
            node_uuid: 2347a8ac-7e2c-11e9-b6f0-da90a2d0a563
         cluster_uuid: bd5fe1c3-7d80-11e9-8913-4f209d688a15
            node_name: galera3
node_incoming_address: AUTO
```

In the results of this example you can see that this cluster is composed of three nodes. The node UUIDs are unique for each node. Notice that the cluster UUID is the same for all three and corresponds to the related value found in the `wsrep_cluster` table shown in the example earlier. Each node has a unique name (for example, galera1). They were named in the configuration file using the `wsrep_node_name` parameter. The incoming node address is set to `AUTO` for all of these nodes, but they can be set individual to specific nodes with the `wsrep-node-address` or the `bind-address` parameter in each node's configuration file.

Cluster Streaming Log

The last Galera related system tables is the `wsrep_streaming_log` table. This system table contains meta data and row events for ongoing streaming transactions, write set fragment per row.

The `node_uuid` column contains the node UUID of the hosting node for the transaction (that is node where the client is executing the transaction). The `trx_id` column stores the transaction identifier, whereas the `seqno` stores the sequence number of the write set fragment. Last, the `flags` columns records flags associated with the write set fragment, and `frag` contains the binary log replication events contained in the write set fragment.

To see the names of columns in this table, either use the `DESCRIBE` statement or execute the following SQL statement from the `mysql` client on one of the nodes in the cluster:

```
SELECT COLUMN_NAME FROM information_schema.columns
WHERE table_schema='mysql'
AND table_name='wsrep_streaming_log';

+-------------+
| COLUMN_NAME |
+-------------+
| node_uuid   |
| trx_id      |
| seqno       |
| flags       |
| frag        |
+-------------+
```

If you execute the following SQL statement from any node in a cluster, you can see the contents of this table:

```
SELECT * FROM mysql.wsrep_streaming_log \G
```

Typically, you won't see any results since it will contain entries only for transactions which have streaming replication enabled. For example:

```
CREATE TABLE table1 (col1 INT PRIMARY KEY);

SET SESSION wsrep_trx_fragment_size=1;

START TRANSACTION;

INSERT INTO table1 VALUES (100);
```

(continues on next page)

(continued from previous page)

```
SELECT node_uuid, trx_id, seqno, flags
FROM mysql.wsrep_streaming_log;

+--------------------------------------+--------+-------+-------+
| node_uuid                            | trx_id | seqno | flags |
+--------------------------------------+--------+-------+-------+
| a006244a-7ed8-11e9-bf00-867215999c7c |     26 |     4 |     1 |
+--------------------------------------+--------+-------+-------+
```

You can see in the results from the example here that the node UUID matches that of the third node (that is, `galera3`) in the results for the example above related to the `wsrep_cluster_members` table. In this example, the `frag` column was omitted from the `SELECT` statement since it contains binary characters that do not format well.

Note: Galera Cluster no longer uses `INFORMATION_SCHEMA.PROCESSLIST`, since it has been deprecated upstream. Instead, it uses "PERFORMANCE_SCHEMA.PROCESSLIST". See the example below:

```
SET GLOBAL wsrep_applier_threads = 10;
SELECT COUNT(*) AS EXPECT_10 FROM performance_schema.threads WHERE NAME = 'thread/sql/
↪wsrep_applier_thread';
```

Or:

```
SELECT COUNT(*) IN (1, 2) FROM performance_schema.processlist WHERE USER = 'system␣
↪user' AND STATE LIKE '%committed%';
```

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Total Order Isolation* (page 83)
- ***Rolling Schema Upgrade*** **(page 83)**
  - *Non-Blocking Operations* (page 84)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

4.5 Schema Upgrades

Schema changes are of particular interest related to Galara Cluster. Schema changes are DDL (Data Definition Language) statement executed on a database (for example, `CREATE TABLE`, `GRANT`). These DDL statements change the database itself and are non-transactional.

Galera Cluster processes schema changes by three different methods:

- *Total Order Isolation* (page 83): Abbreviated as TOI, these are schema changes made on all cluster nodes in the same total order sequence, preventing other transactions from committing for the duration of the operation.
- *Rolling Schema Upgrade* (page 83) Known also as RSU, these are schema changes run locally, affecting only the node on which they are run. The changes do not replicate to the rest of the cluster.
- *Non-Blocking Operations* (page 84): Abbreviated as NBO, these are schema changes made on all cluster nodes in the same total order sequence, preventing other transactions from committing for the duration of the operation, with much more efficient locking strategy that the TOI method.

You can set the method for online schema changes by using the `wsrep_OSU_method` parameter in the configuration file, (`my.ini` or `my.cnf`, depending on your build) or through the ``mysql` client. Galera Cluster defaults to the Total Order Isolation method.

Note: If you are using Galera Cluster for Percona XtraDB Cluster, see the the pt-online-schema-change in the Percona Toolkit.

Total Order Isolation

When you want an online schema change to replicate through the cluster and do not care that other transactions will be blocked while the cluster processes the DDL statements, use the *Total Order Isolation* method. You can do this with a global `SET` statement, as follows:

```
SET GLOBAL wsrep_OSU_method='TOI';
```

The GLOBAL command does not change the "wsrep_OSU_method" for the running session. If you want to change it for the running session, use the session-based `SET` statement, as follows:

```
SET SESSION wsrep_OSU_method='TOI';
```

In Total Order Isolation, queries that change the schema replicate as statements to all nodes in the cluster. The nodes wait for all preceding transactions to commit simultaneously, then they execute the schema change in isolation. For the duration of the DDL processing, no other transactions can commit.

The main advantage of Total Order Isolation is its simplicity and predictability, which guarantees data consistency. Additionally, when using Total Order Isolation, you should take the following particularities into consideration:

- From the perspective of certification, schema upgrades in Total Order Isolation never conflict with preceding transactions, given that they only execute after the cluster commits all preceding transactions. What this means is that the certification interval for schema changes using this method has a zero length. Therefore, schema changes will never fail certification and their execution is guaranteed.
- Transactions that were in progress while the DDL was running and that involved the same database resource will get a deadlock error at commit time and will be rolled back.
- The cluster replicates the schema change query as a statement before its execution. There is no way to know whether or not individual nodes succeed in processing the query. This prevents error checking on schema changes in Total Order Isolation.

Rolling Schema Upgrade

When you want to maintain high-availability during schema upgrades and can avoid conflicts between new and old schema definitions, use the *Rolling Schema Upgrade* method. You can do this with a global `SET` statement, as follows:

```
SET GLOBAL wsrep_OSU_method='RSU';
```

The GLOBAL command does not change the "wsrep_OSU_method" for the running session. If you want to change it for the running session, use the session-based `SET` statement, as follows:

```
SET SESSION wsrep_OSU_method='RSU';
```

In Rolling Schema Upgrade, queries that change the schema are only processed on the local node. While the node processes the schema change, it desynchronizes with the cluster. When it finishes processing the schema change, it applies delayed replication events and synchronizes itself with the cluster.

To change a schema cluster-wide, you must manually execute the query on each node in turn. Bear in mind that during a rolling schema change the cluster continues to operate, with some nodes using the old schema structure while others use the new schema structure.

The main advantage of the Rolling Schema Upgrade is that it only blocks one node at a time. The main disadvantage of the Rolling Schema Upgrade is that it is potentially unsafe, and may fail if the new and old schema definitions are incompatible at the replication event level.

Non-Blocking Operations (this feature is part of Galera Cluster Enterprise Edition)

When you want an online schema change to replicate through the cluster, but are worried that long-running DDL statements block cluster updates, use the *Non-Blocking Operations* method. You can do this with a global `SET` statement, as follows:

```
SET GLOBAL wsrep_OSU_method='NBO';
```

The GLOBAL command does not change the "wsrep_OSU_method" for the running session. If you want to change it for the running session, use the session-based `SET` statement, as follows:

```
SET SESSION wsrep_OSU_method='NBO';
```

The NBO method resembles the TOI method. Queries that change the schema replicate as statements to all nodes in the cluster. The nodes wait for all preceding transactions to commit simultaneously, then they execute the schema change in isolation. For the duration of the DDL processing, no other transactions can commit.

The main advantage of Non-Blocking Operations is that it significantly reduces the impact of DDL statements on the cluster. During DDL processing:

- You can alter another table, using NBO
- You can continue inserting data, excluding the table(s) you are altering
- If one node crashes, the operation will continue on the other nodes, and if successful it will persist

When using Non-Blocking Operations, take the following particularities into consideration:

- The supported statements are:
  - `ALTER TABLE table_name LOCK = {SHARED|EXCLUSIVE} , alter_specification`
  - `ALTER TABLE table_name LOCK = {SHARED|EXCLUSIVE} PARTITION`. The comma after `LOCK=SHARED|EXCLUSIVE` is not used for partition-management `ALTER`s.
  - `ANALYZE TABLE`

  - `OPTIMIZE TABLE`

- The unsupported statements are:

  - `ALTER TABLE LOCK = {DEFAULT|NONE}`. This also means that `ALTER TABLE` without a `LOCK` clause is not supported, as is defaults to `DEFAULT`.

  - `CREATE`

  - `RENAME`

  - `DROP`

  - `REPAIR`

- As some DDL statements, such as `CREATE` without a `LOCK` argument, return an error, it is not recommended to use NBO on a server-wide basis. Only use it for sessions that run compatible DDL statements.

- You cannot perform writes on a table that is being altered under NBO. Write attempts are blocked, until the `ALTER` is complete. Under `LOCK=SHARED`, reading from the table is allowed. Under `LOCK=EXCLUSIVE`, read operations are also blocked.

- Locking the tables at the beginning of the operation is a blocking operation. The cluster may block, if there is an ongoing long transaction against the table being altered. To avoid this, ensure that no clients have open transactions that include the table, prior to running the `ALTER` statement.

- While a DDL operation is running, nodes cannot be donors for SST. Thus, a node cannot join or rejoin the cluster using SST while an NBO DDL is in progress.

- If a node leaves the cluster while an NBO DDL operation is in progress, its data files will be inconsistent and it can only rejoin the cluster through SST, not IST.

- If a DDL statement is expected to take one hour, SST will not be available for one hour, only IST. Set a high-enough value for the `gcache.size` so that there is sufficient cached data to use IST.

- Do not use NBO with statements that operate on more than one table at a time.

- Do not perform online schema upgrades using the RSU method while a statement is running under the NBO method.

Warning: To avoid conflicts between new and old schema definitions, execute SQL statements such as `CREATE TABLE` and `DROP TABLE` using the *Total Order Isolation* (page 83) method.

Note: Contact Codership sales at sales@galeracluster.com for more information, and to get the Galera Cluster Enterprise Edition software.

Related Documents

- *Total Order Isolation* (page 83)
- *Rolling Schema Upgrade* (page 83)
- *Non-Blocking Operations* (page 84)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training

- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Rolling Upgrade* (page 86)
- *Bulk Upgrade* (page 88)
- *Provider Upgrade* (page 89)
- *gcache.size* (page 288)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

4.6 Upgrading Galera Cluster

Since high-availability is a priority for many Galera Cluster administrators, how to go about upgrading the nodes is important. Doing so with the least amount of downtime is tricky. There are three methods for upgrading Galera Cluster, the Galera software on the individual nodes:

- *Rolling Upgrade* (page 86) permits you to upgrade one node at a time, without taking down the cluster—and newly upgraded nodes can join the cluster without problems.
- *Bulk Upgrade* (page 88) is the method by which you take down the cluster and upgrade all of the nodes together.
- *Provider Upgrade* (page 89) is a method in which you only upgrade the Galera Replication Plugin on each node.

There are advantages and disadvantages to each of these methods. For instance, while a rolling upgrade may prove time consuming, the cluster continues to run during the upgrades. Similarly, while a bulk upgrade is faster, depending on on your situation, problems can result from taking down the cluster for a longer period of time. You will have to choose the best method for your situation, needs and concerns.

Rolling Upgrade

When you need the cluster to remain live and do not mind the time it takes to upgrade each node, use rolling upgrades.

In rolling upgrades, you take each node down individually, upgrade its software and then restart the node. When the node reconnects, it brings itself back into sync with the cluster, as it would in the event of any other outage. Once the individual finishes syncing with the cluster, you can move to the next in the cluster.

Note: When upgrading the wsrep provider, sometimes the wsrep protocol version can change. The wsrep provider should not start using the new protocol version until all cluster nodes have been upgraded to the new version. Usually this is not a problem with a rolling upgrade. However, this may cause issues if you restart a non-upgraded node in a cluster where the rest of the nodes have been upgraded.

The main advantage of a rolling upgrade is that in the event that something goes wrong with the upgrade, the other nodes remain operational, giving you time to troubleshoot the problem.

Some of the disadvantages to consider in rolling upgrades are:

Time Consumption Performing a rolling upgrade can take some time, longer depending on the size of the databases and the number of nodes in the cluster, during which the cluster operates at a diminished capacity.

Unless you use *Incremental State Transfer*, as you bring each node back online after an upgrade, it initiates a full *State Snapshot Transfer*, which can take a long time to process on larger databases and slower state transfer methods.

During the State Snapshot Transfer, the node continues to accumulate catch-up in the replication event queue, which it will then have to replay to synchronize with the cluster. At the same time, the cluster is operational and continues to add further replication events to the queue.

Blocking Nodes When the node comes back online, if you use `mysqldump` for State Snapshot Transfers, the *Donor Node* remains blocked for the duration of the transfer. In practice, this means that the cluster is short two nodes for the duration of the state transfer, one for the donor node and one for the node in catch-up.

Using `xtrabackup` or `rsync` with the LVM state transfer methods, you can avoid blocking the donor, but doing so may slow the donor node down.

Depending on the load balancing mechanism, you may have to configure the load balancer not to direct requests at joining and donating nodes.

Cluster Availability Taking down nodes for a rolling upgrade can greatly diminish cluster performance or availability, such as if there are too few nodes in the cluster to begin with or where the cluster is operating at its maximum capacity.

In such cases, losing access to two nodes during a rolling upgrade can create situations where the cluster can no longer serve all requests made of it or where the execution times of each request increase to the point where services become less available.

Cluster Performance Each node you bring up after an upgrade, diminishes cluster performance until the node buffer pool warms back up. Parallel applying can help with this.

Rolling Upgrade Procedure

Assuming you've read and considered the above, below are the steps for upgrading each node in a cluster—one at a time. This procedure, though, is for minor upgrades, not major upgrades. For those, see the next section.

- First, transfer all client connections from the node you are about to upgrade to the other nodes.
- When there are no more client connections trying to access the node, shut down the database software (that is, `mysqld`). This will remove the node from the cluster.
- Now use the method you prefer to upgrade the software. A package management utility such as `yum`, or whatever is appropriate for the your operating system distribution.
- When you've finished updating the database and Galera software, start the node. Check that it has successfully joined the cluster and finished synchronizing before beginning the process to upgrade another node in the cluster.

Tip: If you upgrade a node that will be part of a weighted *Quorum*, set the initial node weight to zero. This guarantees that if the joining node should fail before it finishes synchronizing, it won't affect any quorum computations that follow.

Rolling Upgrades of Major Versions of Galera Cluster

Performing a rolling upgrade between major versions of Galera Cluster (for example, from 8.0 to 8.4) has certain additional limitations. Below is a list of them; you should consider these factors.

SST is not supported between nodes of different major versions. Therefore, nodes of different major versions should not coexist in the same cluster for longer than necessary to perform the upgrade;

Prior to performing the upgrade, ensure that the *gcache.size* (page 288) provider option on all nodes is sized so that it can provide IST for the expected duration of the upgrade;

While the cluster contains nodes of multiple versions, avoid running any statements that are only supported in a particular version or statements that have different effect in different versions. For example, do not run DDL statements that are only available in the newer version.

Rolling Major Upgrade Procedure

Below are the steps of the following procedure for performing rolling upgrades between major versions of Galera Cluster.

- Choose one node to upgrade and make sure that all client connections are directed elsewhere. Once it is free of its cluster obligations, shut down the database daemon (fir example, `mysqld`).
- Edit the database configuration file (that is, `my.cnf`) and temporarily comment out the `wsrep_provider` line. This will prevent the node from attempting to rejoin the cluster during the package upgrade process.
- Uninstall all existing `mysql-wsrep` packages and install the new packages using a package manager (for example, `yum`)
- Start the `mysqld` daemon—without connecting to the cluster—and then run the `mysql_upgrade` script, if it wasn't run automatically as part of package installation.
- Last, restore the `wsrep_provider` line in the database configuration and restart the `mysqld` daemon.

Bulk Upgrade

When you want to avoid time-consuming state transfers and the slow process of upgrading each node, one at a time, use a bulk upgrade.

In bulk upgrades, you take all of the nodes down in an idle cluster, perform the upgrades, then bring the cluster back online. This allows you to upgrade your cluster quickly, but does mean a complete service outage for your cluster.

Warning: Always use bulk upgrades when using a two-node cluster, as the rolling upgrade would result in a much longer service outage.

The main advantage of bulk upgrade is that when you are working with huge databases, it is much faster and results in better availability than rolling upgrades.

The main disadvantage is that it relies on the upgrade and restart being quick. Shutting down InnoDB may take a few minutes as it flushes dirty pages. If something goes wrong during the upgrade, there is little time to troubleshoot and fix the problem.

Note: To minimize any issues that might arise from an upgrade, do not upgrade all of the nodes at once. Rather, run the upgrade on a single node first. If it runs without issue, upgrade the rest of the cluster.

To perform a bulk upgrade on Galera Cluster, complete the following steps:

1. Stop all load on the cluster
2. Shut down all the nodes
3. Upgrade software
4. Restart the nodes. The nodes will merge to the cluster without state transfers, in a matter of seconds.
5. Resume the load on the cluster

Note: You can carry out steps 2-3-4 on all nodes in parallel, therefore reducing the service outage time to virtually the time needed for a single server restart.

Provider-Only Upgrade

When you only need to upgrade the Galera provider, you can further optimize the bulk upgrade to only take a few seconds.

Important: In provider-only upgrade, the warmed up InnoDB buffer pool is fully preserved and the cluster continues to operate at full speed as soon as you resume the load.

Upgrading Galera Replication Plugin

If you installed Galera Cluster for MySQL using the binary package from the Codership repository, you can upgrade the Galera Replication Plugin through your package manager..

To upgrade the Galera Replicator Plugin on an RPM-based Linux distribution, run the following command for each node in the cluster:

```
$ yum update galera
```

To upgrade the Galera Replicator Plugin on a Debian-based Linux distribution, run the following commands for each node in the cluster:

```
$ apt-get update
$ apt-get upgrade galera
```

When `apt-get` or `yum` finish, you will have the latest version of the Galera Replicator Plugin available on the node. Once this process is complete, you can move on to updating the cluster to use the newer version of the plugin.

Updating Galera Cluster

After you upgrade the Galera Replicator Plugin package on each node in the cluster, you need to run a bulk upgrade to switch the cluster over to the newer version of the plugin.

1. Stop all load on the cluster.

2. For each node in the cluster, issue the following queries:

```
SET GLOBAL wsrep_provider='none';
SET GLOBAL wsrep_provider='/usr/lib64/galera/libgalera_smm.so';
```

3. One any one node in the cluster, issue the following query:

```
SET GLOBAL wsrep_cluster_address='gcomm://';
```

4. For every other node in the cluster, issue the following query:

```
SET GLOBAL wsrep_cluster_address='gcomm://node1addr';
```

   For `node1addr`, use the address of the node in step 3.

5. Resume the load on the cluster.

Reloading the provider and connecting it to the cluster typically takes less than ten seconds, so there is virtually no service outage.

Related Documents

- *Rolling Upgrade* (page 86)
- *Bulk Upgrade* (page 88)
- *Provider Upgrade* (page 89)
- *gcache.size* (page 288)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *pc.recovery* (page 296)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

4.7 Recovering Primary Component

Cluster nodes can store the *Primary Component* state to disk. The node records the state of the Primary Component and the UUID's of the nodes connected to it. In the event of an outage, once all nodes that were part of the last saved state achieve connectivity, the cluster recovers the Primary Component.

If the write-set position differs between the nodes, the recovery process also requires a full state snapshot transfer.

For more information on this feature, see the *pc.recovery* (page 296) parameter. By default, it is enabled as of version 3.6.

Understanding the Primary Component State

When a node stores the *Primary Component* state to disk, it saves it as the `gvwstate.dat` file. You'll find this file in the database data directory on the server which is acting as the Primary Component.

The node creates and updates this file when the cluster forms or changes the Primary Component. This ensures that the node retains the latest Primary Component state that it was in. If the node loses connectivity, it has the file to reference.

If the node shuts down gracefully, it deletes the file. If the cluster continues after the node has shutdown (that is, there are other nodes that did not shutdown), one of the remaining nodes will become the host to the Primary Component and will create the `gvwstate.dat` file on its file system.

Below is an example of the contents of the `gvwstate.dat` file:

```
my_uuid: d3124bc8-1605-11e4-aa3d-ab44303c044a
#vwbeg
view_id: 3 0dae1307-1606-11e4-aa94-5255b1455aa0 12
bootstrap: 0
member: 0dae1307-1606-11e4-aa94-5255b1455aa0 1
member: 47bbe2e2-1606-11e4-8593-2a6d8335bc79 1
member: d3124bc8-1605-11e4-aa3d-ab44303c044a 1
#vwend
```

The `gvwstate.dat` file is composed of two parts. **Node Information** provides the node's UUID, in the `my_uuid` field. **View Information** provides information on the node's view of the Primary Component, contained between the `#vwbeg` and `#vwend` tags.

The `view_id` forms an identifier for the view from three parts: *view_type*, which always gives a value of `3` to indicate the primary view; and the *view_uuid* and *view_seq* together form a unique value for the identifier.

The `bootstrap` variable indicates whether or not the node is bootstrapped. It does not, though, effect the Primary Component recovery process. The `member` variables contain the UUID's of nodes connecting to the Primary Component.

Modifying the Saved Primary Component State

If you find yourself in the unusual situation where you need to force certain nodes to join each other specifically, you can do so by manually changing the saved *Primary Component* state.

> **Warning:** Under normal circumstances, for safety reasons, you should entirely avoid editing or otherwise modifying the `gvwstate.dat` file. Doing so may lead to unexpected results.

When a node starts for the first time or after a graceful shutdown, it randomly generates and assigns to itself a UUID, which serves as its identifier to the rest of the cluster. If the node finds a `gvwstate.dat` file in the data directory, it reads the `my_uuid` field to find the value it should use.

By manually assigning arbitrary UUID values to the respective fields on each node, you force them to join each other, forming a new Primary Component, as they start.

For example, assume that you have three nodes that you would like to start together to form a new Primary Component for the cluster. You will need to generate three UUID values, one for each node.

```
SELECT UUID();

+--------------------------------------+
| UUID()                               |
+--------------------------------------+
| 47bbe2e2-1606-11e4-8593-2a6d8335bc79 |
+--------------------------------------+
```

You would then take these values and use them to modify the `gwstate.dat` file on `node1`:

```
my_uuid: d3124bc8-1605-11e4-aa3d-ab44303c044a
#vwbeg
view_id: 3 0dae1307-1606-11e4-aa94-5255b1455aa0 12
bootstrap: 0
member: 0dae1307-1606-11e4-aa94-5255b1455aa0 1
member: 47bbe2e2-1606-11e4-8593-2a6d8335bc79 1
member: d3124bc8-1605-11e4-aa3d-ab44303c044a 1
#vwend
```

Then repeat the process for `node2`:

```
my_uuid: 47bbe2e2-1606-11e4-8593-2a6d8335bc79
#vwbeg
view_id: 3 0dae1307-1606-11e4-aa94-5255b1455aa0 12
bootstrap: 0
member: 0dae1307-1606-11e4-aa94-5255b1455aa0 1
member: 47bbe2e2-1606-11e4-8593-2a6d8335bc79 1
member: d3124bc8-1605-11e4-aa3d-ab44303c044a 1
#vwend
```

And, the same again for `node3`:

```
my_uuid: d3124bc8-1605-11e4-aa3d-ab44303c044a
#vwbeg
view_id: 3 0dae1307-1606-11e4-aa94-5255b1455aa0 12
bootstrap: 0
member: 0dae1307-1606-11e4-aa94-5255b1455aa0 1
member: 47bbe2e2-1606-11e4-8593-2a6d8335bc79 1
member: d3124bc8-1605-11e4-aa3d-ab44303c044a 1
#vwend
```

Then start all three nodes without the bootstrap flag. When they start, Galera Cluster reads the `gvwstate.dat` file for each. It pulls its UUID from the file and uses those of the `member` field to determine which nodes it should join in order to form a new Primary Component.

Related Documents

- *pc.recovery* (page 296)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *wsrep_cluster_status* (page 313)
- *wsrep_last_committed* (page 321)
- *pc.bootstrap* (page 297)
- *wsrep_provider_options* (page 254)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

4.8 Resetting the Quorum

Although it is unlikely, you may find your nodes no longer consider themselves part of the *Primary Component*. There might have been a network failure; perhaps more than half of the cluster failed; or there is a split-brain situation. In these cases, the node come to suspect that there is another Primary Component, to which they are no longer connected.

This loss of integrity can be a problem. When it occurs, the nodes will start to return an `Unknown command` error to all of queries they're given to execute: they simply stop performing their duties for fear of making the situation worse by becoming too out-of-sync with their true cluster.

You can see if this is happening by executing the `SHOW STATUS` statement and checking the *wsrep_cluster_status* (page 313) status variable. Specifically, this is done by executing the following SQL statement on each node:

```
SHOW GLOBAL STATUS LIKE 'wsrep_cluster_status';

+----------------------+---------+
| Variable_name        | Value   |
+----------------------+---------+
| wsrep_cluster_status | Primary |
+----------------------+---------+
```

The return value `Primary` indicates that the node on which it was executed is part of the Primary Component. When the query returns any other value it indicates that the node is part of a non-operational component. If none of the nodes return the value `Primary`, you need to reset the *Quorum*.

Situations in which none of the nodes show they are part of the Primary Component are very rare. If you discover one or more nodes wtih a value `Primary`, it may indicate a problem with network connectivity, rather than a need to reset

the quorum. Investigate the connection possibility. Once the nodes regain network connectivity they automatically resynchronize with the Primary Component.

Finding the Most Advanced Node

Before you can reset the quorum, you need to identify the most advanced node in the cluster. That is, you must find the node whose local database committed the last transaction. Regardless of the method you use in resetting the quorum, this node should serve as the starting point for the new *Primary Component*.

Identifying the most advanced node requires that you find the node with the highest sequence number (that is, seqno). You can determine this by checking the *wsrep_last_committed* (page 321) status variable. From the database client on each node, run the following query:

```
SHOW STATUS LIKE 'wsrep_last_committed';

+----------------------+--------+
| Variable_name        | Value  |
+----------------------+--------+
| wsrep_last_committed | 409745 |
+----------------------+--------+
```

The return value is the sequence number for the last transaction the node committed. If the `mysqld` daemon is down, you can restart `mysqld` without starting Galera. If you do not want to restart the databases, you may be able to ascertain the sequence number from the `grastate.dat` file, located in the data directory.

Once you've found the sequence numbers of each node, the one with the highest value is the most advanced one in the cluster. Use that node as the starting point when bootstrapping the new Primary Component. This is explained in the next section here.

Resetting the Quorum

When you reset the quorum, what you are doing is bootstrapping the *Primary Component* on the most advanced node you have available. This node then functions as the new Primary Component, bringing the rest of the cluster into line with its state.

There are two methods available to you in this process: automatic and manual. The recommended one for a quorum reset is the automatic method. Unlike the manual method, automatic bootstrapping preserve the write-set cache, or GCache, on each node. What this means is that when the new Primary Component starts, some or all of the joining nodes can be provisioned quickly using the *Incremental State Transfer* (IST) method, rather than the slower *State Snapshot Transfer* (SST) method.

Automatic Bootstrap

Resetting the quorum will bootstrap the *Primary Component* onto the most advanced node. With the automatic method, this is done by dynamically enabling *pc.bootstrap* (page 297) through the *wsrep_provider_options* (page 254) through the database client—it is not done through the configuration file. Once you set this option, it will make the node a new Primary Component.

To perform an automatic bootstrap, run the following command using the `mysql` client of the most advanced node:

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

The node now operates as the starting node in a new Primary Component. Nodes in nonoperational components that have network connectivity attempt to initiate incremental state transfers if possible, state snapshot transfers if not, with this node, bringing their own databases up-to-date.

Manual Bootstrap

Resetting the quorum bootstraps the *Primary Component* onto the most advanced node. With the manual method, this is done by shutting down the cluster—shutting down `mysqld` on all of the nodes—and then starting `mysqld` with Galera on each node, beginning with the most advanced one.

To bootstrap manually a cluster, first determine the most advanced node by executing the following from the command-line on each node:

```
mysql -u root -p -e "SHOW STATUS LIKE 'wsrep_last_committed'"
```

Once you've determined which node has the highest sequence number, you can begin shutting down the cluster. Just shut down `mysqld` on all of the nodes in the cluster—leaving the most advanced node until last. For servers that use `init`, enter the following from the command-line:

```
service mysql stop
```

For servers that use `systemd`, execute instead this from the command-line:

```
systemctl stop mysql
```

You are now ready to start the cluster again. Start the most advanced node with the `mysqld_bootstrap` command—not the other nodes. For servers that use `init`, run the following command:

```
service mysql start mysqld_bootstrap
```

For servers that use `systemd` and Galera Cluster 5.7 or 8.0, enter instead the following from the command-line:

```
mysqld_bootstrap
```

For MySQL servers that use `systemd` and at least version 5.7 of Galera Cluster, you can execute the following script from the command-line only on the first node:

```
mysqld_bootstrap
```

For MariaDB servers that use `systemd`, you might try to execute the following script from the command-line—again, only on the first node:

```
galera_new_cluster
```

With that first node running and acting as Primary Component, you are not ready to start all of the other nodes in the cluster. For servers that use `init`, run the following command:

```
service mysql start
```

For servers that use `systemd`, instead run this command:

```
systemctl start mysqld
```

Written into all of these scripts is the `--wsrep-new-cluster` option, but it is done with a certain finesse. Whichever method or script you use, when the first node starts with the `--wsrep-new-cluster` option, it initializes a new cluster using the data from the most advanced state available from the previous cluster. As the other nodes start, they connect to this node and request state snapshot transfers, to bring their own databases up-to-date. In a short amount of time, they all should become synchronized and running smoothly.

Related Documents

- *wsrep_cluster_status* (page 313)

- *wsrep_last_committed* (page 321)
- *pc.bootstrap* (page 297)
- *wsrep_provider_options* (page 254)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Galera Status Variables* (page 307)
- *gcs.recv_q_hard_limit* (page 291)
- *gcs.fc_limit* (page 290)
- *gcs.max_throttle* (page 291)
- *gcs.recv_q_soft_limit* (page 291)
- *gcs.fc_factor* (page 290)
- *wsrep_flow_control_sent* (page 319)
- *wsrep_flow_control_recv* (page 319)
- *wsrep_flow_control_paused* (page 318)
- *wsrep_flow_control_paused_ns* (page 318)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

4.9 Managing Flow Control

The cluster replicates changes synchronously through global ordering, but applies these changes asynchronously from the originating node out. To prevent any one node from falling too far behind the cluster, Galera Cluster implements a feedback mechanism called Flow Control.

Nodes queue the write-sets they receive in the global order and begin to apply and commit them on the database. In the event that the received queue grows too large, the node initiates Flow Control. The node pauses replication while it works the received queue. Once it reduces the received queue to a more manageable size, the node resumes replication.

Monitoring Flow Control

Galera Cluster provides global status variables for use in monitoring Flow Control. These break down into those status variables that count Flow Control pause events and those that measure the effects of pauses.

```
SHOW STATUS LIKE 'wsrep_flow_control_%';
```

Running these status variables returns only the node's present condition. You are likely to find the information more useful by graphing the results by using the tool of your choice, so that you can better see the points where Flow Control engages.

For more information on status variables that relate to flow control, see *Galera Status Variables* (page 307).

Monitoring for Flow Control Pauses

When Flow Control engages, it notifies the cluster that it is pausing replication using an `FC_Pause` event. Galera Cluster provides two status variables that monitor for these events.

- *wsrep_flow_control_sent* (page 319) This status variable shows the number of Flow Control pause events sent by the local node since the last status query.
- *wsrep_flow_control_recv* (page 319) This status variable shows the number of Flow Control pause events on the cluster, both those from other nodes and those sent by the local node, since the last status query.

Measuring the Flow Control Pauses

In addition to tracking Flow Control pauses, Galera Cluster also allows you to track the amount of time since the last `FLUSH STATUS` query during which replication was paused due to Flow Control.

You can find this using one of two status variables:

- *wsrep_flow_control_paused* (page 318) Provides the amount of time replication was paused as a fraction. Effectively, how much the replica lag is slowing the cluster. The value `1.0` indicates replication is paused now.
- *wsrep_flow_control_paused_ns* (page 318) Provides the amount of time replication was paused in nanoseconds.

Configuring Flow Control

Galera Cluster provides two sets of parameters that allow you to manage how nodes handle the replication rate and Flow Control. The first set controls the write-set cache, the second relates to the points at which the node engages and disengages Flow Control.

Managing the Replication Rate

These three parameters control how nodes respond to changes in the replication rate. They allow you to manage the write-set cache on an individual node.

- *gcs.recv_q_hard_limit* (page 291) This sets the maximum write-set cache size (in bytes). The parameter value depends on the amount of RAM, swap size and performance considerations.

  The default value is `SSIZE_MAX` minus 2 gigabytes on 32-bit systems. There is no practical limit on 64-bit systems.

  In the event that a node exceeds this limit and *gcs.max_throttle* (page 291) is not set at `0.0`, the node aborts with an out-of-memory error. If *gcs.max_throttle* (page 291) is set at `0.0`., replication in the cluster stops.

- *gcs.max_throttle* (page 291) This sets the smallest fraction to the normal replication rate the node can tolerate in the cluster. If you set the parameter to `1.0` the node does not throttle the replication rate. If you set the parameter for `0.0`, a complete replication stop is possible.

  The default value is `0.25`.

- *gcs.recv_q_soft_limit* (page 291) This serves to estimate the average replication rate for the node. It is a fraction of the *gcs.recv_q_hard_limit* (page 291). When the replication rate exceeds the soft limit, the node calculates the average replication rate (in bytes) during this period. After that, the node decreases the replication rate linearly with the cache size so that at the *gcs.recv_q_hard_limit* (page 291) it reaches the value of the *gcs.max_throttle* (page 291) times the average replication rate.

  The default value is `0.25`.

  Note: When the node estimates the average replication rate, it can reach a value that is way off from the sustained replication rate.

The write-set cache grows semi-logarithmically with time after the *gcs.recv_q_soft_limit* (page 291) and the time needed for a state transfer to complete.

Managing Flow Control

These parameters control the point at which the node triggers Flow Control and the factor used in determining when it should disengage Flow Control and resume replication.

- *gcs.fc_limit* (page 290) This parameter determines the point at which Flow Control engages. When the replica queue exceeds this limit, the node pauses replication.

  It is essential for multi-primary configurations that you keep this limit low. The certification conflict rate is proportional to the replica queue length. In primary-replica setups, you can use a considerably higher value to reduce Flow Control intervention.

  The default value is `16`.

- *gcs.fc_factor* (page 290) This parameter is used in determining when the node can disengage Flow Control. When the replica queue on the node drops below the value of *gcs.fc_limit* (page 290) times that of *gcs.fc_factor* (page 290) replication resumes.

  The default value is `0.5`.

Bear in mind that, while it is critical for multi-primary operations that you use as small a replica queue as possible, the replica queue length is not so critical in primary-replica setups. Depending on your application and hardware, the node can apply even 1K of write-sets in a fraction of a second. The replica queue length has no effect on primary-replica failover.

Warning: Cluster nodes process transactions asynchronously with regards to each other. Nodes cannot anticipate in any way the amount of replication data. Because of this, Flow Control is always reactive. That is, it only

comes into affect after the node exceeds certain limits. It cannot prevent exceeding these limits or, when they are exceeded, it cannot make any guarantee as to the degree they are exceeded.

Meaning, if you were to configure a node with:

```
gcs.recv_q_hard_limit=100Mb
```

That node can still exceed that limit from a 1Gb write-set.

Related Documents

- *Galera Status Variables* (page 307)
- *gcs.recv_q_hard_limit* (page 291)
- *gcs.fc_limit* (page 290)
- *gcs.max_throttle* (page 291)
- *gcs.recv_q_soft_limit* (page 291)
- *gcs.fc_factor* (page 290)
- *wsrep_flow_control_sent* (page 319)
- *wsrep_flow_control_recv* (page 319)
- *wsrep_flow_control_paused* (page 318)
- *wsrep_flow_control_paused_ns* (page 318)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Upgrading Galera Cluster* (page 86)
- *evs.auto_evict* (page 280)
- *evs.delayed_keep_period* (page 281)
- *evs.delay_margin* (page 281)
- *evs.evict* (page 282)
- *evs.version* (page 286)
- *wsrep_evs_delayed* (page 316)
- *wsrep_evs_evict_list* (page 316)
- *wsrep_evs_state* (page 317)

- *wsrep_provider_options* (page 254)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

4.10 Auto-Eviction

When Galera Cluster notices erratic behavior in a node (for example, unusually delayed response times), it can initiate a process to remove the node permanently from the cluster. This process is called *Auto-Eviction*.

Configuring Auto-Eviction

Each node in a cluster monitors the group communication response times from all other nodes in the cluster. When a cluster registers delayed responses from a node, it makes an entry about the node to the delayed list.

If the delayed node becomes responsive again for a fixed period, entries for that node are removed from the delayed list. However, if the node receives enough delayed entries and it is found on the delayed list for the majority of the cluster, the delayed node is evicted permanently from the cluster. Evicted nodes cannot rejoin the cluster until restarted.

You can configure the parameters of Auto-Eviction by setting the following options through *wsrep_provider_options* (page 254):

- *evs.delay_margin* (page 281): This sets the time period that a node can delay its response from expectations until the cluster adds it to the delayed list. You must set this parameter to a value higher than the round-trip delay time (RTT) between the nodes.

  The default value is `PT1S`.

- *evs.delayed_keep_period* (page 281): This sets the time period you require a node to remain responsive until it is removed from the delayed list.

  The default value is `PT30S`.

- *evs.evict* (page 282) This sets the point in which the cluster triggers manual eviction to a certain node value. Setting this parameter as an empty string causes it to clear the evict list on the node where it is set.

- *evs.auto_evict* (page 280): This sets the number of entries allowed for a delayed node before Auto-Eviction takes place. Setting this to `0` disables the Auto-Eviction protocol on the node, though the node will continue to monitor node response times.

  The default value is `0`.

- *evs.version* (page 286): This sets which version of the EVS Protocol the node uses. Galera Cluster enables Auto-Eviction starting with EVS Protocol version 1.

  - If you use Galera Cluster version 3.9 or older, the default value is `0`.
  - If you use Galera Cluster version 4.0 or newer, the default value is `1`.

  To check your version of Galera Cluster, see *wsrep_provider_version* (page 330).

Checking Eviction Status

If you suspect a node is becoming delayed, you can check its eviction status through Galera status variables. You can do this by using the `SHOW STATUS` statement from the database client. You would enter something like this:

```
SHOW STATUS LIKE 'wsrep_evs_delayed';
```

Below are the Galera status variables available to you:

- *wsrep_evs_state* (page 317): This status variable gives the internal state of the EVS Protocol.
- *wsrep_evs_delayed* (page 316): This status variable gives a comma separated list of nodes on the delayed list. The format used in that list is `uuid:address:count`. The `count` refers to the number of entries for the given delayed node.
- *wsrep_evs_evict_list* (page 316): This status variable lists the UUID's of evicted nodes.

Upgrading from Previous Versions

Releases of Galera Cluster prior to version 3.8 use EVS Protocol version 0, which is not directly compatible with version 1. As such, when you upgrade Galera Cluster for a node, the node continues to use EVS Protocol version 0. Releases of Galera Cluster after version 4.0 use EVS Protocol version 1.

To update the EVS Protocol version, you must first update the Galera Cluster software on each node. Here are the steps to do that:

1. Choose a node to start the upgrade and stop `mysqld` on it. For systems that use `init`, run the following command:

   ```
   # service mysql stop
   ```

   For systems that run `systemd`, use instead this command:

   ```
   # systemctl stop mysql
   ```

2. Once you stop `mysqld`, update the Galera Cluster software for the node. This can vary depending on how you installed Galera Cluster and which database server and operating system distribution the server uses.

3. Using a text editor, edit the configuration file, `/etc/my.cnf`. Set the EVS Protocol version to `0`.

   ```
   wsrep_provider_options="evs.version=0"
   ```

4. After saving the configuration file, restart the node. For systems that use `init`, run the following command:

   ```
   # service mysql start
   ```

   For systems that run `systemd`, instead use this command:

   ```
   # systemctl start mysql
   ```

5. Using the database client, check the node state with the `SHOW STATUS` statement like so:

   ```
   SHOW STATUS LIKE 'wsrep_local_state_comment';

   +----------------------------+--------+
   | Variable_name              | Value  |
   +----------------------------+--------+
   | wsrep_local_state_comment  | Joined |
   +----------------------------+--------+
   ```

When the node state reads as `Synced`, the node is back in sync with the cluster.

Repeat the above steps on each node in the cluster to update them. Once this process is finished, the cluster will have the latest version of Galera Cluster. You can then begin updating the EVS Protocol version for each node. Below are the steps to do that:

1. On the first node, edit the configuration file, `/etc/my.cnf` with a text editor. Change the EVS Protocol version in it like so:

   ```
   wsrep_provider_options="evs.version=1"
   ```

2. After saving, restart `mysqld`. If your system uses `init`, run the following command:

   ```
   # service mysql restart
   ```

   For system that run `systemd`, use instead this command:

   ```
   # systemctl restart mysql
   ```

3. Using the database client, execute the `SHOW STATUS` statement to see if the EVS Protocol is using version 1. This time give it the new *wsrep_evs_state* (page 317) status variable.

   ```
   SHOW STATUS LIKE 'wsrep_evs_state';
   ```

   If the `SHOW STATUS` statement returns an empty set, something went wrong and your database server is still using EVS Protocol version 0. If it returns a results set, the EVS Protocol is on the right version and you can proceed.

4. Once you confirm the server is using the right version, check the node state. Execute the `SHOW STATUS` statement like so:

   ```
   SHOW STATUS LIKE 'wsrep_local_state_comment';

   +----------------------------+--------+
   | Variable_name              | Value  |
   +----------------------------+--------+
   | wsrep_local_state_comment  | Joined |
   +----------------------------+--------+
   ```

   When the node state reads as `Synced`, the node is back in sync with the cluster.

These steps will update the EVS Protocol version for one node in a cluster. Repeat the process on each of the remaining nodes so that they all use EVS Protocol version 1.

For more information on upgrading in general, see *Upgrading Galera Cluster* (page 86).

Related Documents

- *Upgrading Galera Cluster* (page 86)
- *evs.auto_evict* (page 280)
- *evs.delayed_keep_period* (page 281)
- *evs.delay_margin* (page 281)
- *evs.evict* (page 282)
- *evs.version* (page 286)
- *wsrep_evs_delayed* (page 316)
- *wsrep_evs_evict_list* (page 316)

- *wsrep_evs_state* (page 317)
- *wsrep_provider_options* (page 254)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *When to Stream* (page 33)
- *wsrep_trx_fragment_unit* (page 267)
- *wsrep_trx_fragment_size* (page 267)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

4.11 Using Streaming Replication

When a node replicates a transaction under *Streaming Replication*, it breaks the transaction into fragments, and then certifies and applies the fragments to replica nodes while the transaction is still in progress.

This allows you to work with larger data-sets, manage hot records, and help avoid conflicts and hangs in the case of long-running transactions.

Note: Streaming Replication is a new feature introduced in version 4.0 of Galera Cluster. Older versions do not support these operations.

Enabling Streaming Replication

The best practice when working with *Streaming Replication* is to enable it at a session-level for specific transactions, or parts thereof. The reason is that Streaming Replication increases the load on all nodes when applying and rolling back transactions. You'll get better performance if you only enable Streaming Replication on those transactions that won't run correctly without it.

For more information, see *When to Use Streaming Replication* (page 33).

Enabling Streaming Replication requires you to define the replication unit and number of units to use in forming the transaction fragments. Two parameters control these variables: *wsrep_trx_fragment_unit* (page 267) and *wsrep_trx_fragment_size* (page 267).

Below is an example of how to set these two parameters:

```
SET SESSION wsrep_trx_fragment_unit='statements';
SET SESSION wsrep_trx_fragment_size=3;
```

In this example, the fragment is set to three statements. For every three statements from a transaction, the node will generate, replicate and certify a fragment.

You can choose between a few replication units when forming fragments:

- **bytes** This defines the fragment size in bytes.
- **rows** This defines the fragment size as the number of rows the fragment updates.
- **statements** This defines the fragment size as the number of statements in a fragment.

Choose the replication unit and fragment size that best suits the specific operation you want to run.

Streaming Replication with Hot Records

When your application needs to update frequently the same records from the same table (for example, implementing a locking scheme, a counter, or a job queue), Streaming Replication allows you to force critical changes to replicate to the entire cluster.

For instance, consider the use case of a web application that creates work orders for a company. When the transaction starts, it updates the table `work_orders`, setting the queue position for the order. Under normal replication, two transactions can come into conflict if they attempt to update the queue position at the same time.

You can avoid this with Streaming Replication. As an example of how to do this, you would first execute the following SQL statement to begin the transaction:

```
START TRANSACTION;
```

After reading the data that you need for the application, you would enable Streaming Replication by executing the following two `SET` statements:

```
SET SESSION wsrep_trx_fragment_unit='statements';
SET SESSION wsrep_trx_fragment_size=1;
```

Next, set the user's position in the queue like so:

```
UPDATE work_orders
SET queue_position = queue_position + 1;
```

With that done, you can disable Streaming Replication by executing one of the previous `SET` statements, but with a different value like so:

```
SET SESSION wsrep_trx_fragment_size=0;
```

You can now perform whatever additional tasks you need to prepare the work order, and then commit the transaction:

```
COMMIT;
```

During the work order transaction, the client initiates Streaming Replication for a single statement, which it uses to set the queue position. The queue position update then replicates throughout the cluster, which prevents other nodes from coming into conflict with the new work order.

Related Documents

- *When to Stream* (page 33)
- *wsrep_trx_fragment_unit* (page 267)
- *wsrep_trx_fragment_size* (page 267)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Backing Up Cluster Data* (page 109)
- *Galera Parameters* (page 271)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

4.12 Galera Arbitrator

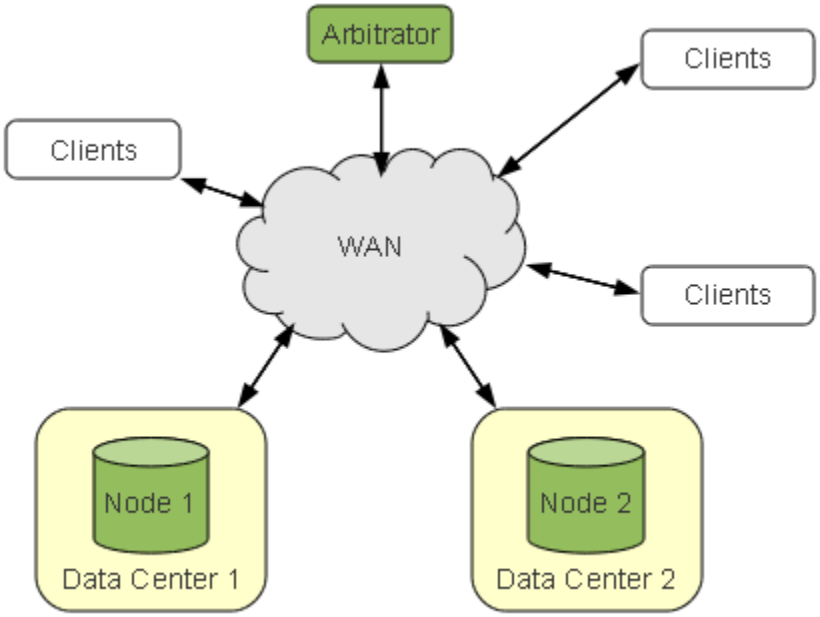
When deploying a Galera Cluster, it is recommended to use a minimum of three instances: three nodes, three data centers and so on.

If the cost of adding resources (such as a third data center) is too much, you can use *Galera Arbitrator*. Galera Arbitrator is a member of a cluster that participates in voting, but not in the actual replication.

Warning: While Galera Arbitrator does not participate in replication, it does receive the same data as all other nodes. You must secure its network connection.

Galera Arbitrator serves two purposes: When you have an even number of nodes, it functions as an odd node, to avoid split-brain situations. It can also request a consistent application state snapshot, which is useful in making backups.

Galera Arbitrator



If one datacenter fails or loses its WAN connection, the node that sees the arbitrator—and by extension sees clients—continues operation.

Note: Even though Galera Arbitrator does not store data, it must see all replication traffic. Placing Galera Arbitrator in a location with poor network connectivity to the rest of the cluster may lead to poor cluster performance.

In the event that Galera Arbitrator fails, it won't affect cluster operation. You can attach a new instance to the cluster at any time and there can be several instances running in the cluster.

For more information on using Galera Arbitrator for making backups, see *Backing Up Cluster Data* (page 109).

Starting Galera Arbitrator

Galera Arbitrator is a separate daemon from Galera Cluster, called `garbd`. This means that you must start it separately from the cluster. It also means that you cannot configure Galera Arbitrator through the `my.cnf` configuration file.

How you configure Galera Arbitrator depends on how you start it. That is to say, whether it runs from the shell or as a service. These two methods are described in the next two sections.

Note: When Galera Arbitrator starts, the script executes a `sudo` statement as the user `nobody` during its process. There is a particular issue in Red Hat Enterprise Linux and some other distributions of Linux, in which the default `sudo` configuration will block users that operate without `tty` access. To correct this, edit with a text editor the `/etc/sudoers` file and comment out this line:

```
Defaults requiretty
```

This will prevent the operating system from blocking Galera Arbitrator.

Starting Galera Arbitrator from the Shell

When starting Galera Arbitrator from the shell, you have two options as to how you may configure it. You can set the parameters through the command line arguments, as in the example here:

```
$ garbd --group=example_cluster \
    --address="gcomm://192.168.1.1,192.168.1.2,192.168.1.3" \
    --option="socket.ssl_key=/etc/ssl/galera/server-key.pem;socket.ssl_cert=/etc/ssl/
↪galera/server-cert.pem;socket.ssl_ca=/etc/ssl/galera/ca-cert.pem;socket.ssl_
↪cipher=AES128-SHA256""
```

If you use SSL, it is necessary to specify the cipher. Otherwise, after initializing the SSL context, an error will occur with a message saying, "Terminate called after throwing an instance of 'gu::NotSet'".

If you do not want to enter the options every time you start Galera Arbitrator from the shell, you can set the options in the `arbitrator.config` configuration file:

```
# arbitrator.config
group = example_cluster
address = gcomm://192.168.1.1,192.168.1.2,192.168.1.3
```

Then, to enable those options when you start Galera Arbitrator, use the `--cfg` option like so:

```
$ garbd --cfg /path/to/arbitrator.config
```

For more information on the options available to Galera Arbitrator through the shell, run `garbd` with the `--help` argument.

```
$ garbd --help

Usage: garbd [options] [group address]

Configuration:
  -d [ --daemon ]       Become daemon
  -n [ --name ] arg     Node name
  -a [ --address ] arg  Group address
  -g [ --group ] arg    Group name
  --sst arg             SST request string
  --donor arg           SST donor name
  -o [ --options ] arg  GCS/GCOMM option list
  -l [ --log ] arg      Log file
  -c [ --cfg ] arg      Configuration file

Other options:
  -v [ --version ]      Print version
  -h [ --help ]         Show help message
```

In addition to the standard configuration, any parameter available to Galera Cluster also works with Galera Arbitrator, except for those prefixed by `repl`. When you start it from the shell, you can set those using the `--option` argument.

For more information on the options available to Galera Arbitrator, see *Galera Parameters* (page 271).

Starting Galera Arbitrator as a Service

When starting Galera Aribtrator as a service, whether using `init` or `systemd`, you would use a different format for the configuration file than you would use when starting it from the shell. Below is an example of the configuration file:

```
# Copyright (C) 2013-2015 Codership Oy
# This config file is to be sourced by garbd service script.

# A space-separated list of node addresses (address[:port]) in the cluster:
GALERA_NODES="192.168.1.1:4567 192.168.1.2:4567"

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

# Optional Galera internal options string (such as SSL settings)
# see https://galeracluster.com/documentation/galera-parameters.html
GALERA_OPTIONS="socket.ssl_cert=/etc/galera/cert/cert.pem;socket.ssl_key=/$"

# Log file for garbd. Optional, by default logs to syslog
LOG_FILE="/var/log/garbd.log"
```

In order for Galera Arbitrator to use the configuration file, you must place it in a file directory where your system looks for service configuration files. There is no standard location for this directory; it varies from distribution to distribution, though it usually in `/etc` and at least one sub-directory down. Some common locations include:

- `/etc/defaults/`
- `/etc/init.d/`
- `/etc/systemd/`
- `/etc/sysconfig/`

Check the documentation for the operating system distribution your server uses to determine where to place service configuration files.

Once you have the service configuration file in the right location, you can start the `garb` service. For systems that use `init`, run the following command:

```
# service garb start
```

For systems that run `systemd`, use instead this command:

```
# systemctl start garb
```

This starts Galera Arbitrator as a service. It uses the parameters set in the configuration file.

In addition to the standard configuration, any parameter available to Galera Cluster also works with Galera Arbitrator, excepting those prefixed by `repl`. When you start it as a service, you can set those using the `GALERA_OPTIONS` parameter.

For more information on the options available to Galera Arbitrator, see *Galera Parameters* (page 271).

Related Documents

- *Backing Up Cluster Data* (page 109)
- *Galera Parameters* (page 271)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses

- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Scriptable SST* (page 74)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

4.13 Backing Up Cluster Data

You can perform backups with Galera Cluster at the same regularity as with a standard database server, using a backup script. Since replication ensures that all nodes have the exact same data, running a backup script on one node will backup the data on all nodes in the cluster.

The problem with such a simple backup method, though, is that it lacks a *Global Transaction ID* (GTID). You can use backups of this kind to recover data, but they are insufficient for use in recovering nodes to a well-defined state. Furthermore, some backup procedures can block cluster operations during the backup.

Getting backups with the associated Global Transaction ID requires a different approach.

State Snapshot Transfer as Backup

Taking a full data backup is very similar to node provisioning through a *State Snapshot Transfer*. In both cases, the node creates a full copy of the database contents, using the same mechanism to associate a *Global Transaction ID* with the database state. Invoking backups through the state snapshot transfer mechanism has the following benefits:

- The node initiates the backup at a well-defined point.
- The node associates a Global Transaction ID with the backup.
- The node desyncs from the cluster to avoid throttling performance while making the backup, even if the backup process blocks the node.
- The cluster knows that the node is performing a backup and won't choose the node as a donor for another node.

In order to use this method for backups, you will need to use a script that implements both your preferred backup procedure and the Galera Arbitrator daemon, triggering it in a manner similar to a state snapshot transfer. You would execute such a script from the command-line, as described below:

On the node where you want to have a backup, start the original SST script manually in "joiner" mode. This command opens a socket listening for a connection from "donor":

```
$ wsrep_sst_rsync --role 'joiner' --address '10.21.32.1:3333' --datadir '/tmp/backup/
↪' \
  --defaults-file '' --defaults-group-suffix '' --parent $$
```

Note the output:

```
$ ready 10.21.32.1:3333/rsync_sst
```

Next, on any node, give the command:

```
$ garbd --address gcomm://10.21.32.1:4567?gmcast.listen_addr=tcp://0.0.0.0:4560 \
  --group my_cluster --sst rsync:10.21.32.1:3333/rsync_sst
```

Note: In the command, `?gmcast.listen_addr=tcp://0.0.0.0:4560` is an arbitrary listen socket address that Galera Arbitrator opens to communicate with the cluster. You only need to specify this in the event that the default socket address (that is, `0.0.0.0:4567`) is busy.

Note: In the command, the value of the `--sst` option is `<sst_method>:<sst_address>`, where `<sst_address>` is given in the output of the joiner script above.

Note: You may find it useful to create your backup script using a modified version of the standard state snapshot transfer script. For information on scripts of this kind, see *Scriptable State Snapshot Transfers* (page 74).

Related Documents

- *Scriptable SST* (page 74)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Container Deployments* (page 127)
- *Deployment Variants* (page 112)
- *Load Balancing* (page 118)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

CHAPTER

FIVE

DEPLOYMENT

When you start Galera Cluster, you have to do so by initializing a series of nodes that are configured to communicate with each other and to replicate each other. Each node in the cluster is a particular instance of a MySQL, MariaDB, or Percona XtraDB database server. How your application servers interact with the cluster and how you manage the load and the individual nodes represents your deployment.

Cluster Deployment Variants (page 112)

Galera Cluster provides synchronous multi-primary replication. This means that the nodes collectively operate as a single database server that listens across many interfaces. This section provides various examples of how you might deploy a cluster in relation to your application servers.

Load Balancing (page 118)

In high availability environments, you may sometimes encounter situations in which some nodes have a much greater load of traffic than others. If you discover such a situation, there may be some benefit in configuring and deploying load balancers between your application servers and Galera Cluster. Doing so will allow you to distribute client connections more evenly between the nodes, ensuring better performance.

This section provides guides to installing, configuring and deploying HAProxy, Pen and Galera Load Balancer, helping you to manage traffic between clients and the cluster.

Container Deployments (page 127)

When using the standard deployment methods of Galera Cluster, nodes run directly on the server hardware – interacting directly with the operating system, that is, Linux or FreeBSD (FreeBSD is available up to Galera Cluster for MySQL 5.7). By contrast, with container deployments nodes run in containerized virtual environments on the server. You may find containers useful in building portable deployments across numerous machines, when testing applications or scripting installations, or when isolating processes for security.

This section provides guides to installing, configuring and deploying Galera Cluster nodes in container instances using FreeBSD Jails and Docker.

Related Documents

- *Container Deployments* (page 127)
- *Deployment Variants* (page 112)
- *Load Balancing* (page 118)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses

- Tutorial Articles
- Training Videos
- FAQ
- search
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

5.1 Cluster Deployment Variants

A Galera Cluster will consist of multiple nodes, preferably three or more. Each node is an instance of MySQL, MariaDB or Percona XtraDB that you convert to Galera Cluster, allowing you to use that node as a cluster base.

Galera Cluster provides synchronous multi-primary replication. You can treat the cluster as a single database server that listens through many interfaces. To appreciate this, consider a typical *n*-tier application and the various benefits that would come from deploying it with Galera Cluster.

No Clustering

In the typical *n*-tier application cluster without database clustering, there's no concern for database replication or synchronization.

Internet traffic will be filtered down to your application servers, all of which read and write from the same DBMS server. Given that the upper tiers usually remain stateless, you can start as many instances as you need to meet the demand from the internet. Each instance stores its data in the data tier.

This solution is simple and easy to manage, but has a particular weakness in the data tier's lack of redundancy.

For example, if for any reason the DBMS server become unavailable, your application also becomes unavailable. This is the same whether the server crashes or it has been shut down for maintenance.

Similarly, this deployment also introduces performance concerns. While you can start as many instances as you need to meet the demands on your web and application servers, they can only so much load on the DBMS server can be handled before the load begins to slow end-user activities.

Whole Stack Clustering

In the typical *n*-tier application cluster you can avoid the performance bottleneck by building a whole stack cluster.

Internet traffic filters down to the application server, which stores data on its own dedicated DBMS server. Galera Cluster then replicates the data through to the cluster, ensuring it remains synchronous.

This solution is simple and easy to manage, especially if you can install the whole stack of each node on one physical machine. The direct connection from the application tier to the data tier ensures low latency.

There are, however, certain disadvantages to whole stack clustering that you should consider:

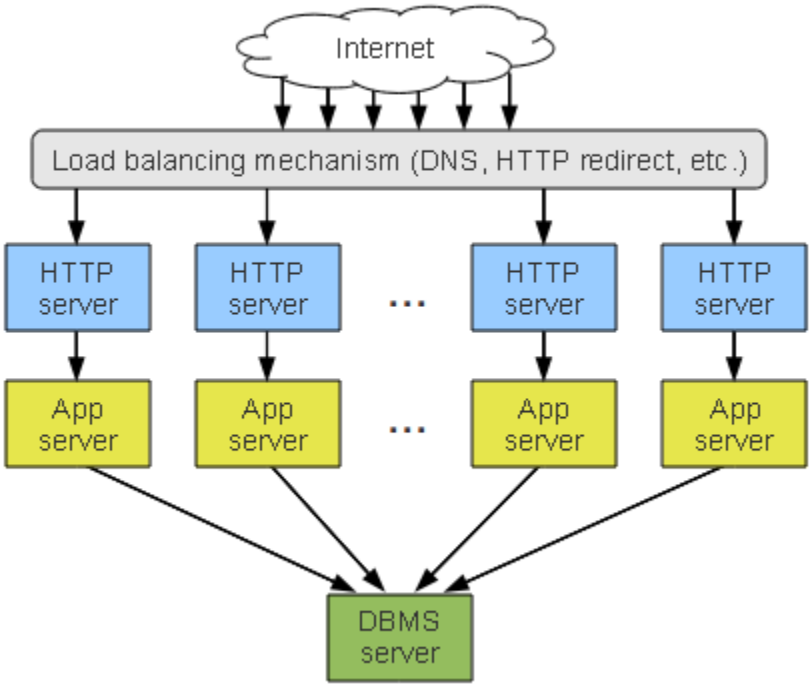


Fig. 1: *No Clustering*

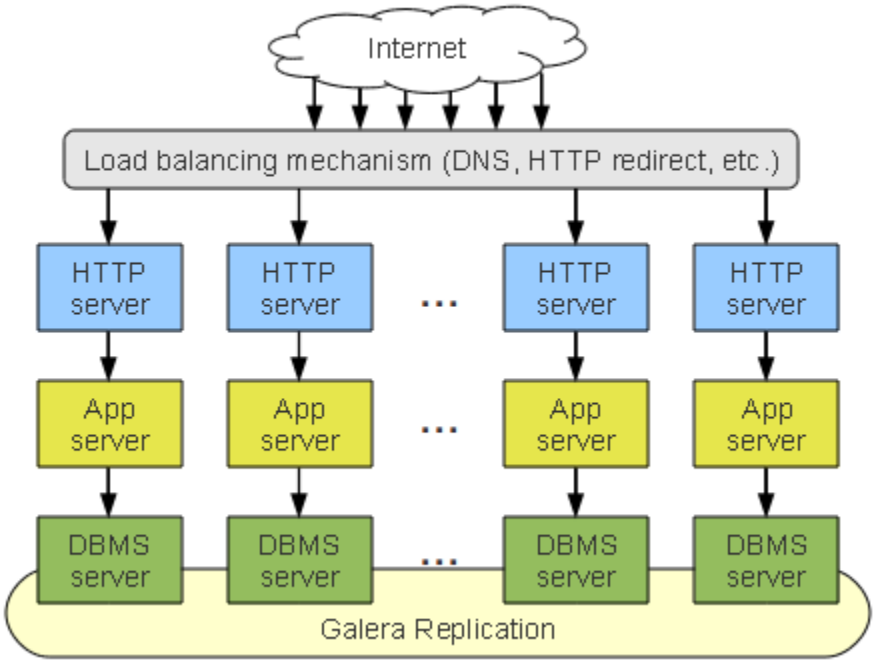


Fig. 2: *Whole Stack Cluster*

- **Lack of Redundancy**: When the database server fails, the whole stack fails. This is because the application server uses a dedicated database server. If the database server fails there's no alternative for the application server, so the whole stack goes down.
- **Inefficient Resource Usage**: A dedicated DBMS server for each application server will be overused. This is poor resource consolidation. For instance, one server with a 7 GB buffer pool is much faster than two servers with 4 GB buffer pools.
- **Increased Unproductive Overhead**: Each server reproduces the work of the other servers in the cluster. This redundancy is a drain on the server's resources.
- **Increased Rollback Rate**: Given that each application server writes to a dedicated database server, cluster-wide conflicts are more likely. This can increase the likelihood of corrective rollbacks.
- **Inflexibility**: There is no way for you to limit the number of primary nodes or to perform intelligent load balancing.

Despite the disadvantages, however, this setup can prove very usable for several applications, depending on your needs.

Data Tier Clustering

To compensate for the shortcomings in whole stack clusters, you can cluster the data tier separately from your web and application servers.

With data tier clustering, the DBMS servers form a cluster distinct from your *n*-tier application cluster. The application servers treat the database cluster as a single virtual server, making calls through load balancers to the data tier.

In a data tier cluster, the failure of one node does not effect the rest of the cluster. Furthermore, resources are consolidated better and the setup is flexible. That is to say, you can assign nodes different roles using intelligent load balancing.

There are, however, certain disadvantages to consider in data tier clustering:

- **Complex Structure**: Since load balancers are involved, you must back them up in case of failure. This typically means that you have two more servers than you would otherwise, as well as a failover solution between them.
- **Complex Management**: You need to configure and reconfigure the load balancers whenever a DBMS server is added to the cluster or removed.
- **Indirect Connections**: The load balancers between the application cluster and the data tier cluster increase the latency for each query. As a result, this can easily become a performance bottleneck. You will need powerful load balancing servers to avoid this.
- **Scalability**: This setup does not scale well over several datacenters. Attempts to do so may reduce any benefits you gain from resource consolidation, given that each datacenter must include at least two DBMS servers.

Data Tier Clustering with Distributed Load Balancing

One solution to the limitations of data tier clustering is to deploy them with distributed load balancing. This method roughly follows the standard data tier cluster method, but includes a dedicated load balancer installed on each application server.

In this deployment, the load balancer is no longer a single point of failure. Furthermore, the load balancer scales with the application cluster and thus is unlikely to become a bottleneck. Additionally, it minimizes the client-server communications latency.

Data tier clustering with distributed load balancing has the following disadvantage:

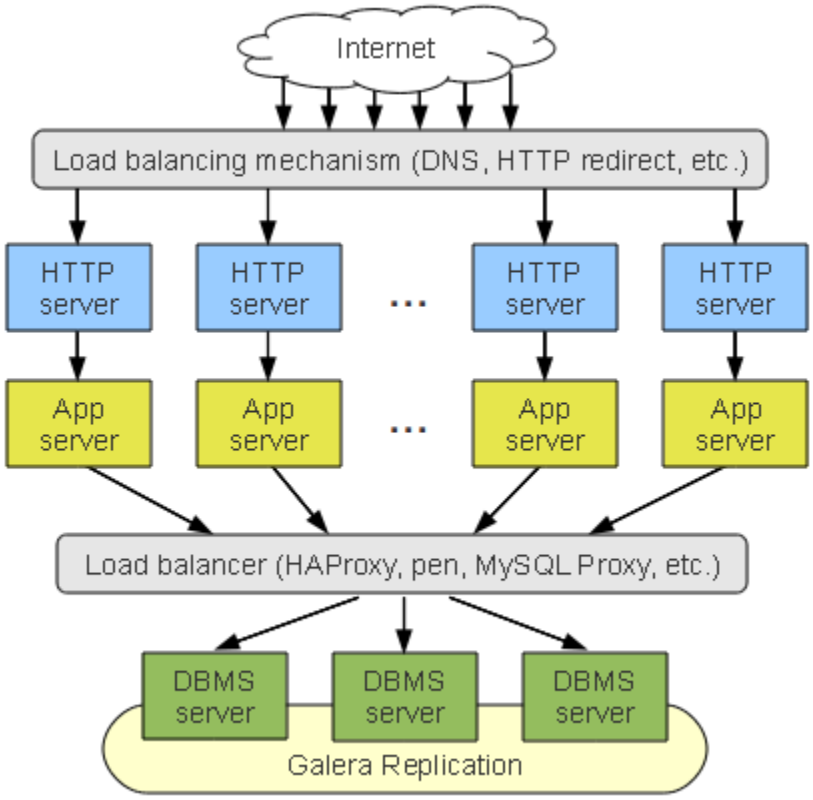


Fig. 3: *Data Tier Clustering*

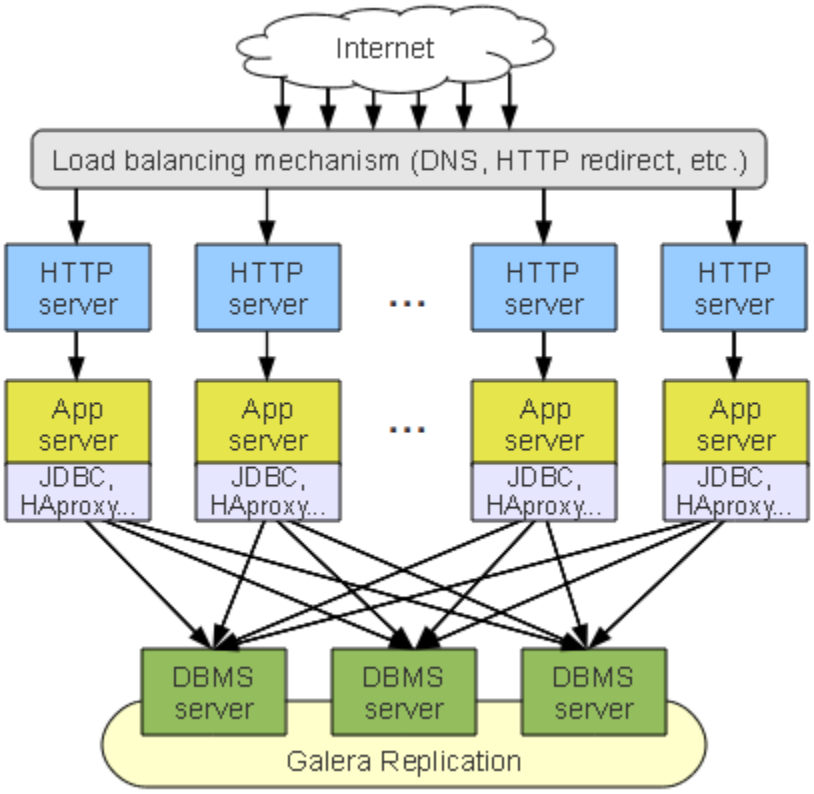


Fig. 4: *Data Tier Cluster with Distributed Load Balancing*

- **Complex Management**: Each application server deployed for an *n*-tier application cluster will require another load balancer that you need to set up, manage and reconfigure whenever you change or otherwise update the database cluster configuring.

Aggregated Stack Clustering

Besides the deployment methods already mentioned, you could set up a hybrid method that integrates whole stack and data tier clustering by aggregating several application stacks around single DBMS servers.

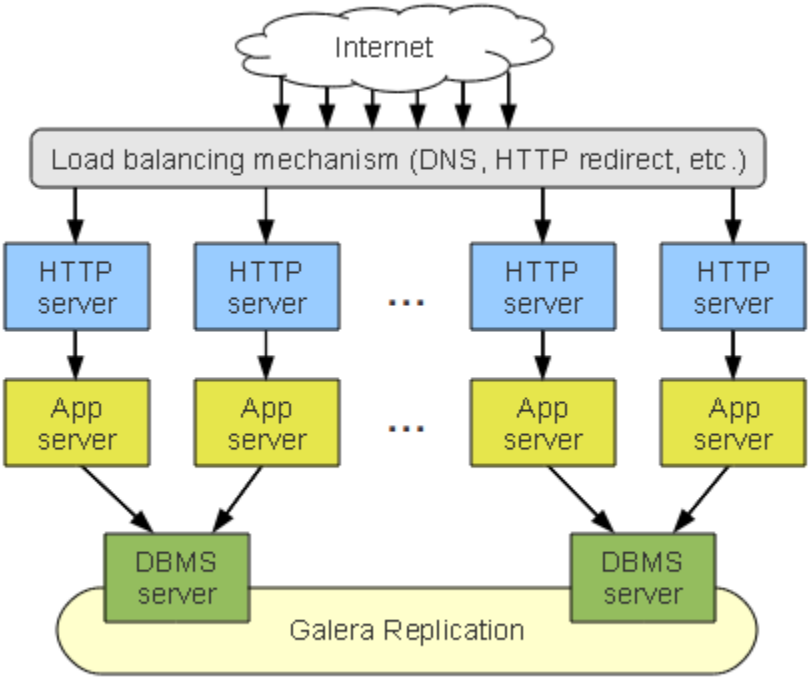


Fig. 5: *Aggregated Stack Clustering*

This layout improves on the resource utilization of the whole stack cluster, while maintaining its relative simplicity and direct DBMS connection benefits. It is also how a data tier cluster with distributed load balancing will look if you were to use only one DBMS server per datacenter.

The aggregated stack cluster is a good setup for sites that are not very large, but are hosted at more than one datacenter.

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ

- search

Related Documents

- *Deployment Variants* (page 112)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

5.2 Load Balancing

Galera Cluster guarantees node consistency regardless of where and when the query is issued. In other words, you are free to choose a load-balancing approach that best suits your purposes. If you decide to place the load balancing mechanism between the database and the application, you can consider, for example, the following tools:

- **HAProxy** an open source TCP/HTTP load balancer.
- **Pen** another open source TCP/HTTP load balancer. Pen performs better than HAProxy on SQL traffic.
- **Galera Load Balancer** inspired by Pen, but is limited to balancing generic TCP connections only.

For more information or ideas on where to use load balancers in your infrastructure, see *Cluster Deployment Variants* (page 112).

Related Documents

- *Deployment Variants* (page 112)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search
- Home
- *Docs* (page 1)
- KB
- Training

- FAQ

5.2.1 HAProxy

High Availability Proxy, or HAProxy is a single-threaded event-driven non-blocking engine that combines a fast I/O layer with a priority-based scheduler. You can use it to balance TCP connections between application servers and Galera Cluster.

Installation

HAProxy is available in the software repositories of most Linux distributions and it is the ports tree of FreeBSD. You can install it using the appropriate package manager.

- For DEB-based Linux distributions (for example, Debian and Ubuntu), run the following from the command-line:

```
# apt-get install haproxy
```

- For RPM-based Linux distributions (for example, Red Hat Enterprise Linux and CentOS), execute the following from the command-line:

```
# yum install haproxy
```

- For FreeBSD and similar operating systems, HAProxy is available in the ports tree at */usr/ports/net/haproxy*. Alternatively, you can install it using the package manager like so:

```
# pkg install net/haproxy
```

Whichever method you use, it installs HAProxy on your server. In the event that the command for your Linux distribution or operating system does not work as expected, check your system's documentation or software repository for the correct procedure to install HAProxy.

Configuration

Configuration options for HAProxy are managed through an `haproxy.cfg` configuration file. The above package installations generally put this file in the `/etc/haproxy/` directory. However, it may have a different path depending on your operating system distribution.

To configure HAProxy to work with Galera Cluster, add the lines to the `haproxy.cfg` configuration file similar to the following:

```
# Load Balancing for Galera Cluster
listen galera 192.168.1.10:3306
     balance source
     mode tcp
     option tcpka
     option mysql-check user haproxy
     server node1 192.168.1.1:3306 check weight 1
     server node2 192.168.1.2:3306 check weight 1
     server node3 192.168.1.3:3306 check weight 1
```

You will create the proxy for Galera Cluster using the `listen` parameter. This gives HAProxy an arbitrary name for the proxy and defines the IP address and port you want it to listen on for incoming connections. Under this parameter, indent and define a series of options to tell HAProxy what you want it to do with these connections.

- `balance` defines the destination selection policy HAProxy should use in choosing which server it routes incoming connections.
- `mode tcp` defines the type of connections it should route. Galera Cluster uses TCP connections.
- `option tcpka` enables the keepalive function to maintain TCP connections.
- `option mysql-check user <username>` enables a database server check to determine whether the node is currently operational.
- `server <server-name> <IP_address> check weight 1` defines the nodes HAProxy should use in routing connections.

Destination Selection Policies

When HAProxy receives a new connection, there are a number of options available to define which algorithm it uses to choose where to route the connection. This algorithm is its destination selection policy. It is defined by the `balance` parameter.

- **Round Robin** directs new connections to the next destination in a circular order list, modified by the server's weight. Enable it with `balance roundrobin`.
- **Static Round Robin** directs new connections to the next destination in a circular order list, modified by the server's weight. Unlike the standard implementation of round robin, in static round robin you can't modify the server weight on the fly. Changing the server weight requires you to restart HAProxy. Enable it with `balance static-rr`.
- **Least Connected** directs new connections to the server with the smallest number of connections available, which is adjuted for the server's weight. Enable it with `balance leastconn`.
- **First** directs new connections to the first server with a connection slot available. They are chosen from the lowest numeric identifier to the highest. Once the server reaches its maximum connections value, HAProxy moves to the next in the list. Enable it with `balance first`.
- **Source Tracking** divides the source IP address by the total weight of running servers. Ensures that client connections from the same source IP always reach the same server. Enable it with `balance source`.

In the above configuration example, HAProxy is configured to use the source selection policy. For your implementation, choose the policy that works best with your infrastructure and load.

Enabling Database Server Checks

In addition to routing TCP connections to Galera Cluster, HAProxy can also perform basic health checks on the database server. When enabled, HAProxy attempts to establish a connection with the node and parses its response, or any errors, to determine if the node is operational.

For HAProxy, you can enable this through the `mysql-check` option. However, it requires that you also create a user in the cluster for HAProxy to use when connecting.

```
CREATE USER 'haproxy'@'192.168.1.10';
```

Make the user name the same as given in the `haproxy.cfg` configuration file for the `mysql-check` option. Replace the IP address with that of the server that runs HAProxy.

Using HAProxy

When you finish configuring HAProxy and the nodes to work with HAProxy, you can start it on the server. For servers that use `init`, run the following command:

```
# service haproxy start
```

For servers that use `systemd`, run instead this command:

```
# systemctl start haproxy
```

After doing this, the server will be running HAProxy. When new connections are made to this server, it routes them through to nodes in the cluster.

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

5.2.2 Pen Load Balancer

Pen is a high-scalability, high-availability, robust load balancer for TCP- and UDP-based protocols. You can use it to balance connections between application servers and Galera Cluster.

Installation

Pen is available in the software repositories of most Linux distributions. You can install it using a package manager.

- For DEB-based Linux distributions (that is, Debian and Ubuntu), run the following from the command-line:

  ```
  # apt-get install pen
  ```

- For RPM-based Linux distributions (that is, Red Hat Enterprise Linux and CentOS), use the `yum` utility instead by executing the following from the command-line:

  ```
  # yum install pen
  ```

Whichever you use, they will install Pen on your system. In the event that the command for your distribution or operating system does not work as expected, check your system's documentation or software repository for information on the correct procedure to install Pen. For instance, on a RPM-based system, you may have to install the `yum` utility.

Using Pen

Once you've installed Pen on the load balancing server, you can launch it from the command-line by entering something like the following:

```
# pen -l pen.log -p pen.pid localhost:3306 \
      191.168.1.1:3306 \
      191.168.1.2:3306 \
      191.168.1.3:3306
```

When one of the application servers attempts to connect to the Pen server on port `3306`, Pen routes that connection to one of the Galera Cluster nodes.

For more information on Pen configuration and use, see its manpage.

Server Selection

When Pen receives a new connection from the application servers, it first checks to see where the application was routed on the last connection and attempts to send traffic there. In the event that it cannot establish a connection, it falls back on a round-robin selection policy.

There are a number of options you can use to modify this behavior when you launch Pen.

- **Default Round Robin:** This directs all new connections to the next destination in a cirular order, without determining which server a client used the last time. You can enable this with the `-r` option.
- **Stubborn Selection:** In the event that the initial choice is unavailable, Pen closes the client connection. This is enabled with the `-s` option.
- **Hash Client IP Address:** Pen applies a hash on the client IP address for the initial server selection, making it more predictable where it routes client connections in the future.

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Using Galera Load Balancer* (page 125)
- *LISTEN_ADDR* (page 344)
- *DEFAULT_TARGETS* (page 344)
- *OTHER_OPTIONS* (page 344)
- *–round* (page 349)
- *–single* (page 350)
- *–random* (page 349)

- *–source* (page 350)
- *Service Installation* (page 124)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

5.2.3 Galera Load Balancer (Galera Load Balancer binaries are part of Galera Cluster Enterprise Edition)

Galera Load Balancer provides simple TCP connection balancing. It was developed with scalability and performance in mind. It draws on Pen for inspiration, but its functionality is limited to only balancing TCP connections. It provides several features:

- Support for configuring back-end servers at runtime.
- Support for draining servers.
- Support for the epoll API for routing performance.
- Support for multithreaded operations.
- Optional watchdog module to monitor destinations and adjust the routing table.

Installation

Unlike Galera Cluster, there is no binary installation available for Galera Load Balancer. Installing it on your system will require you to build it from the source files. They're available on GitHub at glb.

To build Galera Load Balancer, you will need to complete a few steps. First, from a directory convenient for source builds (for example, `/opt`), use the `git` utility to clone the GitHub repository for Galera Load Balancer. You would do this like so:

```
$ git clone https://github.com/codership/glb
```

Next, from within `glb` directory created by `git`, run the bootstrap script–which will be found in that directory.

```
$ cd glb/
$ ./bootstrap.sh
```

Now you will need to configure `make` to build on your system, then run `make` to build the application. After that, you will use `make` to install it. This may seem like a lot, but it is simple. Just execute the following lines, one at a time, from the command-line:

```
$ ./configure

$ make

# make install
```

Note: Galera Load Balancer installs in the `/usr/sbin` directory. So you will need to run the last line above as root.

Once you've successfully execute everything above, Galera Load Balancer will be installed on your system. You can launch it from the command-line, using the `glbd` command.

In addition to the system daemon, you will also have installed `libglb`, a shared library for connection balancing with any Linux applications that use the `connect()` call from the C Standard Library.

Service Installation

The above installation procedure only installs Galera Load Balancer to be run manually from the command-line. However, you may find it more useful to run this application as a system service. To do this, you will need to copy a couple of files to the appropriate directories.

In the source directory you cloned from GitHub, navigate into the `files` directory. Within that directory there is a configuration file and a service script that you need to copy to their relevant locations.

First, copy `glbd.sh` into `/etc/init.d` directory under a service name. You would execute the following from the command-line to do this:

```
# cp glbd.sh /etc/init.d/glb
```

Now, copy the default `glbd.cfg` file into the appropriate configuration directory. For Red Hat and its derivatives, this is `/etc/sysconfig/glbd.cfg`. For Debian and its derivatives, use `/etc/default/glbd.cfg`. For the former possibility, you would execute this from the command-line:

```
# cp glbd.cfg /etc/sysconfig/glbd.cfg
```

When you finish this, you will be able to manage Galera Load Balancer through the `service` command. For more information on available commands, see *Using Galera Load Balancer* (page 125).

Configuration

When you run Galera Load Balancer, you can configure its use through the command-line options. You can get a list of by exeduting `glb` with the `--help` option. For servers running Galera Load Balancer as a service, you can manage it through the `glbd.cfg` configuration file.

- *LISTEN_ADDR* (page 344): This is the address that Galera Load Balancer monitors for incoming client connections.
- *DEFAULT_TARGETS* (page 344): This specifies the default servers where Galera Load Balancer is to route incoming client connections. For this parameter, use the IP addresses for the nodes in your cluster.
- *OTHER_OPTIONS* (page 344): This is used to define additional Galera Load Balancer options. For example, you might want to set the balancing policy. Use the same format as you would from the command-line.

Below is an example of a *glbd.cfg'* configuration file:

```
# Galera Load Balancer Configuration
LISTEN_ADDR="8010"
DEFAULT_TARGETS="192.168.1.1 192.168.1.2 192.168.1.3"
OTHER_OPTIONS="--random --top 3"
```

The `glbd.cfg` configuration file would be the one you copied into `/etc` as mentioned in the previous section.

Destination Selection Policies

Galera Load Balancer–both the system daemon and the shared library–supports five destination selection policies. When you run it from the command-line, you can define these using the command-line arguments. Otherwise, you will have to add the arguments to the *OTHER_OPTIONS* (page 344) parameter in the `glbd.cfg` configuration file.

- **Least Connected**: This directs new connections to the server using the smallest number of connections possible. It will be adjusted for the server weight. This is the default policy.
- **Round Robin**: This sets new connections to the next destination in the circular order list. You can enable it with the *–round* (page 349) option.
- **Single**: This directs all connections to the single server with the highest weight of those available. Routing continues to that server until it fails, or until a server with a higher weight becomes available. You can enable it with the *–single* (page 350) option.
- **Random**: This will direct connections randomly to available servers. You can enable it using the *–random* (page 349) option.
- **Source Tracking**: This will direct connections originating from the same address to the same server. You can enable it with the *–source* (page 350) option.

Using Galera Load Balancer

The section on *Service Installation* (page 124) explained how to configure a system to run Galera Load Balancer as a service. If you do that, you can then manage common operations with the `service` command. The format for doing this is to enter `service`, followed by `glb`, and then an option.

Below is an example of how you might use `service` to get information on the Galera Load Balancer:

```
# service glb getinfo

Router:
-----------------------------------------
     Address       : weight   usage  cons
  192.168.1.1:4444 : 1.000    0.000    0
  192.168.1.2:4444 : 1.000    0.000    0
  192.168.1.3:4444 : 1.000    0.000    0
-----------------------------------------
Destinations: 3, total connections: 0
```

In the results shown here, you can see a list of servers available, their weight and usage, as well as the number of connections made to them.

The `service` script supports several operations. Below is a list of them and their uses:

- `start` is used to start `glb`, the Galera Load Balancer.
- `stop` will stop Galera Load Balancer.
- `restart` tells `glb` to stop and restart the Galera Load Balancer.
- `getinfo` is used as shown in the example above to retrieve the current routing information.
- `getstats` will provide performance statistics related to the cluster.
- `add <IP Address>` can be used to add an IP address from the routing table.
- `remove <IP Address>` will remove the designated IP address from the routing table.

- `drain <IP Address>` will sets the designated server to drain. When doing this, Galera Load Balancer won't send new connections to the given server, but it also won't kill existing connections. Instead, it waits for the connections to the specified server to end gracefully.

When adding an IP address to Galera Load Balancer at runtime, keep in mind that it must follow the convention, `IP Address:port:weight`. A hostname may be used instead of an IP address.

Note: Contact Codership sales at sales@galeracluster.com for more information, and to get Galera Load Balancer binaries and the Galera Cluster Enterprise Edition software.

Related Documents

- *Using Galera Load Balancer* (page 125)
- *LISTEN_ADDR* (page 344)
- *DEFAULT_TARGETS* (page 344)
- *OTHER_OPTIONS* (page 344)
- *–round* (page 349)
- *–single* (page 350)
- *–random* (page 349)
- *–source* (page 350)
- *Service Installation* (page 124)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *wsrep_node_address* (page 248)
- *wsrep_node_name* (page 250)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

5.3 Container Deployments

In the standard deployment methods for Galera Cluster, a node runs on a server in the same manner as would an individual stand-alone instance of MySQL or MariaDB. In container deployments, a node runs in a containerized virtual environment on the server.

You may find these methods useful in portable deployments across numerous machines, testing applications that depend on Galera Cluster, process isolation for security, or scripting the installation and configuration process.

The configuration for a node running in a containerized environment remains primarily the same as a node running in the standard manner. However, there are some parameters that draw their defaults from the base system configurations. You will need to set these, manually. Otherwise, the jail will be unable to access the host file system.

- *wsrep_node_address* (page 248): A node determines the default address from the IP address on the first network interface. Jails cannot see the network interfaces on the host system. You need to set this parameter to ensure that the cluster is given the correct IP address for the node.
- *wsrep_node_name* (page 250): The node determines the default name from the system hostname. Jails have their own hostnames, distinct from that of the host system.

Bear in mind that the configuration file must be placed within the container `/etc` directory, not that of the host system.

Related Documents

- *wsrep_node_address* (page 248)
- *wsrep_node_name* (page 250)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Firewall Settings* (page 208)
- *wsrep_node_address* (page 248)
- *wsrep_node_name* (page 250)

Related Articles

- Starting a Cluster
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

5.3.1 Using Docker

Docker provides an open source platform for automatically deploying applications within software containers.

Galera Cluster can run from within a such a container, within Docker. You may find containers useful in portable deployment across numerous machines, testing applications that depend on Galera Cluster, or scripting the installation and configuration process.

Note: This guide assumes that you are only running one container node per server. For more information on running multiple nodes per server, see *Getting Started Galera with Docker, 'Part I <https://galeracluster.com/2015/05/getting-started-galera-with-docker-part-1/>'_ and 'Part II <https://galeracluster.com/2015/05/getting-started-galera-with-docker-part-2-2/>'_.*

Configuring a Container

Images are the containers that Docker has available to run. There are a number of base images available through Docker Hub. You can pull these to your system through the `docker` command-line tool. You can also build new images.

When Docker builds a new image, it sources a `Dockerfile` to determine the steps that it needs to take in order to generate the image you want to use. This means you can script the installation and configuration process. Basically, such a script would need to load the needed configuration files, run updates, and install packages when the image is built - all through a single command. Below are examples of how you might write such a script.

For Galera Cluster 8.0:

```
# Galera Cluster Dockerfile
FROM ubuntu:14.04
MAINTAINER your name <your.user@example.org>

ENV DEBIAN_FRONTEND noninteractive

RUN apt-get update
RUN apt-get install -y  software-properties-common
RUN apt-key adv --keyserver keyserver.ubuntu.com --recv 8DA84635
RUN add-apt-repository 'deb https://releases.galeracluster.com/galera-4/ubuntu trusty␣
↪main'
RUN add-apt-repository 'deb https://releases.galeracluster.com/mysql-wsrep-8.0/ubuntu␣
↪trusty main'


RUN apt-get update
RUN apt-get install -y galera-4 galera-arbitrator-4 mysql-wsrep-8.0 rsync
RUN apt-get install -y galera-4 galera-arbitrator-4 mysql-wsrep-8.0 rsync

COPY my.cnf /etc/mysql/my.cnf
ENTRYPOINT ["mysqld"]
```

For Galera Cluster 8.4:

```
# Galera Cluster Dockerfile
FROM ubuntu:14.04
MAINTAINER your name <your.user@example.org>

ENV DEBIAN_FRONTEND noninteractive
```

(continues on next page)

(continued from previous page)

```
RUN apt-get update
RUN apt-get install -y  software-properties-common
RUN apt-key adv --keyserver keyserver.ubuntu.com --recv 8DA84635
RUN add-apt-repository 'deb https://releases.galeracluster.com/galera-4/ubuntu trusty␣
↪main'
RUN add-apt-repository 'deb https://releases.galeracluster.com/mysql-wsrep-8.4/ubuntu␣
↪trusty main'


RUN apt-get update
RUN apt-get install -y galera-4 galera-arbitrator-4 mysql-wsrep-8.4 rsync
RUN apt-get install -y galera-4 galera-arbitrator-4 mysql-wsrep-8.4 rsync

COPY my.cnf /etc/mysql/my.cnf
ENTRYPOINT ["mysqld"]
```

Note that for packages before MySQL 5.7.44 and 8.0.35, the signing key is BC19DDBA.

This example follows the installation process for running Galera Cluster from within a Docker container running on Ubuntu. When you run the build command, Docker pulls down the Ubuntu 14.04 image from Docker Hub, if it is needed. It then runs each command in the `Dockerfile` to initialize the image for your use.

Configuration File

Before you build the container, you need to create the configuration file for the node. The `COPY` command in the `Dockerfile` example above copies `my.cnf`, the MySQL configuration file, from the build directory into the container.

For the most part, the configuration file for a node running within Docker is the same as when the node is running on a standard Linux server. However, there are some parameters that may not be included in the MySQL configuration file and instead use the default values from the underlying database system—or they may have been set manually, on-the-fly using the `SET` statement. For these parameters, since Docker can't access the host system, you may need to set them, manually.

- *wsrep_node_address* (page 248): A node will determine the default address from the IP address on the first network interface. Containers cannot see the network interfaces on the host system. Therefore, you will need to set this parameter to ensure the cluster is given the correct IP address for the node.
- *wsrep_node_name* (page 250): A node will determine the default host name from the system hostname. Containers have their own hostnames distinct from the host system.

Changes to the `my.cnf` file will not propagate into an existing container. Therefore, whenever you make changes to the configuration file, run the build again to create a new image with the updated configuration file. Docker caches each step of the build and only runs those steps that have changed when rebuilding. For example, using the `Dockerfile` example above, if you rebuild an image after changing `my.cnf`, Docker will run only the last two steps.

Note: If you need Docker to rerun the entire build, use the `--force-rm=true` option.

Building a Container Image

Building an image simplifies everyting—the node installation, the configuration and the deployment process—by reducing it to a single command. It will create a server instance where Galera Cluster is already installed, configured and ready to start.

You can build a container node using the `docker` command-line tool like so:

```
# docker build -t ubuntu:galera-node1 ./
```

When this command runs, Docker looks in the current working directory, (that is, `./`), for the `Dockerfile`. It then follows each command in the `Dockerfile` to build the image. When the build is complete, you can view the addition among the available images by executing the following:

```
# docker images

REPOSITORY  TAG           IMAGE ID      CREATED         SIZE
ubuntu      galera-node-1 53b97c3d7740  2 minutes ago   362.7 MB
ubuntu      14.04         ded7cd95e059  5 weeks ago     185.5 MB
```

You can see in the results here that there is a working node image available for use as a container. You would launch it executing `docker run` at the command-line. You would repeat the build process on each server to create a node container image for Galera Cluster.

You would then update the container tag to help differentiate between each node by executing something like this:

```
[root@node2]# docker build -t ubuntu:galera-node2 ./
[root@node3]# docker build -t ubuntu:galera-node3 ./
```

Deploying a Container

When you finish building an image, you are ready to launch the node container. For each node, start the container using the Docker command-line tool with the `run` argument like so:

```
# docker run -i -d --name Node1 --host node1 \
      -p 3306:3306 -p 4567:4567 -p 4568:4568 -p 4444:4444 \
      -v /var/container_data/mysql:/var/lib/mysql \
      ubuntu:galera-node1
```

In this example, Docker launches a pre-built Ubuntu container tagged as `galera-node1`, which was built using the example `Dockerfile` from above. The `ENTRYPOINT` parameter is set to `/bin/mysqld` so that the container launches the database server when starting. You would modify the `--name` option in the example here for each node container you start.

You'll notice in the example here there are several `-p` options included. Those are described in the next section on Firewall Settings. The `-v` option is described in the section after it on Persistent Data.

Note: The above command starts a container node meant to be attached to an existing cluster. If you are starting the first node in a cluster, use the `mysqld_bootstrap` command. For more information, see Starting a Cluster.

Firewall Settings

When you launch the Docker container (that is, `docker run` as shown above), the series of `-p` options connect the ports on the host system to those in the container. When the container is launched this way, nodes in the container have the same level of access to the network as the node would if it were running on the host system.

Use these settings, though, when you run only one container on the server. If you are running multiple containers on the server, you will need a load balancer to handle and direct incoming connections to individual nodes.

For more information on configuring the firewall for Galera Cluster, see *Firewall Settings* (page 208).

Persistent Data

Docker containers are not meant to carry persistent data. When you close a container, the data it carries is lost. To avoid this problem, you can link volumes in the container to directories on the host file system. This is done with the `-v` option when you launch the container.

In the launch example above (that is, the `docker run` lines), the `-v` argument connects the `/var/container_data/mysql/` directory to `/var/lib/mysql/` in the container. This replaces the local datadir inside the container with a symbolic link to a directory on the host system. This ensures that you won't lose data when the container restarts.

Database Client

Once you have a container node running, you can execute additional commands on the container using the `docker exec` command with the container name given with the `--name` parameter.

Using the example above, if you want access to the database client, you would run the following command:

```
# docker exec -ti Node1 /bin/mysql -u root -p
```

Notice here that `Node1` is the name given with the `--name` parameter in the example earlier.

Related Documents

- *Firewall Settings* (page 208)
- *wsrep_node_address* (page 248)
- *wsrep_node_name* (page 250)

Related Articles

- Starting a Cluster

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Firewall Configuration with PF* (page 213)
- *Galera Cluster for MySQL* (page 43)
- *MariaDB Galera Cluster* (page 50)
- *wsrep_node_name* (page 250)
- *wsrep_provider* (page 254)
- *wsrep_node_address* (page 248)

- *wsrep_node_name* (page 250)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

5.3.2 Using Jails

In FreeBSD, `jails` provides a platform for securely deploying applications within virtual instances. You may find it useful in portable deployments across numerous machines for testing and security.

Galera Cluster can run from within a jail instance.

Preparing the Server

Jails exist as isolated file systems within, but unaware of, the host server. In order to grant the node running within the jail network connectivity with the cluster, you need to configure the network interfaces and firewall to redirect from the host into the jail.

Network Configuration

To begin, create a second loopback interface for the jail. this allows you to isolate jail traffic from `lo0`, the host loopback interface.

Note: For the purposes of this guide, the jail loopback is called `lo1`, if `lo1` already exists on your system, increment the digit to create one that does not already exist, (for instance, `lo2`).

To create a loopback interface, complete the following steps:

1. Using your preferred text editor, add the loopback interface to `/etc/rc.conf`:

   ```
   # Network Interface
   cloned_interfaces="${cloned_interfaces} lo1"
   ```

2. Create the loopback interface:

   ```
   # service netif cloneup
   ```

This creates `lo1`, a new loopback network interface for your jails. You can view the new interface in the listing using the following command:

```
$ ifconfig
```

Firewall Configuration

FreeBSD provides packet filtering support at the kernel level. Using PF you can set up, maintain and inspect the packet filtering rule sets. For jails, you can route traffic from external ports on the host system to internal ports within the

jail's file system. This allows the node running within the jail to have network access as though it were running on the host system.

To enable PF and create rules for the node, complete the following steps:

1. Using your preferred text editor, make the following additions to `/etc/rc.conf`:

```
# Firewall Configuration
pf_enable="YES"
pf_rules="/etc/pf.conf"
pflog_enable="YES"
pflog_logfile="/var/log/pf.log"
```

2. Create the rules files for PF at `/etc/pf.conf`

```
# External Network Interface
ext_if="vtnet0"

# Internal Network Interface
int_if="lo1"

# IP Addresses
external_addr="host_IP_address"
internal_addr="jail_IP_address_range"

# Variables for Galera Cluster
wsrep_ports="{3306,4567,4568,4444}"
table <wsrep_cluster_address> persist {192.168.1.1,192.168.1.2,192.168.1.3}

# Translation
nat on $ext_if from $internal_addr to any -> ($ext_if)

# Redirects
rdr on $ext_if proto tcp from any to $external_addr/32 port 3306 -> jail_IP_
 ↪address port 3306
rdr on $ext_if proto tcp from any to $external_addr/32 port 4567 -> jail_IP_
 ↪address port 4567
rdr on $ext_if proto tcp from any to $external_addr/32 port 4568 -> jail_IP_
 ↪address port 4568
rdr on $ext_if proto tcp from any to $external_addr/32 port 4444 -> jail_IP_
 ↪address port 4444

pass in proto tcp from <wsrep_cluster_address> to any port $wsrep_ports keep state
```

   Replace `host_IP_address` with the IP address of the host server and `jail_IP_address` with the IP address you want to use for the jail.

3. Using `pfctl`, check for any typos in your PF configurations:

```
# pfctl -v -nf /etc/pf.conf
```

4. If `pfctl` runs without throwing any errors, start PF and PF logging services:

```
# service pf start
# service pflog start
```

The server now uses PF to manage its firewall. Network traffic directed at the four ports Galera Cluster uses is routed to the comparable ports within the jail.

For more information on firewall configurations for FreeBSD, see *Firewall Configuration with PF* (page 213).

Creating the Node Jail

While FreeBSD does provide a manual interface for creating and managing jails on your server, (`jail(8)`), it can prove cumbersome in the event that you have multiple jails running on a server.

The application `ezjail` facilitates this process by automating common tasks and using templates and symbolic links to reduce the disk space usage per jail. It is available for installation through `pkg`. Alternative, you can build it through ports at `sysutils/ezjail`.

To create a node jail with `ezjail`, complete the following steps:

1. Using your preferred text editor, add the following line to `/etc/rc.conf`:

   ```
   ezjail_enable="YES"
   ```

   This allows you to start and stop jails through the `service` command.

2. Initialize the `ezjail` environment:

   ```
   # ezjail-admin install -sp
   ```

   This install the base jail system at `/usr/jails/`. It also installs a local build of the ports tree within the jail.

   Note: While the database server is not available for FreeBSD in ports or as a package binary, a port of the *Galera Replication Plugin* is available at `databases/galera`.

   ---

3. Create the node jail.

   ```
   # ezjail-admin create galera-node 'lo1|192.168.68.1'
   ```

   This creates the particular jail for your node and links it to the `lo1` loopback interface and IP address. Replace the IP address with the local IP for internal use on your server. It is the same address as you assigned in the firewall redirects above for `/etc/pf.conf`.

   Note: Bear in mind that in the above command `galera-node` provides the hostname for the jail file system. As Galera Cluster draws on the hostname for the default node name, you need to either use a unique jail name for each node, or manually set *wsrep_node_name* (page 250) in the configuration file to avoid confusion.

   ---

4. Copy the `resolve.conf` file from the host file system into the node jail.

   ```
   # cp /etc/resolv.conf /usr/jails/galera-node/etc/
   ```

   This allows the network interface within the jail to resolve domain names in connecting to the internet.

5. Start the node jail.

   ```
   # ezjail-admin start galera-node
   ```

The node jail is now running on your server. You can view running jails using the `ezjail-admin` command:

```
# ezjail-admin list
STA JID  IP             Hostname     Root Directory
--- ---- -------------- ------------ ----------------------
DR  2    192.168.68.1   galera-node  /usr/jails/galera-node
```

While on the host system, you can access and manipulate files and directories in the jail file system from `/usr/jails/galera-node/`. Additionally, you can enter the jail directly and manipulate processes running within using the following command:

```
root@FreeBSDHost:/usr/jails # ezjail-admin console galera-node
root@galera-node:~ #
```

When you enter the jail file system, note that the hostname changes to indicate the transition.

Installing Galera Cluster

Regardless of whether you are on the host system or working from within a jail, you can install Galera Cluster on FreeBSD from a binary package, or build the database server from source code.

The specific build process that you need to follow depends on the database server that you want to use:

- *Galera Cluster for MySQL* (page 43)
- *MariaDB Galera Cluster* (page 50)

Due to certain Linux dependencies, the *Galera Replication Plugin* cannot be built from source on FreeBSD. Instead you can use the port at `/usr/ports/databases/galera` or install it from a binary package within the jail:

```
# pkg install galera
```

This install the wsrep Provider file in `/usr/local/lib`. Use this path in the configuration file for the *wsrep_provider* (page 254) parameter.

Configuration File

For the most part, the configuration file for a node running in a jail is the same as when the node runs on a standard FreeBSD server. But, there are some parameters that draw their defaults from the base system. These you need to set manually, as the jail is unable to access the host file system.

- *wsrep_node_address* (page 248) The node determines the default address from the IP address on the first network interface. Jails cannot see the network interfaces on the host system. You need to set this parameter to ensure that the cluster is given the correct IP address for the node.
- *wsrep_node_name* (page 250) The node determines the default name from the system hostname. Jails have their own hostnames, distinct from that of the host system.

```
[mysqld]
user=mysql
#bind-address=0.0.0.0

# Cluster Options
wsrep_provider=/usr/lib/libgalera_smm.so
wsrep_cluster_address="gcomm://192.168.1.1, 192.168.1.2, 192.16.1.3"
wsrep_node_address="192.168.1.1"
wsrep_node_name="node1"
wsrep_cluster_name="example_cluster"

# InnoDB Options
default_storage_engine=innodb
innodb_autoinc_lock_mode=2
innodb_flush_log_at_trx_commit=0
```

(continues on next page)

(continued from previous page)

```
# SST
wsrep_sst_method=rsync
```

If you are logged into the jail console, place the configuration file at `/etc/my.cnf`. If you are on the host system console, place it at `/usr/jails/galera-node/etc/my.cnf`. Replace `galera-node` in the latter with the name of the node jail.

Starting the Cluster

When running the cluster from within jails, you create and manage the cluster in the same manner as you would in the standard deployment of Galera Cluster on FreeBSD. The exception being that you must obtain console access to the node jail first.

To start the initial cluster node, run the following commands:

```
# ezjail-admin console galera-node
# service mysql start --wsrep-new-cluster
```

To start each additional node, run the following commands:

```
# ezjail-admin console galera-node
# service mysql start
```

Each node you start after the initial will attempt to establish network connectivity with the *Primary Component* and begin syncing their database states into one another.

Related Documents

- *Firewall Configuration with PF* (page 213)
- *Galera Cluster for MySQL* (page 43)
- *MariaDB Galera Cluster* (page 50)
- *wsrep_node_name* (page 250)
- *wsrep_provider* (page 254)
- *wsrep_node_address* (page 248)
- *wsrep_node_name* (page 250)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Database Server Logs* (page 145)

- *Notification Command* (page 201)
- *Using Status Variables* (page 139)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

CHAPTER

SIX

CLUSTER MONITORING

Occasionally, you may want or need to check on the status of the cluster. For instance, you may want to check the state of nodes. You may want to check for network connectivity problems amongst nodes.

There are four methods available in monitoring cluster activity and replication health: query status variables through a database client; check regularly related log files; use a monitoring application; or write a notification script.

- *Using Status Variables* (page 139)

  In addition to the standard status variables in MySQL, Galera Cluster provides a series of its own status variables. These will allow you to check node and cluster states, as well as replication health through the database client.

- *Database Server Logs* (page 145)

  Queries entered through the database client will provide information on the current state of the cluster. However, to check its history for more systemic issues, you need to check the logs. This section provides a guide to the Galera Cluster parameter used to configure database logging to ensure it records the information you need.

- *The Galera Manager* (page 147)

  If you have an account with Amazon Web Services (AWS) for server hosting, you can use Galera Manager to create and configure easily Galera Clusters. Additionally, Galera Manager provides a graphical user interface with charts for monitoring over one thousand cluster metrics: just configure it to track metrics you find most useful.

- *Notification Command* (page 201)

  Although checking logs and status variables may give you the information you need while logged into a node, getting information from them is a manual process. Using the notification command, you can set the node to call a script in response to changes in cluster membership or node status. You can use this to raise alerts and adjust load balances. You can use it in a script for any situation in which you need the infrastructure to respond to changes in the cluster.

You can also use Nagios for monitoring Galera Cluster. For more information, see Galera Cluster Nagios Plugin

Related Documents

- *Database Server Logs* (page 145)
- *Notification Command* (page 201)
- *Using Status Variables* (page 139)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training

- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Notification Command* (page 201)
- *Reset Quorum* (page 93)
- *checking node status* (page 141)
- *wsrep_cert_deps_distance* (page 311)
- *wsrep_cluster_address* (page 236)
- *wsrep_cluster_conf_id* (page 312)
- *wsrep_cluster_name* (page 237)
- *wsrep_cluster_size* (page 313)
- *wsrep_cluster_state_uuid* (page 313)
- *wsrep_cluster_status* (page 313)
- *wsrep_connected* (page 315)
- *wsrep_local_send_queue_avg* (page 325)
- *wsrep_local_state_comment* (page 327)
- *wsrep_local_recv_queue_avg* (page 324)
- *wsrep_local_recv_queue_max* (page 324)
- *wsrep_local_recv_queue_min* (page 324)
- *wsrep_ready* (page 331)
- *wsrep_applier_threads* (page 258)
- *wsrep_slave_threads* (page 259)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

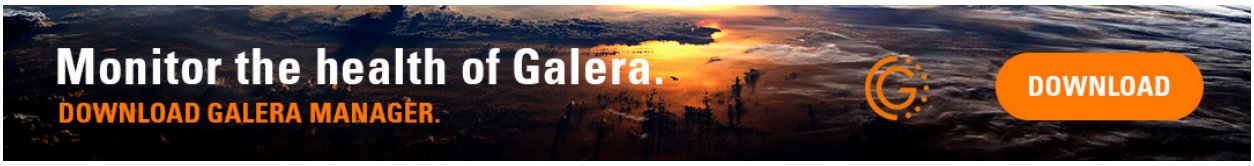
6.1 Using Status Variables

From the database client, you can check the status of write-set replication throughout the cluster using standard queries. Status variables that relate to write-set replication have the prefix `wsrep_`, meaning that you can display them all using the following query:

```
SHOW GLOBAL STATUS LIKE 'wsrep_%';

+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| wsrep_protocol_version | 5     |
| wsrep_last_committed   | 202   |
| ...                    | ...   |
| wsrep_thread_count     | 2     |
+------------------------+-------+
```

Note: In addition to checking status variables through the database client, you can also monitor for changes in cluster membership and node status through `wsrep_notify_cmd.sh`. For more information on its use, see *Notification Command* (page 201).



Checking Cluster Integrity

The cluster has integrity when all nodes in it receive and replicate write-sets from all other nodes. The cluster begins to lose integrity when this breaks down, such as when the cluster goes down, becomes partitioned, or experiences a split-brain situation.

You can check cluster integrity using the following status variables:

- *wsrep_cluster_state_uuid* (page 313) shows the cluster *state UUID*, which you can use to determine whether the node is part of the cluster.

  ```
  SHOW GLOBAL STATUS LIKE 'wsrep_cluster_state_uuid';

  +--------------------------+--------------------------------------+
  | Variable_name            | Value                                |
  +--------------------------+--------------------------------------+
  | wsrep_cluster_state_uuid | d6a51a3a-b378-11e4-924b-23b6ec126a13 |
  +--------------------------+--------------------------------------+
  ```

  Each node in the cluster should provide the same value. When a node carries a different value, this indicates that it is no longer connected to rest of the cluster. Once the node reestablishes connectivity, it realigns itself with the other nodes.

- *wsrep_cluster_conf_id* (page 312) shows the total number of cluster changes that have happened, which you can use to determine whether or not the node is a part of the *Primary Component*.

  ```
  SHOW GLOBAL STATUS LIKE 'wsrep_cluster_conf_id';

  +-----------------------+-------+
  | Variable_name         | Value |
  +-----------------------+-------+
  | wsrep_cluster_conf_id | 32    |
  +-----------------------+-------+
  ```

Each node in the cluster should provide the same value. When a node carries a different, this indicates that the cluster is partitioned. Once the node reestablishes network connectivity, the value aligns itself with the others.

- *wsrep_cluster_size* (page 313) shows the number of nodes in the cluster, which you can use to determine if any are missing.

```
SHOW GLOBAL STATUS LIKE 'wsrep_cluster_size';

+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| wsrep_cluster_size | 15    |
+--------------------+-------+
```

You can run this check on any node. When the check returns a value lower than the number of nodes in your cluster, it means that some nodes have lost network connectivity or they have failed.

- *wsrep_cluster_status* (page 313) shows the primary status of the cluster component that the node is in, which you can use in determining whether your cluster is experiencing a partition.

```
SHOW GLOBAL STATUS LIKE 'wsrep_cluster_status';

+----------------------+---------+
| Variable_name        | Value   |
+----------------------+---------+
| wsrep_cluster_status | Primary |
+----------------------+---------+
```

The node should only return a value of `Primary`. Any other value indicates that the node is part of a nonoperational component. This occurs in cases of multiple membership changes that result in a loss of *Quorum* or in cases of split-brain situations.

Note: If you check all nodes in your cluster and find none that return a value of `Primary`, see *Resetting the Quorum* (page 93).

When these status variables check out and return the desired results on each node, the cluster is up and has integrity. What this means is that replication is able to occur normally on every node. The next step then is *checking node status* (page 141) to ensure that they are all in working order and able to receive write-sets.

Checking the Node Status

In addition to checking cluster integrity, you can also monitor the status of individual nodes. This shows whether nodes receive and process updates from the cluster write-sets and can indicate problems that may prevent replication.

- *wsrep_ready* (page 331) shows whether the node can accept queries.

```
SHOW GLOBAL STATUS LIKE 'wsrep_ready';

+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| wsrep_ready   | ON    |
+---------------+-------+
```

When the node returns a value of `ON` it can accept write-sets from the cluster. When it returns the value `OFF`, almost all queries fail with the error:

```
ERROR 1047 (08501) Unknown Command
```

- *wsrep_connected* (page 315) shows whether the node has network connectivity with any other nodes.

```
SHOW GLOBAL STATUS LIKE 'wsrep_connected';

+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| wsrep_connected | ON    |
+-----------------+-------+
```

  When the value is `ON`, the node has a network connection to one or more other nodes forming a cluster component. When the value is `OFF`, the node does not have a connection to any cluster components.

  Note: The reason for a loss of connectivity can also relate to misconfiguration. For instance, if the node uses invalid values for the *wsrep_cluster_address* (page 236) or *wsrep_cluster_name* (page 237) parameters.

  Check the error log for proper diagnostics.

- *wsrep_local_state_comment* (page 327) shows the node state in a human readable format.

```
SHOW GLOBAL STATUS LIKE 'wsrep_local_state_comment';

+---------------------------+--------+
| Variable_name             | Value  |
+---------------------------+--------+
| wsrep_local_state_comment | Joined |
+---------------------------+--------+
```

  When the node is part of the *Primary Component*, the typical return values are `Joining`, `Waiting on SST`, `Joined`, `Synced` or `Donor`. If the node is part of a nonoperational component, the return value is `Initialized`.

  Note: If the node returns any value other than the one listed here, the state comment is momentary and transient. Check the status variable again for an update.

In the event that each status variable returns the desired values, the node is in working order. This means that it is receiving write-sets from the cluster and replicating them to tables in the local database.

Checking the Replication Health

Monitoring cluster integrity and node status can show you issues that may prevent or otherwise block replication. These status variables will help in identifying performance issues and identifying problem areas so that you can get the most from your cluster.

Note: Unlike other the status variables, these are differential and reset on every `FLUSH STATUS` command.

Galera Cluster triggers a feedback mechanism called Flow Control to manage the replication process. When the local received queue of write-sets exceeds a certain threshold, the node engages Flow Control to pause replication while it catches up.

You can monitor the local received queue and Flow Control using the following status variables:

- *wsrep_local_recv_queue_avg* (page 324) shows the average size of the local received queue since the last status query.

```
SHOW STATUS LIKE 'wsrep_local_recv_queue_avg';

+----------------------------+----------+
| Variable_name              | Value    |
+----------------------------+----------+
| wsrep_local_recv_que_avg | 3.348452 |
+----------------------------+----------+
```

  When the node returns a value higher than `0.0` it means that the node cannot apply write-sets as fast as it receives them, which can lead to replication throttling.

  Note: In addition to this status variable, you can also use *wsrep_local_recv_queue_max* (page 324) and *wsrep_local_recv_queue_min* (page 324) to see the maximum and minimum sizes the node recorded for the local received queue.

- *wsrep_flow_control_paused* (page 318) shows the fraction of the time, since `FLUSH STATUS` was last called, that the node paused due to Flow Control.

```
SHOW STATUS LIKE 'wsrep_flow_control_paused';

+----------------------------+----------+
| Variable_name              | Value    |
+----------------------------+----------+
| wsrep_flow_control_paused | 0.184353 |
+----------------------------+----------+
```

  When the node returns a value of `0.0`, it indicates that the node did not pause due to Flow Control during this period. When the node returns a value of `1.0`, it indicates that the node spent the entire period paused. If the time between `FLUSH STATUS` and `SHOW STATUS` was one minute and the node returned `0.25`, it indicates that the node was paused for a total 15 seconds over that time period.

  Ideally, the return value should stay as close to `0.0` as possible, since this means the node is not falling behind the cluster. In the event that you find that the node is pausing frequently, you can adjust the *wsrep_slave_threads* (page 259) or *wsrep_applier_threads* (page 258) parameter or you can exclude the node from the cluster.

- *wsrep_cert_deps_distance* (page 311) shows the average distance between the lowest and highest sequence number, or seqno, values that the node can possibly apply in parallel.

```
SHOW STATUS LIKE 'wsrep_cert_deps_distance';

+----------------------------+---------+
| Variable_name              | Value   |
+----------------------------+---------+
| wsrep_cert_deps_distance | 23.8889 |
+----------------------------+---------+
```

  This represents the node's potential degree for parallelization. In other words, the optimal value you can use with the *wsrep_slave_threads* (page 259) or *wsrep_applier_threads* (page 258) parameter, given that there is no reason to assign more replica threads than transactions you can apply in parallel.

Detecting Slow Network Issues

While checking the status of Flow Control and the received queue can tell you how the database server copes with incoming write-sets, you can check the send queue to monitor for outgoing connectivity issues.

Note: Unlike other the status variables, these are differential and reset on every `FLUSH STATUS` command.

---

wsrep_local_send_queue_avg (page 325) shows an average for the send queue length since the last `FLUSH STATUS` query.

```
SHOW STATUS LIKE 'wsrep_local_send_queue_avg';

+----------------------------+-----------+
| Variable_name              | Value     |
+----------------------------+-----------+
| wsrep_local_send_queue_avg | 0.145000 |
+----------------------------+-----------+
```

Values much greater than `0.0` indicate replication throttling or network throughput issues, such as a bottleneck on the network link. The problem can occur at any layer from the physical components of your server to the configuration of the operating system.

Note: In addition to this status variable, you can also use *wsrep_local_send_queue_max* (page 326) and *wsrep_local_send_queue_min* (page 326) to see the maximum and minimum sizes the node recorded for the local send queue.

Related Documents

- *Notification Command* (page 201)
- *Reset Quorum* (page 93)
- *checking node status* (page 141)
- *wsrep_cert_deps_distance* (page 311)
- *wsrep_cluster_address* (page 236)
- *wsrep_cluster_conf_id* (page 312)
- *wsrep_cluster_name* (page 237)
- *wsrep_cluster_size* (page 313)
- *wsrep_cluster_state_uuid* (page 313)
- *wsrep_cluster_status* (page 313)
- *wsrep_connected* (page 315)
- *wsrep_local_send_queue_avg* (page 325)
- *wsrep_local_state_comment* (page 327)
- *wsrep_local_recv_queue_avg* (page 324)
- *wsrep_local_recv_queue_max* (page 324)
- *wsrep_local_recv_queue_min* (page 324)
- *wsrep_ready* (page 331)

- *wsrep_applier_threads* (page 258)
- *wsrep_slave_threads* (page 259)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *cert.log_conflicts* (page 279)
- *wsrep_debug* (page 240)
- *wsrep_log_conflicts* (page 245)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

6.2 Database Server Logs

Galera Cluster provides the same database server logging features available to MySQL, MariaDB and Percona XtraDB, depending on which you use. By default, it writes errors to a `<hostname>.err` file in the data directory. You can change this in the `my.cnf` configuration file using the log_error option, or by using the `--log-error` parameter.

Log Parameters

Galera Cluster provides parameters and wsrep options that allow you to enable error logging on events that are specific to the replication process. If you have a script monitoring the logs, these entries can give you information on conflicts occurring in the replication process.

- *wsrep_log_conflicts* (page 245): This parameter enables conflict logging for error logs. An example would be when two nodes attempt to write to the same row of the same table at the same time.
- *cert.log_conflicts* (page 279): This wsrep Provider option enables logging of information on certification failures during replication.

- *wsrep_debug* (page 240): This parameter enables debugging information for the database server logs.

  Warning: In addition to useful debugging information, this parameter also causes the database server to print authentication information, (that is, passwords) to the error logs. Do not enable it in production environments as it is a security vulnerability.

You can enable these through the `my.cnf` configuration file. The excerpt below is an example of these options and how they are enabled:

```
# wsrep Log Options
wsrep_log_conflicts=ON
wsrep_provider_options="cert.log_conflicts=ON"
wsrep_debug=ON
```

Additional Log Files

Whenever a node fails to apply an event on a replica node, the database server creates a special binary log file of the event in the data directory. The naming convention the node uses for the filename is `GRA_*.log`.

Related Documents

- *cert.log_conflicts* (page 279)
- *wsrep_debug* (page 240)
- *wsrep_log_conflicts* (page 245)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

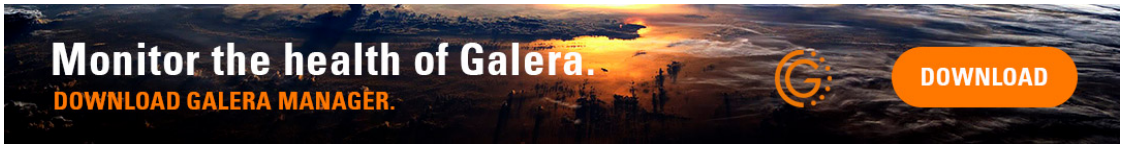
Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)
- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)

- *Upgrading* (page 199)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

6.3 The Galera Manager

The Galera Manager is a graphical user interface for provisioning and monitoring Galera Clusters in various environments, like Amazon Web Services (AWS) or on-premises hardware. It allows an administrator to add nodes easily, and without having to configure each node, manually. Perhaps more useful is that Galera Manager provides charts for monitoring host and database metrics to ensure the proper and efficient functioning of a Galera Cluster. There are over a thousand metrics from which to choose. You may use any standard web browser for accessing Galera Manager, to administer and monitor clusters.



This section of the Codership documentation provides detailed information and instructions on how to install and configure Galera Manager. Below is a brief summary of each aspects of the process to start using Galera Manager, with each heading linked to the related page for much more information—there are also links in the margin to all pages of the Galera Manager documentation. However, if you are an advanced administrator and are confident in your abilities, this page will provide you a summary of what you need to install and start using Galera Manager.

Install Galera Manager

The *Galera Manager Installer* is provided to make installation and configuration of Galera Manager as simple as possible. This section is only a brief summary of the procedure. Some of this text contains links to other pages where you will find more detailed information on each step.

Choose or Create a *Galera Manager Host*

Galera Manager is a server program (it serves client requests from both the cluster nodes and graphical frontend) so it should be installed on a host (server or other computer) which can be connected to from both the prospective cluster nodes and a computer running the graphical client. A laptop behind a WiFi NAT is a poor choice. You may use a local computer (such as a desktop or laptop computer), but most administrators would want to use a computer in the same network as the cluster nodes. For example, if the cluster is in EC2, you'd want to use an EC2 instance, if it is on premises, then you'd use a host in the on-premises network. It is also possible to use one of the cluster nodes, but it is recommended to have a dedicated *Galera Manager Host*.

For information on the operating systems that the Galera Manager Installer supports, see Supported Operating Systems

Future releases of Galera Manager Installer may support other platforms. For now, it is recommended you use one of these distributions for *Galera Manager Host*. The Galera Manager itself will, however,

manage or monitor clusters that may run on a different platform and nodes that will run either MySQL or MariaDB.

Download the *Installer*

After you've decided on and prepared the *Galera Manager Host*, you will need to download the *Installer* from Codership's site at this address:

https://galeracluster.com/galera-manager/gm-installer.

Run the *Installer*

Once the *Installer* has been downloaded, run it with superuser privileges to install Galera Manager. It will ask you some basic information: an administrative password, as well as a domain name and a site certificate, if you have these and want to use them.

```
chmod a+x gm-installer && sudo ./gm-installer install
```

When the Installer is finished, the *gmd* daemon will be running on the *Galera Manager Host*. The *Installer* will print out some bookkeeping information that you may want to save for future reference and also the address at which you can connect to *gmd* from your browser and start using it:

```
INFO[0223] Galera Manager installation finished. Enter http://10.0.3.73 in a␣
↪web browser to access.
```

Check the *Installing Galera Manager* (page 150) documentation page for more details on using the *Installer* and explanations of the questions you will be asked, as well as suggestions on how to respond to them. You might also read the *Galera Manager Daemon (gmd)* (page 163) page.

Deploying a Cluster

Having installed Galera Manager, you are now ready to use it to deploy a Galera Cluster, including adding and configuring nodes.

Access Galera Manager

In the address field of your web browser, enter the address provided in the *Installer* output.

If you didn't provide a certificate, your web browser may try to protect you from accessing the address. Push past those warnings until you reach the login screen. Then enter the administrative user name and password you gave when installing.

Create a Cluster & Add Nodes

After you log into Galera Manager, you may create a cluster, and then nodes to it. Typically, one would start with three nodes—you can add more nodes later, or delete some if you added too many. You will be able to choose between several node host types (`localxd`, `unmanaged`, `ec2`), host OS variants and database flavors. When you create a cluster, be sure to provide an public encryption key to facilitate manual troubleshooting via SSH connection.

Log in to cluster

If you created a cluster from scratch, you will need to get the login credentials (that is, user name, host address, password) to access one of the nodes with a MySQL client. This can be found by clicking on one of the nodes in Galera Manager, then its Configuration tab. There you will find the *DB Address* and the *DB Root Password* for accessing the database system.

You can find more details on deploying a cluster on the *Deploying a Cluster in Galera Manager* (page 167) and the *Adding Nodes with Galera Manager* (page 175) documentation pages. You may also find the *Adding Users to Galera Manager* (page 182) page helpful at some point.

Monitor a Cluster

With a Galera Cluster and nodes in place, including the data loaded and accessible to users, you can monitor your cluster using charts of database and host metrics in Galera Manager. Still, you may want to configure these charts or add more to suit your particular needs.

Configure Charts

By default, there are a few charts configured for commonly watched metrics. However, there are over one-thousand metrics that you may track. Click on the cluster in Galera Manager and you will see the default charts. You may click the *X* at the top right of any chart to remove it. To add a chart, click the vertical ellipsis to access a pull-down menu where you can select *Add Chart*. A dialog box will appear for you to choose the metric you want to monitor.

For more information on adding charts and related information, see the *Monitoring a Cluster with Galera Manager* (page 191) documentation page.

Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)
- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos

- FAQ
- search

Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)
- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

6.3.1 Installing Galera Manager

Galera Manager host requirements.

To use Galera Manager, you need to install it on a computer that

a) can be accessed from the cluster nodes you want to manage or monitor;

b) can be accessed from the computer where you'd want to display the GUI.

Normally that would be a computer in the same network as the prospective cluster nodes. Additionally you may want to consider providing sufficient disk space for the logging and metrics data. Technically Galera Manager can run on one of the cluster nodes, but it is recommended to use a dedicated machine.

Galera manager can use SSL encryption for all communications. However this requires the host to be accessible via a domain name, not just an IP address. Externally resolvable domain name is required to utilize external Certificate Authority.

Download the *Installer*

The *Installer* is a simple installation program for installing Galera Manager. When you run it, you will be asked a series of questions about configuring Galera Manager. After that it will download, install and configure required software packages. When it is finished, the `gmd` daemon will be started on the *Installer Host*, allowing you to use this server to deploy new Galera clusters in different environments, as well as monitor already existing clusters.

Below are more details on these steps to download and run the *Installer*. The questions you will be presented when installing are fairly self-explanatory. However, you may want to read this page before beginning, in case there are questions about which you want to know more before starting the installation.

To install Galera Manager, you will need to download the *Installer* to the computer chosen as the Galera Manager host. Currently *Installer* runs on the following x86_64 Linux distributions: Ubuntu 20.04 and Ubuntu 22.04, Debian 11 ("bullseye") and Debian 12 ("bookworm"), RedHat 8 and RedHat 9. Eventually, the *Installer* will be made available for other distributions.

After you've decided on and prepared the *Galera Manager Host*, you will need to download the *Installer* from Codership's site at this address:

https://galeracluster.com/galera-manager/gm-installer.

Listing 1: Making *Galera Manager Installer* Executable (Example 1)

```
chmod +x gm-installer
```

Having downloaded the installation program and made it executable, you are ready to run the *Installer* to install Galera Manager.

Start the Installer

There are two options available at this time when starting the *Installer*: `install` and `certificates`. The `install` option is necessary to install Galera Manager. The `certificates` option is used to generate your own, self-signed SSL certificates for encryption. Both options may be given together.

Note that to safely use SSL encryption Galera Manager host needs to be accessible through a valid DNS name (domain names given by Amazon EC2 do not count). The *Installer* will refuse to configure SSL encryption if the host has only an IP address. Galera Manager will retain full functionality working via unencrypted links.

Below is how you would start the *Installer* with only the `install` option. You'll have to run it with superuser privileges:

Listing 2: Starting Installation of *Galera Manager Installer* (Example 2)

```
sudo ./gm-installer install
```

After starting the *Installer*, you will first be asked to accept the Galera Manager End-User Licensing Agreement (EULA). Below is how this question will be presented—although it might change slightly in future releases:

Listing 3: Message about User Agreement from the *Installer* (Example 3)

```
To use GMD you must accept EULA.
Press [a] to accept it, [r] to read the EULA text, [n] to reject EULA.
```

If you are willing to accept the agreement, enter `a`. If you'd like to read the agreement, enter `r` and it will be displayed on the screen—along with the opportunity again to accept or reject the agreement. You can also read *the agreement* (page 161) in the documentation before even starting to install.

User Names & Passwords

Next you will be asked for Galera Manager repository address. If you've been given a link to a private repository, you will have to enter your user name and password for the repository. Then you will be asked for the login and password of the initial administrator of Galera Manager. You may want to ensure you have answers to the following questions:

Listing 4: Installation Credential Questions from the *Installer* (Example 4)

```
GMD Package Repository URL (blank for default):
GMD Package Repository User:
GMD Package Repository Password:
GMD Admin User Login [admin]:
GMD Admin Password:
```

The default user name is *admin*. Enter whatever password you'd like to use for the administrator. You'll be able to remove this user later and add a replacement administrator later, as well as add other users with lesser privileges. This is covered on the *Adding Users to Galera Manager* (page 182) page.

Domains & Certificates

You'll next need to provide either an IP address or a domain name for Galera Manager, the address on which you are running the *Installer*. This is the server where you will be accessing Galera Manager. Here are the related questions you will be presented:

Listing 5: *Installer* Messages about Site Address and Certification (Example 5)

```
By what domain name or IP address this service will be reached?
(Note that an externally resolvable domain name is needed to use an external
Certification Authority, otherwise we will have to resort to self-signed
certificates for SSL if encryption is required):
Enter your domain name or IP of the server:
```

An IP address works well, but you won't be able to utilize an external certification authority, neither you will be able to use self-signed certificates.

Listing 6: *Installer* Warning using an IP Address (Example 6)

```
Since you entered an IP address, SSL won't be available.
```

As this notification implies, SSL won't be available if Galera Manager host does not have a domain name.

If you chose to provide a resolvable domain name for Galera Manager host, you will have several options to set up SSL encryption (HTTPS protocol) to protect all Galera Manager connections (from both the GUI client and cluster nodes):

Listing 7: *Installer* Asking to Use a Secure Protocol (Example 7)

```
Enter your domain name or IP of the server: gm.example.com
Enable https? [Y/n]
Use LetsEncrypt Certbot to autogenerate the certificates [Y/n]:
```

Answering *Yes* to the above question will set up automatic certificates generation and renewal using LetsEncrypt site as a Certificate Authority. *NOTE:* not all domain names are accepted by LetsEncrypt, such as domain names autogenerated by AWS EC2 are not. If you want to set up your own Certificate Authority and/or use your own certificates, answer *No* and you will be offered to provide them:

Listing 8: *Installer* Questions about SSL credentials (Example 8)

```
Use LetsEncrypt Certbot to autogenerate the certificates [Y/n]: n
Do you want to provide your own SSL CA? [y/N] y
Use your own SSL certificate (y), or let installer generate one (n)? [y/N] y
SSL CA Certificate [ca.crt]:
SSL CA Certificate Key [ca.key]:
SSL Host Certificate [ssl.crt]:
SSL Host Certificate Key [ssl.key]:
```

NOTE: if you want to specify your own Certificate Authority, you need to make sure that it is known to your GUI frontend as well, otherwise it won't be able to confirm the validity of the Galera Manager certificate and most likely will refuse to connect with the warning about security risk.

Also you will be responsible to re-generate your own SSL certificate after it expires.

Closing Messages

After you finish answering all of the questions presented to you by the *Installer*, it will install and configure the software needed and start Galera Manager. You'll see messages regarding this pass by on the screen. At the end, if it is successful, you will see a message like this:

Listing 9: Final Messages after Successfully Installing Galera Manager (Example 9)

```
INFO[0223] Galera Manager installation finished. Enter http://10.0.3.73 in a web␣
↪browser to access. Please note, you chose to use an unencrypted http protocol, such␣
↪connections are prone to several types of security issues. Always use only trusted␣
↪networks when connecting to the service.
INFO[0223] Logs DB url: http://10.0.3.73:8081
Metrics DB url: http://10.0.3.73:8082
IMPORTANT: ensure TCP ports 80, 8081, 8082 are open in firewall.
INFO[0223] Below you can see Logs DB credentials (if once asked):
DB name: gmd
DB user: gmd
DB password: yAl4p84vR8
The installation log is located at /tmp/gm-installer.log
```

Note the ports that need to be opened on Galera Manager host.

TCP Ports

Regarding ports, notice the line in the example above about TCP ports 80, 8081, 8082. You'll need to make sure ports 8081, 8082 are accessible from the prospective nodes, and port 80 - from the GUI client.

If you deploy Galera Manager on AWS, those ports are closed by default. Go to the EC2 console in AWS, and click on *Security Groups* in the left margin. Then look for the security group for the server on which you installed Galera Manager. Edit the *Inbound Rules* for that group to open those ports (see the screenshot below).

In the example in this screenshot, notice that we set port 22 to the administrator's IP address to restrict access, in addtion to requiring an encryption key to log in. The other ports are accessible from anywhere so that you can access Galera Manager from wherever you and other administrators may be located. You may have noticed that port 3306 or other ports used by MySQL and Galera are not included in the *Inbound Rules* above. Those are needed by the nodes, not Galera Manager. When you add nodes, Galera Manager will add them to each host's *Inbound Rules*. You'll find more on these nuances by reading the *AWS Ports with Galera Manager* (page 157) page of this documentation.

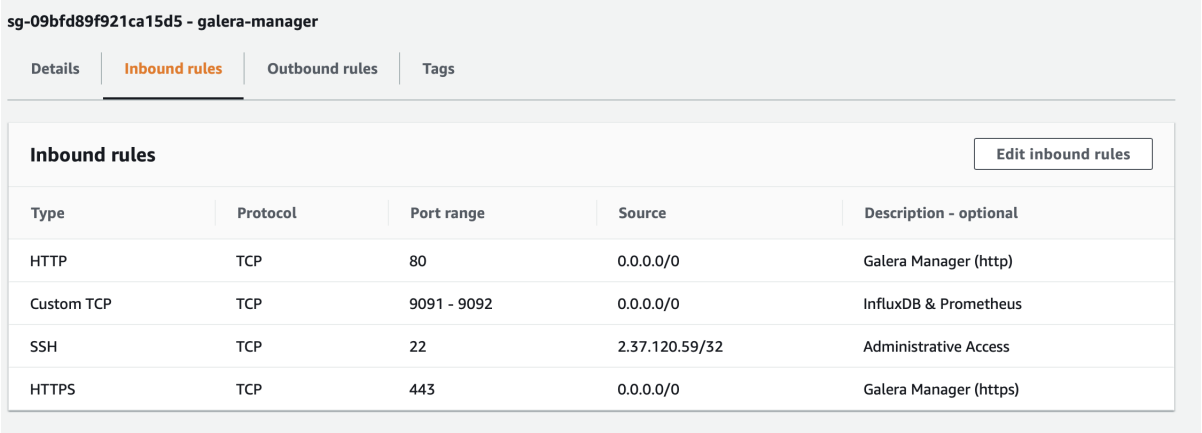


Fig. 1: AWS Inbound Rules for Galera Manager (Figure 1)

Logs & Installation Failure

In the last lines of the installation message, there's also the login name and password for accessing the InfluxDB database for the logs for the nodes. You wouldn't normally need to know these unless you are trying to debug something very unusual. They're used by Galera Manager behind-the-scenes. The logs are viewable within Galera Manager.

Should you encounter problems installing Galera Manager, though, check the installation log. It will be located in your server's temporary directory (for example, `/tmp`). You can see the file path and name of the installation log in the last line of a successful installation, as shown above. It is a simple and tidy text file that's easy to review, if you need it.

Galera Manager Daemon

Once you've answered all of the questions presented to you by the *Installer*, it will finish the installation and start the `gmd` daemon. You can enter something like the following from the command-line to check that it is running:

Listing 10: Checking if Galera Manager Daemon is Running (Example 10)

```
ps -e |grep gmd

30472 ?        00:00:40 gmd
```

The results showing the process identification number and the amount of time `gmd` has been running will be different on your server. For more information on the `gmd` daemon, or to learn how to make changes to some of its settings, see the documentation page called, *Galera Manager Daemon (gmd)* (page 163).

Connect to Galera Manager

After you've finished installing, you may log into Galera Manager with a standard web browser by entering the address where you installed it. At the end of the installation, there was a message like this one:

Listing 11: Installation Message containing URL for Galera Manager (Example 11)

```
INFO[0213] Galera Manager installation complete.
Direct your browser to http://34.217.114.37 to use it.
...
```

In the example here, a domain name wasn't used during the installation, so the URL has an IP address. If you provided a domain name, though, you would enter that domain name in your browser: `https://my-domain.com`.

If you didn't enable `https` when installing, you would instead start the URL with `http` (that is, without the `s`). Be aware that without that extra security layer, your connections will be vulnerable. Therefore, when using `http` for Galera Manager, you should use only trusted networks.

Shortly after you enter the URL for Galera Manager into your web browser, you will see a simple login screen like the one below. Here you will enter the *GMD Admin User* name and password you provided during the installation.

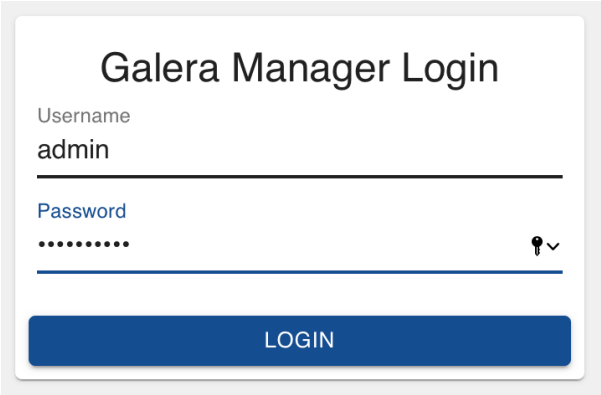


Fig. 2: Galera Manager Login (Figure 2)

At the start, after you log into Galera Manager for the first time, you will see only a fairly empty screen that shows something like the screenshot below. This is because you haven't yet created a cluster or added any nodes.

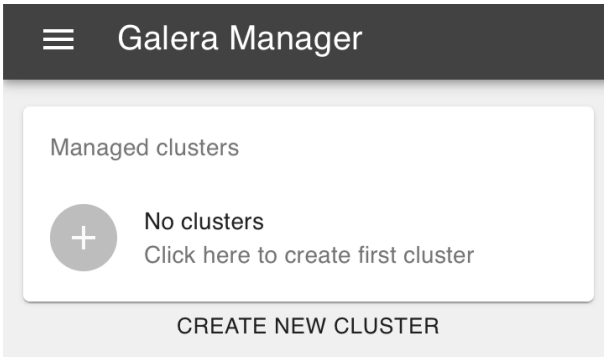


Fig. 3: New Galera Manager Installation (Figure 3)

To create a cluster, you would click on the plus-sign icon, or the text below the box where it says, *Create New Cluster*. The process for adding a cluster and nodes is covered on the *Deploying a Cluster in Galera Manager* (page 167) documentation page. For information on upgrading Galera Manager, see the *Upgrading Galera Manager (gmd)* (page 199) page.

Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)
- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)
- *AWS Ports* (page 157)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)
- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

6.3.2 AWS Ports with Galera Manager

There are several ports that Galera Manager uses to communication with the hosts in a cluster, as well as the ports that the nodes in a Galera Cluster use to communicate among themselves—and clients use to communicate with MySQL or MariaDB. There are also ports administrators need to access Galera Manager and the hosts.

You'll have to modify the *Security Group* on AWS (Amazon Web Services) for the *Instance* on which you installed Galera Manager. In that *Security Group*, you will have to add *Inbound Rules* for each of the ports that Galera Manager needs. A *Security Group* will be generated automatically by Galera Manager for each host added to a cluster, but you may want to edit each one, or you may want consolidate those *Security Groups* by creating one for all hosts in the cluster.

Below is information on ports used by Galera Manager, followed by information on the ports used by the hosts. See the *Installing Galera Manager* (page 150) for more information on installing Galera Manager.

Galera Manager Ports

When you successfully completed the installation of Galera Manager using the *Installer*, the final message displayed mentions the TCP ports used by Galera Manager. Below are excerpts from that message, showing the lead-in and the message about ports:

Listing 12: Closing Messages from Galera Manager Installer (Example 1)

```
INFO[0213] Galera Manager installation complete.
Direct your browser to https://34.217.114.37 to use it.
Since there was no publicly resolvable domain name provided,
we'll be using self-signed SSL certificate.
You will be responsible to re-generate it after it expires.
Also, if the browser warns about security risk when connecting
to service for the first time, you should choose to "continue".
...

Please make sure you have TCP ports 80, 443, 9091, 9092 open in the server firewall.
```

As the highlighted line at the bottom here says, you will need to make sure the TCP ports 80, 443, 9091, 9092 are open. This could be on a local computer or on an *AWS Instance* on which you've installed Galera Manager.

Ports 80 and 443 are used to access Galera Manager through a web browser. Port 9091 is used by `gmd` to access InfluxDB for logging, and port 9092 is used by `gmd` to access Prometheus for cluster and node metrics, both of which are installed by the *Installer*.

To open these ports on AWS, go to the EC2 console, and click on *Security Groups* in the left margin. Then look for the *Security Group* for the *Instance* on which you installed Galera Manager. Edit the *Inbound Rules* for that group to open those ports. When you are finished, the *Inbound Rules* will look something like the screenshot below:

In the example in this screenshot, these ports can be accessed from anywhere. The user will still need the user name and password to access the particular service. Having these ports accessible like this will allow you, and others you designate, to monitor your cluster from wherever you or they might be. If you do not need this flexibility, you could limit all of these ports to the specific IP addresses from where they would be accessed—just as you might normally limit `ssh` access for port 22 to the IP address of the administrator who has authority to log into the server.

Incidentally, the *Installer* message shown above is the result of having chosen to enable `https`. If you had chosen not to enable it, though, the list of ports to open in AWS will be different:

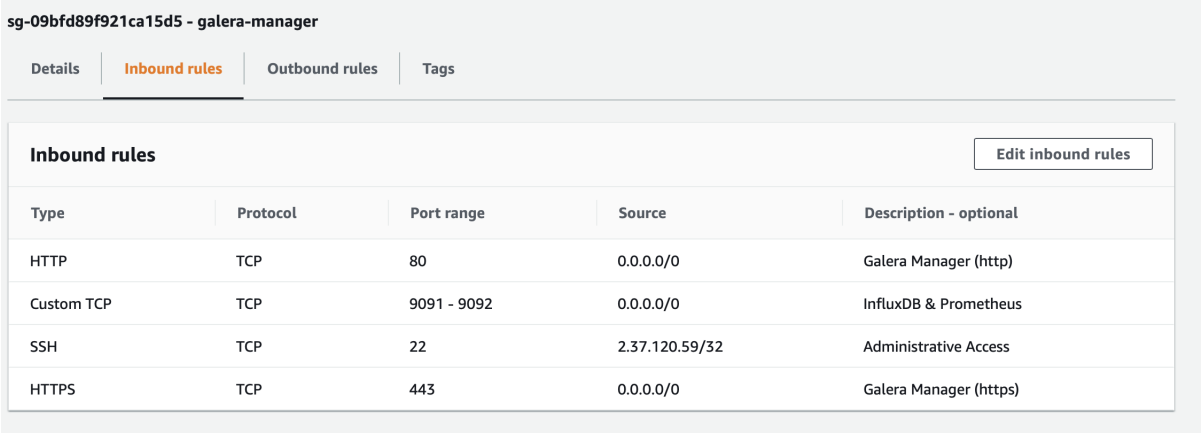


Fig. 4: AWS Inbound Rules for Galera Manager (Figure 1)

Listing 13: Excerpt from Installer listing Ports to Open (Example 2)

```
...
Please make sure you have TCP ports 80, 8081, 8082 open in the server firewall.
```

Only port 80 is used to access Galera Manager from a web broswer. Port 8081 is used by `gmd` to access InfluxDB; port 8082 is used for Prometheus. Since you didn't enable `https`, you'll have to open these three ports in the *Security Group* for the *Instance* on which you've installed Galera Manager. But in this case, you don't need also to enable ports 443, 9091, and 9092.

You may have noticed when looking at the lists of ports above and in Figure 1 that they do not include port 3306 and other ports used by MySQL, MariaDB and Galera Cluster. Galera Manager does not need them to create and monitor a cluster and nodes. Those ports are needed on the hosts of the nodes. So separate *Security Groups* will be needed for them.

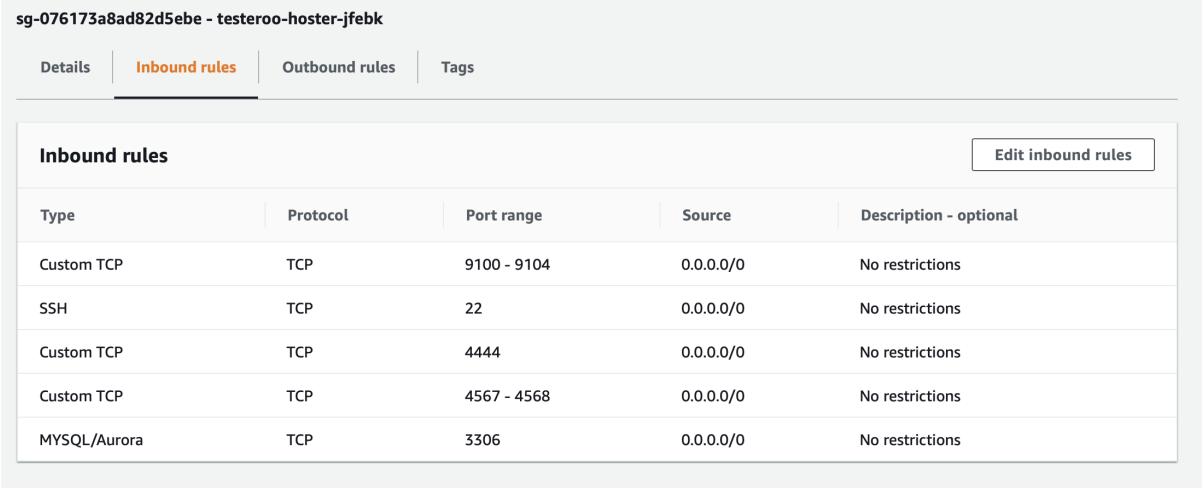
Host & Node Ports

After you create a cluster within Galera Manager, and then add nodes to that cluster, Galera Manager will create *Instances* or hosts for each node, and a *Security Group* in AWS with *Inbound Rules* for each host. These rules will open the ports needed for `gmd` on the Galera Manager server to communicate with the nodes, as well as the normal ports required by the nodes within a Galera Cluster to communicate with each other, in addition to ports for users and other clients need to communicate with MySQL or MariaDB.

The screenshot in Figure 2 below shows an example of a *Security Group* for a host created with Galera Manager for a node in a cluster:

Notice this *Security Group* includes ports 9091 and 9092, which are necessary for Galera Manager to communicate with the host related to host metrics and llogs. Port 3036 is used by users and clients to communicate with MySQL or MariaDB to access the databases. The other three ports are used by Galera Cluster and the nodes to communicate with each other and synchronize lagging or new nodes that join the cluster.

You might be tempted to tighten security more, to create one *Security Group* to be used by all hosts and in it to specify the IP addresses of each node for ports 4444, 4567, and 4568—perhaps because of the default description, *No Restructions* for each of those rules. Along these lines, you might also change ports 9100 - 9104 to the IP address of wherever Galera Manager or `gmd` is running. You would have to leave port 3306 accessible from anywhere, though, so that MySQL users and clients can access the databases.



sg-076173a8ad82d5ebe - testeroo-hoster-jfebk

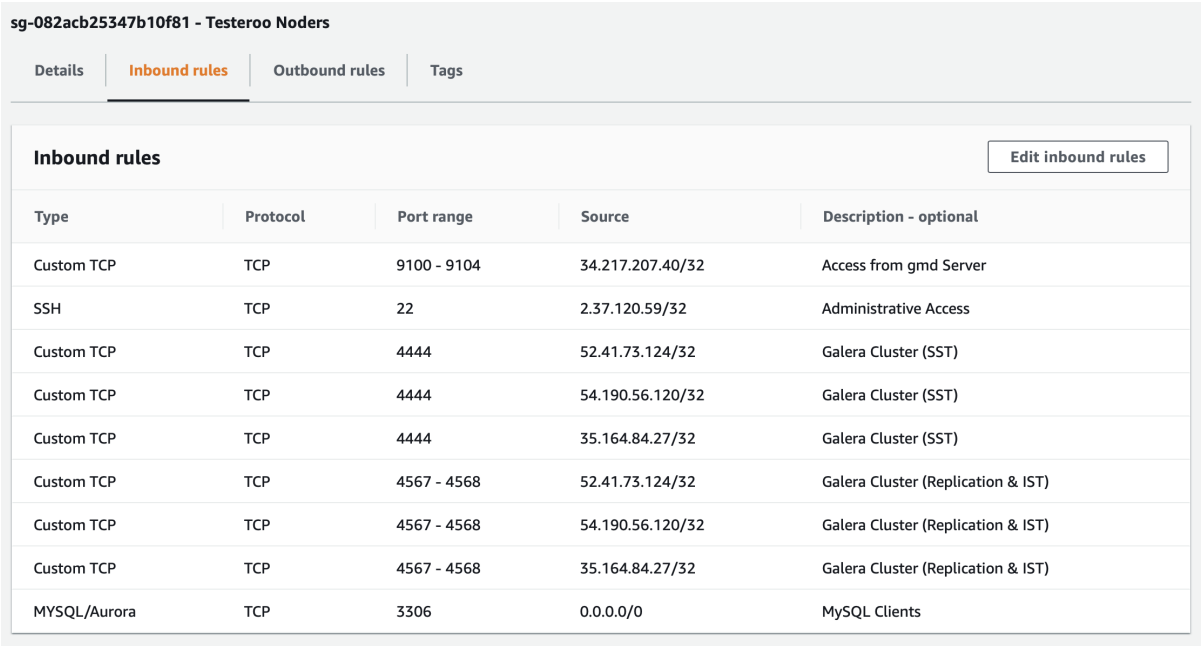
Details | Inbound rules | Outbound rules | Tags

Inbound rules — Edit inbound rules

Type	Protocol	Port range	Source	Description - optional
Custom TCP	TCP	9100 - 9104	0.0.0.0/0	No restrictions
SSH	TCP	22	0.0.0.0/0	No restrictions
Custom TCP	TCP	4444	0.0.0.0/0	No restrictions
Custom TCP	TCP	4567 - 4568	0.0.0.0/0	No restrictions
MYSQL/Aurora	TCP	3306	0.0.0.0/0	No restrictions

Fig. 5: AWS Inbound Rules for a Host (Figure 2)

If you were to do all of that, make all of those changes to the *Source* IP addresses, it would like something like what you see in the screenshot in Figure 3 below:



sg-082acb25347b10f81 - Testeroo Noders

Details | Inbound rules | Outbound rules | Tags

Inbound rules — Edit inbound rules

Type	Protocol	Port range	Source	Description - optional
Custom TCP	TCP	9100 - 9104	34.217.207.40/32	Access from gmd Server
SSH	TCP	22	2.37.120.59/32	Administrative Access
Custom TCP	TCP	4444	52.41.73.124/32	Galera Cluster (SST)
Custom TCP	TCP	4444	54.190.56.120/32	Galera Cluster (SST)
Custom TCP	TCP	4444	35.164.84.27/32	Galera Cluster (SST)
Custom TCP	TCP	4567 - 4568	52.41.73.124/32	Galera Cluster (Replication & IST)
Custom TCP	TCP	4567 - 4568	54.190.56.120/32	Galera Cluster (Replication & IST)
Custom TCP	TCP	4567 - 4568	35.164.84.27/32	Galera Cluster (Replication & IST)
MYSQL/Aurora	TCP	3306	0.0.0.0/0	MySQL Clients

Fig. 6: *AWS Security Group* for All Hosts (Figure 3)

That looks professional and is in a sense more secure. However, it takes away from the flexibility of using Galera Manager for adding and removing nodes, if you have to add and delete rules manually for each host, and set the new hosts to this *Security Group*. Given that the traffic between Galera Manager and the hosts is already very secure already, limiting the IP addresses might be overkill. Should you decide that you want this extra security and that it is worth the effort, you probably won't have to do it in future releases of Galera Manager, as it is on the list of features to add.

For more on the nuances and configuring some of the background settings of Galera Manager, see the *Galera Manager Daemon (gmd)* (page 163) page of this documentation.

Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)
- *AWS Ports* (page 157)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)
- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)
- *AWS Ports* (page 157)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)
- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

6.3.3 Galera Manager End-User License Agreement (EULA)

THE SOFTWARE LICENSE TERMS CONTAINED HEREIN (THE "LICENSE TERMS") CONSTITUTE A LEGAL AND BINDING AGREEMENT BETWEEN YOU ("YOU" OR "CUSTOMER") AND CODERSHIP OY ("COMPANY"). BY DOWNLOADING THE SOFTWARE ("GALERA MANAGER") AND/OR INSTALLING AND USING CODERSHIP OY SOFTWARE (THE "PRODUCT"), YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS AGREEMENT. READ IT CAREFULLY BEFORE COMPLETING THE INSTALLATION PROCESS AND USING THE PRODUCT.

IF YOU DO NOT AGREE TO BE BOUND BY THESE TERMS, OR THE PERSON OR ENTITY INSTALLING AND/OR USING THE PRODUCT DOES NOT HAVE AUTHORITY TO BIND THE CUSTOMER TO THESE LICENSE TERMS, THEN DO NOT INSTALL AND/OR USE THE PRODUCTS.

1. Grant of License and Restrictions. Subject to the License Terms, Company grants Customer a non-sublicensable, non-exclusive right to use the Product strictly in accordance with the related user documentation and specification sheets (collectively, the "Documentation") and any terms and procedures Company may prescribe from time to time. Company retains complete ownership of the Product and copies. The Customer must maintain the Copyright Notice and any other notices that appear on the Product on any copies and any media. Customer will not (and will not allow any third party to:

    i. reverse engineer or attempt to discover any source code or underlying ideas or algorithms of any Product,

    ii. provide, lease, lend, use for time sharing or service bureau purposes or otherwise use or allow others to use the Product for the benefit of any third party, or (iii) use any Product, or allow the transfer, transmission, export, or re-export of any Products or portion thereof in violation of any export control laws or regulations administered by the U.S. Commerce Department, OFAC, or any other government agency.

   All the limitations and restrictions on the Products in these License Terms also apply to the Documentation.

2. Termination. This Agreement and all licenses hereunder may be terminated by the Company at any time if the Customer fails to comply with any term of this Agreement. Upon termination, Customer shall immediately cease all use of all affected Products and return or destroy all copies of all affected Products and all portions thereof and so certify to the Company in writing. Except as otherwise expressly provided herein, the terms of this Agreement shall survive termination. Termination is not an exclusive remedy and all other remedies will be available to the Company whether or not termination occurs.

3. Limitation of Liability. THE PRODUCT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. FURTHER, THE COMPANY DOESNOT WARRANT RESULTS OF USE OR THAT THE PRODUCT IS BUG FREE OR THAT THEIR USE WILL BE UNINTERRUPTED. THE COMPANY FURTHER SHALL NOT BE RESPONSIBLE FOR ANY COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES OR RIGHTS, FOR ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR INTERRUPTION OF USE OR LOSS OR CORRUPTION OF DATA.

4. Miscellaneous. Neither this Agreement nor the licenses granted hereunder are assignable or transferable by the Customer without the prior written consent of the Company and any attempt to do so shall be void. No failure or delay in exercising any right hereunder will operate as a waiver thereof, nor will any partial exercise of any right or power hereunder preclude further exercise. If any provision of this Agreement shall be adjudged by any court of competent jurisdiction to be unenforceable or invalid, that provision shall be limited or eliminated to the minimum extent necessary so that this Agreement shall otherwise remain in full force and effect and enforceable. This Agreement shall be

> construed pursuant to the laws of Finland without regard to the United Nations Convention on the International Sale of Goods. This Agreement is the complete and exclusive statement of the mutual understanding of the parties and supersedes and cancels all previous written and oral agreements and communications relating to the subject matter of this Agreement. In any action to enforce this Agreement, the Company will be entitled to recover its attorney's fees and costs in connection with such action.

Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)
- *AWS Ports* (page 157)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)
- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)
- *AWS Ports* (page 157)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)
- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)
- Home

- *Docs* (page 1)
- KB
- Training
- FAQ

6.3.4 Galera Manager Daemon (gmd)

The Galera Manager is driven by the gmd daemon program that can create clusters, add and remove nodes, and gather monitoring data from the Galera Cluster. For information on installing gmd, see the documentation page, *Installing Galera Manager* (page 150).

gmd Process

If Galera Manager was installed on a host, you can enter something like the following from the command-line to check that it is running:

Listing 14: Checking if Galera Manager Daemon is Running (Example 1)

```
ps -e |grep gmd

5810 ?        00:00:18 gmd
```

The results showing the process identification number and the amount of time gmd has been running will be different on your server. Although it is unlikely you will need to restart gmd, to do so you may enter the following from the command-line:

Listing 15: Restarting the Galera Manager Daemon (Example 2)

```
systemctl restart gmd
```

You can replace restart with stop to shutdown the Galera Manager daemon—and use start to start it later. If the server is rebooted, gmd is set to start automatically.

Configuration File

When you installed Galera Manager, the *Installer* created a configuration file for gmd based on the responses you gave. You do not have to create it yourself. However, if you want to change some of the information you provided when installing, you can edit the configuration file. It is located in the sub-directory, /etc/default/ and called, gmd.

The gmd configuration file will look something like this:

Listing 16: Contents of Galera Manager Configuration File (Example 3)

```
ARGS="--rsa-private-key=/var/lib/gmd/jwt-rsa.key"
GMD_CONFIG_DIR=/var/lib/gmd
GMD_LOGS_DIR=/var/log/gmd
INFLUXDB_URL=https://gmd:8hCh2GeYv9@34.217.207.40:8091
PROMETHEUS_URL=https://34.217.207.40:8092
```

There are few settings here. You can change the values with a simple text editor. Just remember to restart gmd for the changes to take effect. See above for how to restart the daemon.

gmd Logs

In the previous section, you may have noticed the location of the log files: `/var/log/gmd`. Should you have difficulty starting `gmd` or encounter similar problems, you can check this directory for log files containing messages that may indicate the cause. Below is an example of the contents of that log file directory:

Listing 17: List of Galera Manager Log Files (Example 3)

```
ls -1 /var/log/gmd

cluster-testeroo.log
default.log
host-hoster-jfebk-stdout.log
host-hoster-jfebk.log
host-hoster-lisvt-stdout.log
host-hoster-lisvt.log
host-hoster-mlksh-stdout.log
host-hoster-mlksh.log
node-noder-jfebk.log
node-noder-lisvt.log
node-noder-mlksh.log
```

There's a log file for the `gmd` daemon (that is, `default.log`), one for the cluster, a pair for each host, and one for each node.

You may be confused as to the difference between a host and a node in this context. A host has to do with the computer system on which the Galera Cluster software is installed. This includes software configuration, network traffic, as well as where particular software like Galera Manager and MySQL are running. Whereas, a node is a database engine process running on the host and forming the Galera Cluster by connecting with other such processes running elsewhere. Is the node available and handling database client traffic? Is it synchronized with the other nodes in the cluster?

What's important to an administrator, though, is knowing where to find log messages to troubleshoot problems that may be encountered. Below are descriptions of what may be found in each log, with the most information recorded in the host standard output log (for example, `host-hoster-mlksh-stdout.log`).

Default Log

The main log file for the `gmd` daemon, the `default.log` file, contains information related to starting and stopping the daemon. Here's an excerpt from such a log file:

Listing 18: Excerpt from Galera Manager's Default Log (Example 4)

```
time="2020-05-18T08:05:19Z" level=info msg="Starting gmd"
time="2020-05-18T08:05:19Z" level=info msg="Listening on 127.0.0.1:8000"
time="2020-05-18T08:05:19Z" level=info msg="ConfigDir = /var/lib/gmd"
time="2020-05-18T08:05:19Z" level=info msg="LogsDir = /var/log/gmd"
```

As you can see, it records when it started the `gmd` daemon, on which IP address and port it is listening for connections from users (such as `admin`), and the directories for configuration and log files.

Cluster Log

As mentioned above, there's a log file for the cluster. Its name contains the name of the cluster appended to it (for example, `testeroo` from the examples on other pages of this documentation section). This log file contains some very basic information on the settings of the cluster. Below is an example of its contents:

Listing 19: Excerpt from Galera Manager's Cluster Log (Example 5)

```
time="2020-06-07T06:27:39Z" level=info msg="cluster record created" cluster-
↪name=testeroo
```

It is not much, since it is from a new installation of Galera Manager, one used in examples elsewhere in this documentation. It contains the date and time the cluster was created, as well as the name of the cluster. As a result of that name, this log file is named, `cluster-testeroo.log`.

Host Logs

As mentioned earlier, there is a pair of log files for each host in the cluster. One is labeled `host`, followed by the name of the host and the extension, `.log` This file contains primarily entries showing the data time or changes to the host's status.

Below is an excerpt from the `host-hoster-mlksh.log` file from the examples used here in documentation on Galera Manager:

Listing 20: Excerpt from a Galera Manager Host Log (Example 6)

```
time="2020-06-07T06:28:58Z" level=info
   msg="setting deployment status to pending" host-name=hoster-mlksh
time="2020-06-07T06:30:04Z" level=info
  msg="setting deployment status to ok" host-name=hoster-mlksh
```

This is actually two lines of entries, but we broke the lines to fit more easily on the screen. Still, there's not much information here. Nevertheless, you might write a custom shell script to parse this file to check for the latest entry, looking for when the deployment status is not *ok*, and send you a message saying as much—and then have `cron` run that script frequently at regular intervals. Or you could just keep Galera Manager open in a window on your computer.

The other log file for each host is labeled `host`, followed by the name of the host, then `stdout` and the extention, `.log` (for example, `host-hoster-mlksh-stdout.log`). This log file contains the messages generated by the host server when activities happen, when various commands, utilities and other programs are run by Galera Manager. If these commands and all were executed manually, some messages would normally be shown on the screen (that is, the standard output). However, since they're run in the background, there's no one to see them. So Galera Manager writes them to a log file for each host.

These host `stdout` log files are extensive. They contain information on updating Galera Manager software, network traffic, and many other logistical system information related to Galera. As a result, they can become fairly large files. But they can also be useful when trying to troubleshoot a problem with Galera Manager software—but not the synchronizing and accessing of data within the cluster, on nodes.

Node Logs

In the log directory for `gmd`, there is a log file for each node. As mentioned earlier, these log files contain information related to the nodes of the cluster, their interactions with each other. Below is an excerpt from the `node-noder-mlksh.log` file from examples elsewhere in this documentation:

Listing 21: Excerpt from a Galera Manager Node Log (Example 7)

```
time="2020-06-07T06:31:54Z" level=info msg="updating cluster IPs" ctx=update-cluster-
↪ips node-name=noder-mlksh
time="2020-06-07T08:15:09Z" level=info msg="checking node status" node-name=noder-
↪mlksh
```

(continues on next page)

(continued from previous page)

```
time="2020-06-07T08:15:10Z" level=info msg="node status is healthy" node-name=noder-
↪mlksh
time="2020-06-07T08:15:10Z" level=info msg="already started" node-name=noder-mlksh
```

Notice these entries are related to nodes in the cluster having started, being ready to accept MySQL client traffic, and in sync—that is to say, the node's health.

Should one of the nodes have problems that are not reflected in the metrics you are tracking in Galera Manager, you could check the log for that node for an indication of what's wrong with it. Afterwards, you might want to add the appropriate metrics to Galera Manager to monitor the situation more closely and conveniently from within Galera Manager. For more information on adding metrics to track in Galera Manager, see the *Monitoring a Cluster with Galera Manager* (page 191) documentation page.

Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)
- *AWS Ports* (page 157)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)
- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)
- *AWS Ports* (page 157)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)

- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

6.3.5 Deploying a Cluster in Galera Manager

With Galera Manager installed, you are ready to create a Galera Cluster or start monitoring an existing cluster. This page of the Codership documentation describes how to connect to Galera Manager, create a cluster and how to add nodes to a cluster. If you haven't already installed Galera Manager, go to the *Installing Galera Manager* (page 150) documentation page to do that first.

Without Galera Manager, to create a Galera Cluster, you have to set up multiple servers or AWS Instances, and then install MySQL or MariaDB and Galera software on each. You also have to configure each server or node. It is a fairly detailed process. Instead, you can use the Galera Manager to make the process of creating a cluster and adding nodes simple and quick.

Create a Cluster

To create a cluster in Galera Manager, click on *Create New Cluster*. You'll then see a large dialog box like the one below in Figure 1. In this box you will give the cluster a name, as well as make some default choices for creating nodes and hosts

Looking at the screenshot here, in the first section labeled *Cluster Configuration*, you can see that you have to provide a name for your cluster. In this example, the name *testeroo* was given, but you should enter something more meaningful to your organization or system—especially if you will be creating more than one cluster.

Default Node Configuration

In the next section of the box shown in Figure 1, the section labeled *Node Configuration*, you are asked to make default choices that will be used when you later add nodes to the cluster. To be clear as to what's discussed, below is the screenshot from Figure 1, but cropped around the default node configuration section:

The first field in this section asks you to specify which version of MySQL or MariaDB you want to use. It is not a default but a final choice, because the nodes should all use the same version of the same database system. You shouldn't have one node in a cluster using MySQL and another MariaDB, or even have them all using MySQL, but different versions. If you create another cluster, however, it may use a different database system and version.

Database Engine Configuration

Next, to the right in Figure 2, you may provide default custom configuration for your nodes, for example `innodb_buffer_pool_size` or `wsrep_provider_options`. This should be given in the same format you'd use in `my.cnf`. You would click on the icon of a cogwheel, where is says *Custom DB Engine Configuration*, to add

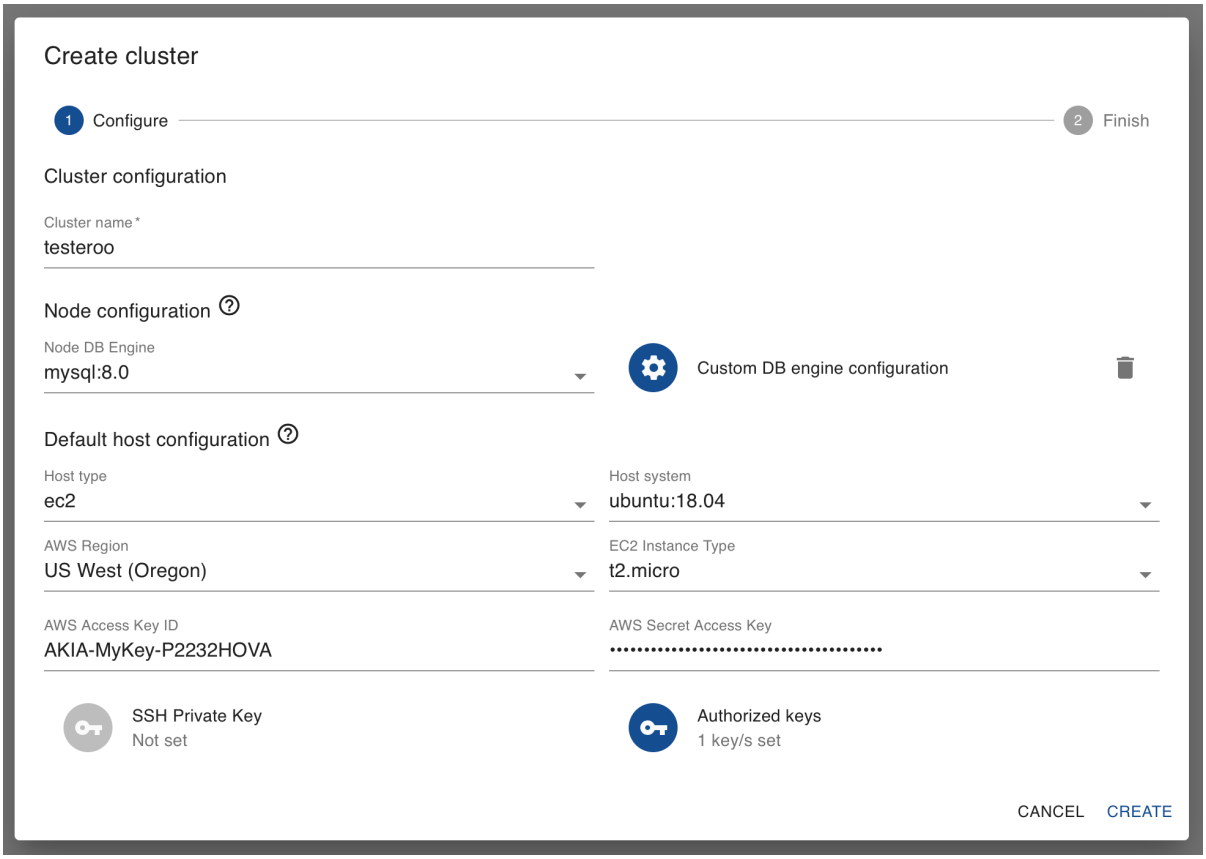


Fig. 7: Create Cluster Dialog Box (Figure 1)

Fig. 8: Host Defaults Section of Create Cluster Dialog Box (Figure 2)

those options or variable names with the values you want. Galera Manager manages configuration pertaining to its functionality (for example `wsrep_cluster_address`), so those variables managed by Galera Manager will be overridden.

In Example 1 below is the contents of `/etc/mysql/mysql.conf.d/mysqld.cnf` on one of the nodes which is running Ubuntu—it is the same for each node:

Listing 22: MySQL Daemon Configuration File (Example 1)

```
[mysqld]
pid-file      = /var/run/mysqld/mysqld.pid
socket = /var/run/mysqld/mysqld.sock
datadir = /var/lib/mysql
log-error = /var/log/mysql/error.log
```

These are minimal settings for MySQL. For Galera Cluster, there is an additional configuration file. These are the settings in `/etc/mysql/wsrep/conf.d/99.galera.cnf`, for the same node running Ubuntu:

Listing 23: Galera Configuration File (Example 2)

```
[mysqld]
bind-address = 0.0.0.0
wsrep_on = ON
# make it dynamic according to the distribution
wsrep_provider = /usr/lib/galera/libgalera_smm.so
wsrep_cluster_address = 'gcomm://52.25.88.43,54.213.111.232,35.163.3.151'
wsrep_cluster_name = 'testeroo'
wsrep_node_name = 'noder-jfebk'
wsrep_sst_method = rsync
binlog_format = row
default_storage_engine = InnoDB
innodb_autoinc_lock_mode = 2
log_error = /var/log/mysql/error.log
general_log_file = /var/log//mysql/general.log
general_log = 1
slow_query_log_file = /var/log/mysql/mysql-slow.log
slow_query_log = 1
```

Although these excerpts were taken from Ubuntu nodes, the same settings will be found in nodes using other Linux distributions generated by Galera Manager. Notice that two lines were set for this particular cluster: `wsrep_cluster_address` contains all of the IP addresses of the nodes; and `wsrep_cluster_name` contains the name of the cluster. There's one line that is set for this particular node: `wsrep_node_name` contains the node's name. These adjustments are made by Galera Manager when adding nodes.

Again, if you want to add some other settings, perhaps setting values for InnoDB buffers, you add them to the box for *Custom DB Engine Configuration*. Below in Figure 3 is a screenshot of that box and how you might enter values:

Notice that you have to include the `[mysqld]` heading. When you are finished, click on *Set* to save. At this time, you won't be able to make changes to these settings once you finish creating the cluster. So be sure you have everything you want before clicking *Create*. Otherwise, you will have to log into each node to make changes manually to the configuration files and restart the nodes. However, when adding a new node to the cluster you will be able to provide an alternative configuration. In a future release, you will be able to edit these default settings from within Galera Manager.

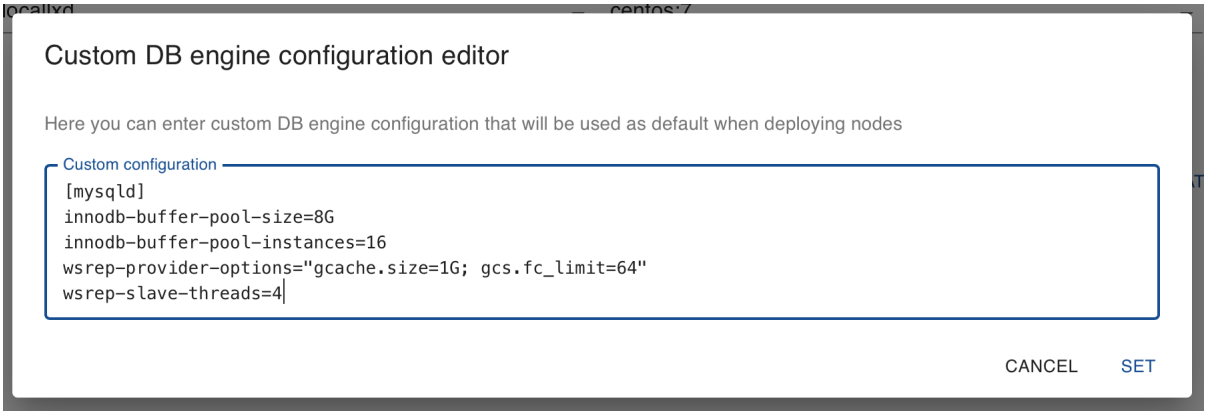


Fig. 9: Node Default Database Custom Configuration (Figure 3)

Default Host Configuration

The next section of the *Create Cluster* dialog box relates to how you want to create hosts, the servers on which nodes will be running. This has to do primarily with the operating system to install on the servers and how you will access those servers using `ssh`.

To make it easier to discuss, below is the screenshot from Figure 1, but this time cropped around the default host configuration section:

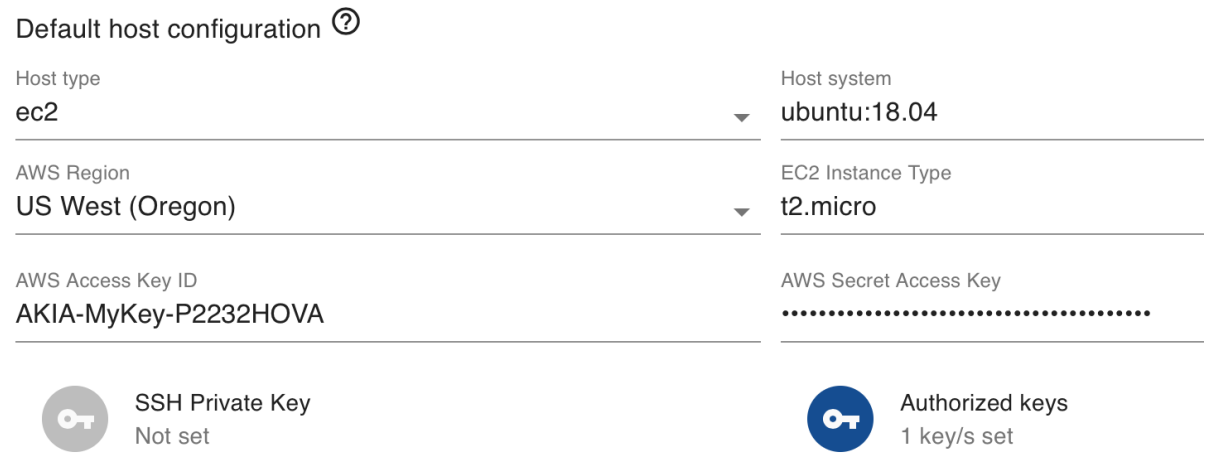


Fig. 10: Host Defaults Section of Create Cluster Dialog Box (Figure 4)

The first field allows you to chose the host type: *locallxd*, *ec2* or *unmanaged*. Choosing *locallxd* will instruct Galera Manager to generate Linux Containers, using the `lxd` daemon, to create hosts when you add nodes later. They'll all run on the same server where you have Galera Manager. This option is primarily for testing purposes and shouldn't generally be used.

Choosing *ec2* will use Amazon's EC2 service to create separate *AWS Instances* for each host needed for each node you add to the cluster. When you choose this, there will be fields allowing you to choose which AWS region to use for hosts, and which type of *Instance*—these relate to the size and speed of the server.

Choosing *unmanaged* will mean that you already have a manually managed running cluster. In that case Galera Manager will attempt only to install and configure monitoring software on the nodes and will perform only cluster

monitoring but not management.

In a future release, Galera Manager will add more host types an support more cloud providers.

AWS Access Keys

In case you chose *ec2* host type, you will also be asked to provide your *Access Key* information so that Galera Manager may interface with AWS on your behalf. This information is secure and not shared with Codership: it is confined to your installation of Galera Manager. Even if you already have a copy of the ID and the password, you may want create another key for use only by Galera Manager. You would do this on AWS's site. To get to this page, log into the EC2 console. Then click where the name for your account is show. A pull-down menu will appear, as you see in the screenshot below:

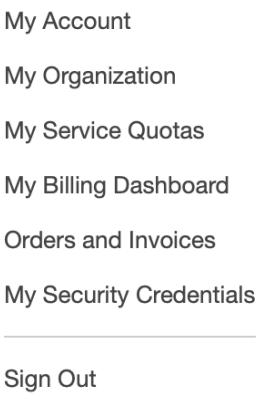


Fig. 11: AWS Account Menu (Figure 5)

Click where it says, *My Security Credentials*. This will take you to the *Identity and Access Management (IAM)* page. Look in the main panel for the choice labeled, *Access Keys* and click on it to expand that sub-section. Your screen will look something like the screenshot below:

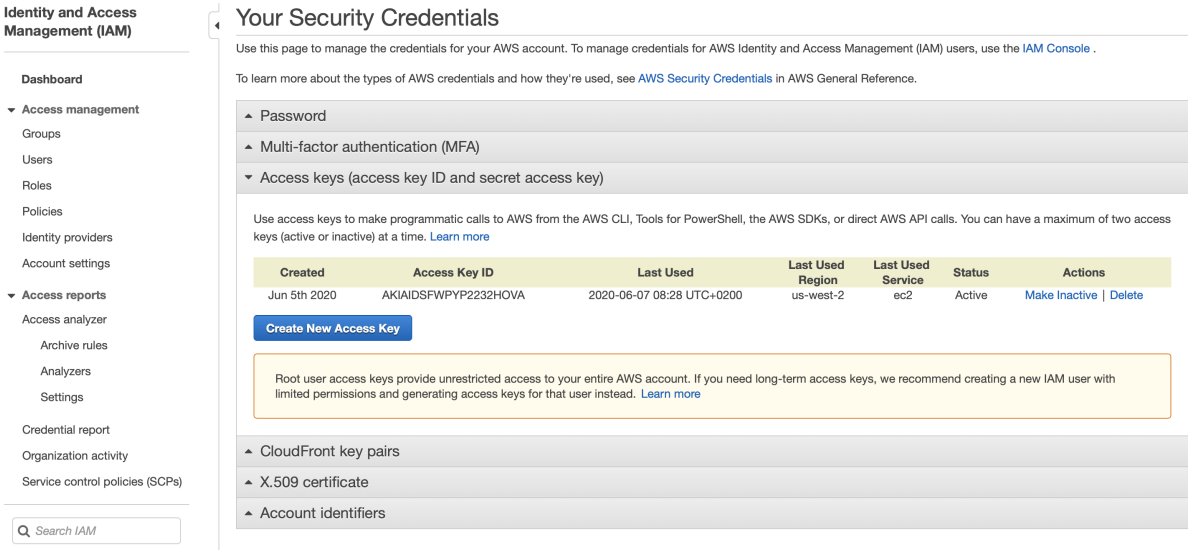


Fig. 12: AWS Security Credentials (Figure 6)

Then just click on the blue button labeled, *Create New Access Key*. It will immediately create a new *AWS Access Key ID* and *AWS Access Key*. You'll see a box appear like the one in the screenshot below:

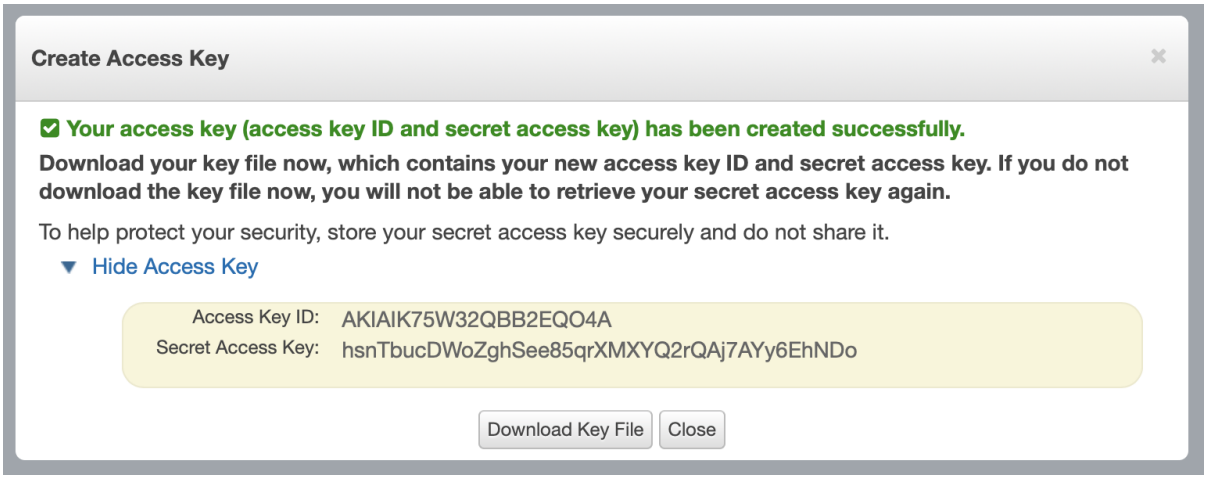


Fig. 13: Created AWS Access Key (Figure 7)

You can copy the text containing the *AWS Access Key ID* and the *AWS Access Key* (see Figure 7), and paste each of them into the appropriate fields in Galera Manager where you are in the process of creating a cluster, in the *Default Host Configuration* section. You may also want to click on the gray button that reads, *Download Key File* to download the *AWS Access Key*. This will download a CSV file containing the same information. Make a copy or download the key immediately and save it to a secure location since you won't be able to access this key on AWS or in Galera Manager once you close this box.

SSH Keys

The last two fields of the *Default Host Configuration* section are related to `ssh` enryption keys. To be clear, below is the screenshot from Figure 1 again, but cropped here around the part on SSH keys:

Fig. 14: SSH Keys for Default Host Configuration (Figure 8)

These encryption keys are used to access the hosts via SSH.

You should provide the private key in case you want Galera Manager to use a specific key to access the hosts, for example if you have set up corresponding public keys on the hosts. You need to supply the key if you have chosen *unmanaged* host type, otherwise it is not required: Galera Manager will generate its own key pair.

For you as administrator to access the host with `ssh`, you will need to provide one of your own public keys. Click on the icon of a key on the right, where it says, *Authorized Keys*. A box will appear like the one below in Figure 9 for you to paste in your public key from your own private key:

After pasting in the public key, click on the plus-sign icon to store it. The field will become empty again so that you may paste in another public key. You may want to paste in a public key for each person for whom you want to provide command-line superuser access to the hosts.

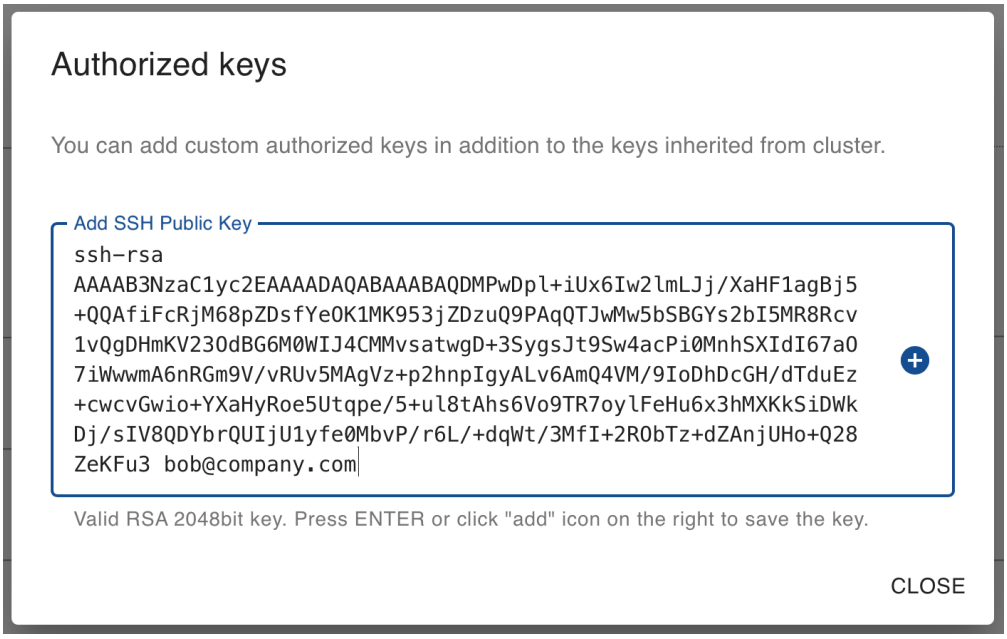


Fig. 15: Add Authorized Public SSH Keys (Figure 9)

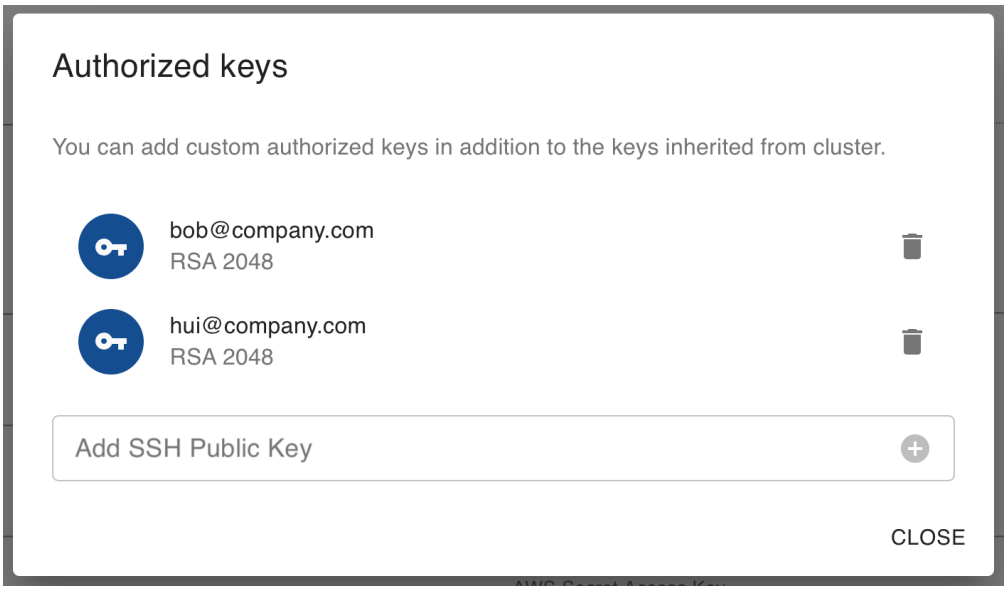


Fig. 16: Authorized Public SSH Keys (Figure 10)

Finishing Deployment

When you finish with all of your settings for the new cluster, click on the text in blue where it reads, *Create*. This will open a small box that should say it was successful, like the one below:

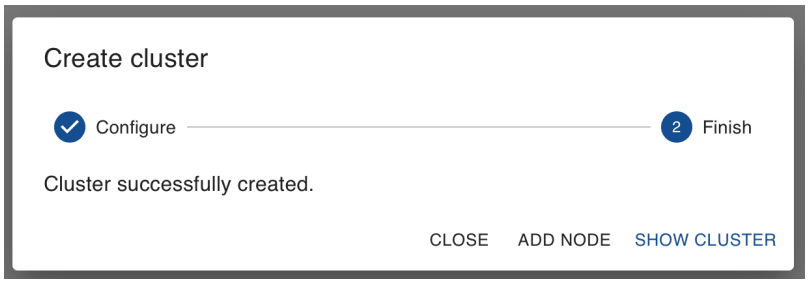


Fig. 17: Cluster Finished Creating (Figure 11)

There's not much to this because you haven't yet added nodes to the cluster. To learn about how to add nodes to a cluster, read the *Adding Nodes with Galera Manager* (page 175) documentation page. Check the *Adding Users to Galera Manager* (page 182) page on adding users, the *Loading Initial Data* (page 185) on adding data, and the *Monitoring a Cluster with Galera Manager* (page 191) page on configuring the metrics to track.

Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)
- *AWS Ports* (page 157)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)
- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Galera Manager Documents

- *Getting Started* (page 147)

- *Installing* (page 150)
- *AWS Ports* (page 157)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)
- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

6.3.6 Adding Nodes with Galera Manager

After you've created a cluster, set the defaults for nodes within Galera Manager, you will need to add nodes to that cluster. .. When you add nodes to a cluster, Galera Manager will add hosts on AWS (Amazon Web Services) and install all of the software needed, including either MySQL or MariaDB. It will then configure `mysqld` to be a node in the cluster.

If you haven't yet created a cluster, read the *Deploying a Cluster in Galera Manager* (page 167) page—installing Galera Manager is covered in the *Installing Galera Manager* (page 150) page.

Node & Host Deployment

To add nodes to a cluster, after logging into Galera Manager from a web browser, click on the cluster in the left margin. In the main panel, click then on the vertical ellipsis in the top right margin. When you do, a small box (see Figure 1 below) will offer you two choices: to add a node or to delete the cluster. Click on *Add Node*.

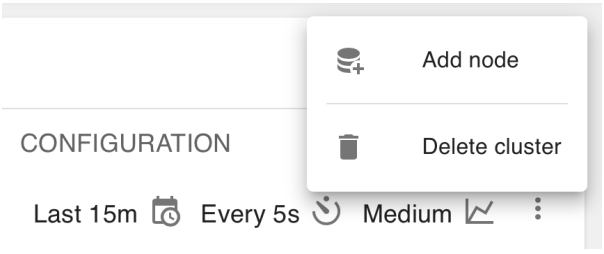


Fig. 18: Close-Up of Menu to Change a Cluster (Figure 1)

After you click *Add Node*, a large box like the one shown in the screenshot below in Figure 2 will appear. Here you will provide your preferences for the node or nodes, and the hosts you want to add.

The first field at the top left of the *Node Deployment Wizard* is to enter the number of nodes you want to add, depending on the host type of the node. If the host is managed by Galera Manager (for example EC2 host type), then Galera

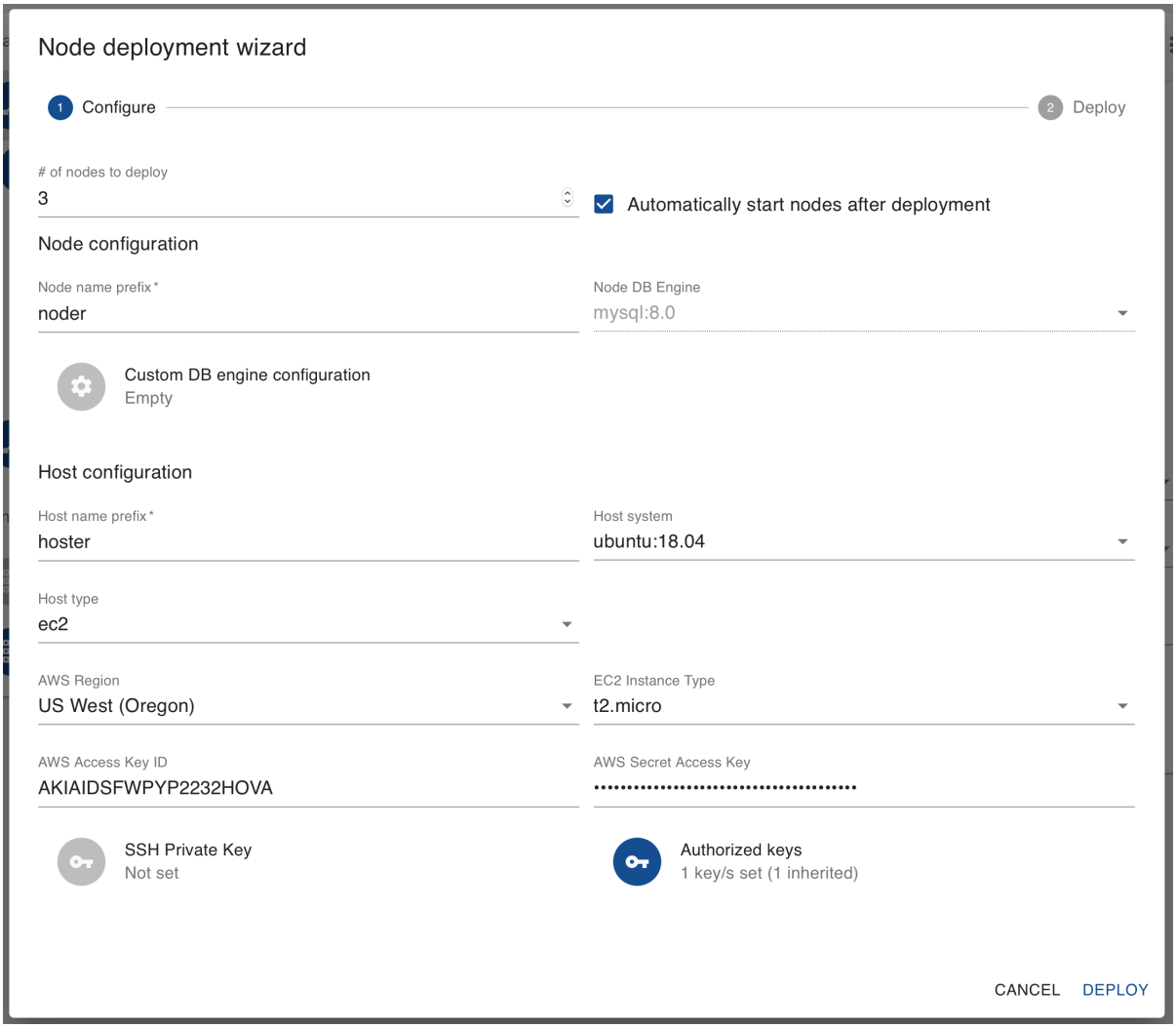


Fig. 19: Node Deployment for a Cluster (Figure 2)

Manager can automatically provision and set up several nodes at once. If hosts for the nodes are provided by the user (*unmanaged* host type), then each node will have to be added individually.

In the example here, we are creating a cluster in AWS EC2, so 3 has been entered. By default, the nodes will be started automatically after the hosts have been provisioned and then nodes set up.

Node Deployment Choices

Next, you will enter specific information on this node or set of nodes. To make discussing easier, below is the screenshot from Figure 2, but cropped around the default node configuration section:

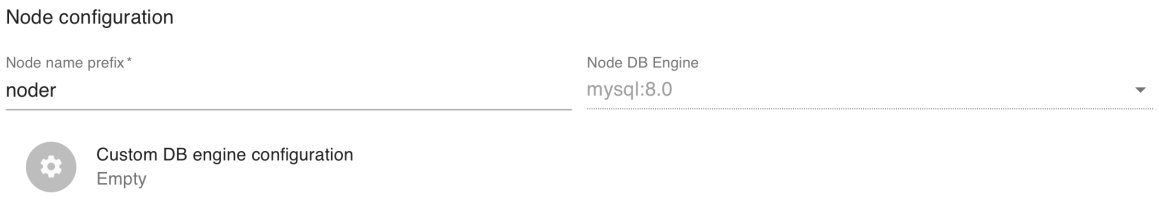


Fig. 20: Node Configuration (Figure 3)

At a minimum, you would enter the prefix for naming nodes. If you are creating only one node, what you enter here will be used. If you are creating multiple nodes, this text will be used as a prefix to each node's name. The suffix of the node name will be randomly generated. If it is important to you to name each node, you will need to add them one at a time to the cluster.

The database system and version is already set from when you created the cluster. You have to use the same database system for each node. However, although the custom database settings you might have added at that time will be passed to the nodes—if you are creating nodes one at a time—you may give one node extra settings depending on their hardware and operational purpose. You probably wouldn't do this with the initial set of nodes, but later when you are adding temporarily another node because of a surge in traffic, you might want the extra node to handle more traffic. Therefore, you may want to set its buffers and other settings to higher values. You can add those settings then for the one node.

Host Deployment Choices

The next part of the *Node Deployment Wizard*, shown in the cropped screenshot below, relates to configuring the hosts. By default host setting are inherited from the cluster values, but you can change them for particular host here. If you are adding a host that is not created by Galera Manager, here you will need to provide private SSH key for Galera Manager root access to the host. Host defaults are explained in the *Default Host Configuration* (page 169) section of the *Deploying a Cluster in Galera Manager* (page 167) documentation page.

Being able to make different choices for the host when adding nodes is particularly useful when adding nodes to an existing and running cluster. For example, if you are adding temporarily a node because of an increase in traffic, you might want to use a larger server. To do this, you would select a different *EC2 Instance Type*, one with more memory and processing power. If you want to migrate to a new release of Linux, you can add new nodes with that choices. After they've synchronized, you could then delete the old nodes.

Finishing Deployment

After you finish entering the number of nodes in the *Node Deployment Wizard*, and the node and host names, as well as any changes you want to make to the default settings, you would then click on *Deploy* in the right-hand corner. A small box, like the one below, will appear in which you can observe the progress of the hosts and nodes being

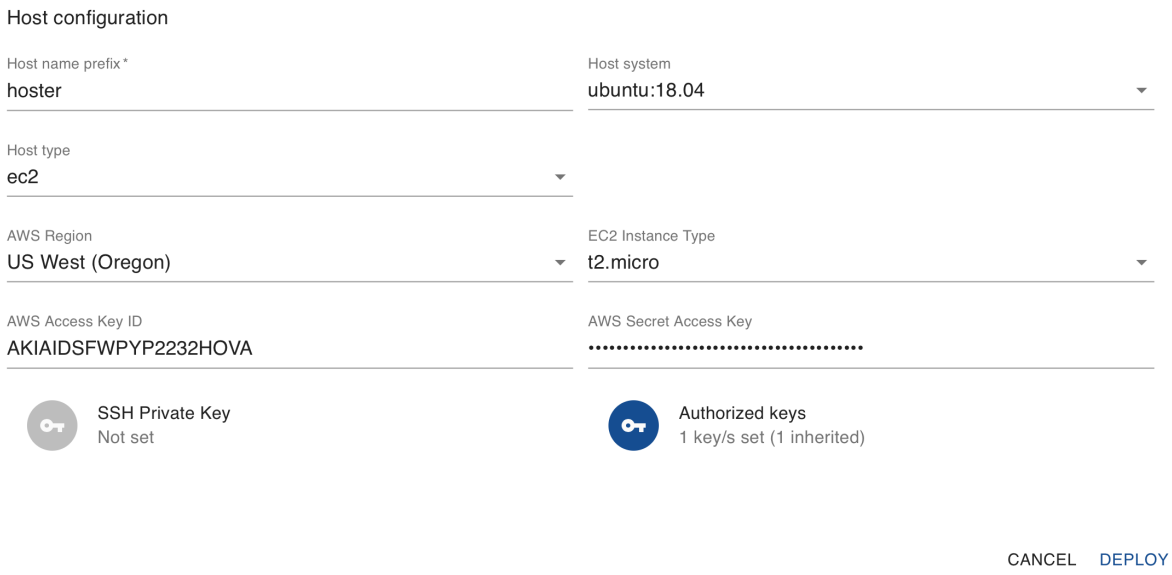


Fig. 21: Host Configuration (Figure 4)

deployed. Note, here we illustrate an example of adding nodes in AWS EC2, which involves automatic provisioning of EC2 instances for hosts, installing cluster and monitoring software, and finally starting up the nodes)

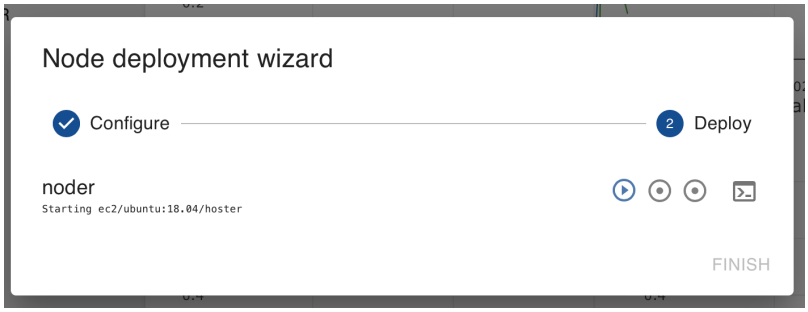


Fig. 22: Progress of Adding Nodes (Figure 5)

The deployment process may take some time. If it fails, you will see in the small red text at which point it failed. You can also check the *Logs* and *Jobs* tabs for the cluster and node for more information. When the node deployment succeeds, all of the circled-dot and right-arrow play buttons on the right (see Figure 5) will change to check marks and the *Finish* link will become active. Click on that link to close the box when it is done.

Finished Results

When the *Node Deployment Wizard* has finished running and you've closed the related box, you will see the nodes that were added listed in the left margin, under the name of the cluster. The results should look similar to the screenshot below in Figure 6 below:

Notice that although a node name of `noder` was entered, some extra text was added to make each node name unique (for example, `noder-jfebk`). As mentioned earlier, if you add one node at a time, you can name each and no suffix will be appended.

If you chose to have the nodes started automatically, they should all have a status of *Synced*. If one wasn't started automatically, click on the node in the left margin, and then click on the vertical ellipsis at the top right of the main

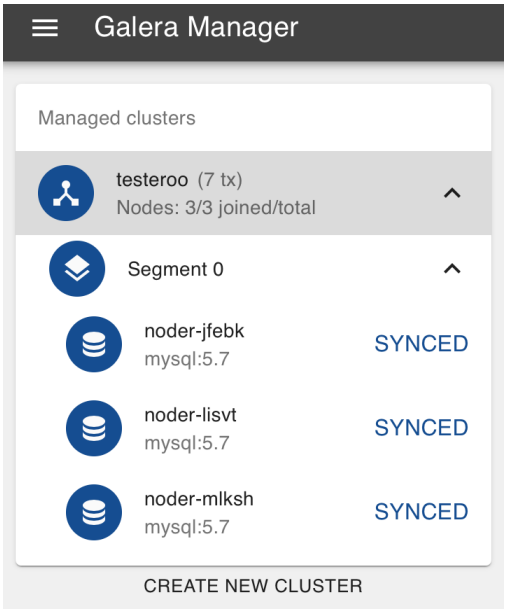


Fig. 23: Left Margin with Results of Adding Three Nodes (Figure 6)

panel. From the choices you are offered there, click *Start* to start the node.

Now that we created our cluster in AWS EC2, Galera Manager has provisioned a EC2 instance for each node's host. If you look in your EC2 console showing your *Instances*, you will see something like the screenshot below:

Name	Instance ID	Instance Type	Availability Zone	Instance State
main server	i-08549fee9b5ad66e0	t2.large	us-west-2b	running
testeroo:hoster-jfebk	i-0dadb1f2217ac0293	t2.micro	us-west-2a	running
testeroo:hoster-lisvt	i-062af632a22a8f72b	t2.micro	us-west-2a	running
testeroo:hoster-mlksh	i-0cb68a8471e400207	t2.micro	us-west-2a	running

Fig. 24: AWS Instances: Galera Manager and Three Hosts (Figure 7)

In this example, there's one *Instance*, on which Galera Manager is installed. There's an *Instance* for each node of the three in the cluster (for example, `hoster-jfebk`, etc.). .. You see the host names because that's the physical or virtual server on which the node and its software is running.

When you click on a node in the left margin of Galera Manager, you will see charts for monitoring the node's activities. To start, it will be fairly empty like the screenshot below:

At this point, the charts are rather featureless. However, as you start to add data, which is covered in *Loading Initial Data* (page 185) page of the documentation, you will start to see some activity. You can learn more about how to use these charts, as well as how to add other charts to track other metrics than these initial few, by reading the *Monitoring a Cluster with Galera Manager* (page 191) page. You may also want to add other users to Galera Manager who can monitor and add clusters and nodes. This is covered on the *Adding Users to Galera Manager* (page 182) page.

Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)

Fig. 25: New Node in Galera Manager (Figure 8)

- *AWS Ports* (page 157)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)
- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)
- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

6.3.7 Adding Users to Galera Manager

Galera Manager offers several features for the maintenance of Galera Clusters, saving you plenty of time in deploying nodes and similar tasks. However, you may want the assistance of other administrators, especially when you are not around to monitor your clusters. In anticipation of such situations and needs, you can add other users to the Galera Manager. Note, these are not users for the MySQL or MariaDB databases.

This page of the Codership documentation describes how to add users to Galera Manager. If you haven't already installed Galera Manager, go to the *The Galera Manager* (page 147) documentation page.

Adding Users

When you installed Galera Manager with the *Installer*, you were asked to specify a user name and password for the administrator. You were only allowed one user during installation. Now that Galera Manager is installed, you may add more users. Click on the menu icon, the three horizontal strips at the top left. It will reveal what you see in the screenshot of Figure 1 below:

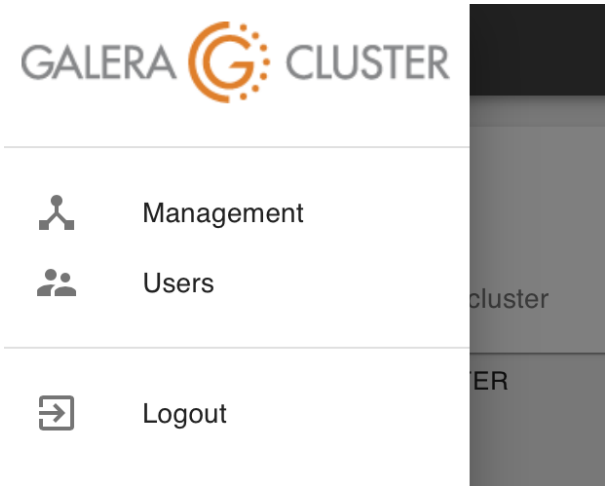


Fig. 26: Galera Manager Menu (Figure 1)

As you can see, there are three choices: *Management*, which brings you back to the primary screen for managing and monitoring Galera Clusters; *Users*, which is where you will add and remove users for Galera Manager; and *Logout*, which is to log out of Galera Manager. If you click on *Users*, you will see a box similar to the screenshot in Figure 2 below:

In the example in the screenshot above, a user name of *bob* was entered, along with a password. The other choice is to set the user's privileges: *None*, *Observer*, *Manager*, and *Administrator*.

The setting of *None* will create a user who may log in, but has no access, can see nothing. In the future it can be used to temporary block the user, but currently the users can only be created and deleted. A user designated as an *Observer* will be allowed to log in and monitor clusters and nodes, but not make any changes. A *Manager* will have all of the privileges of an *Observer*, but will also be allowed to add and delete clusters and nodes, as well as add and remove metrics to monitor. The *Administrator* can do everything, including adding and removing users.

Changing Users

After you've added some users, you will see them on the user page. To return to this page, click on the menu icon and then *Users*. You'll see a screen like the one in Figure 3 below:

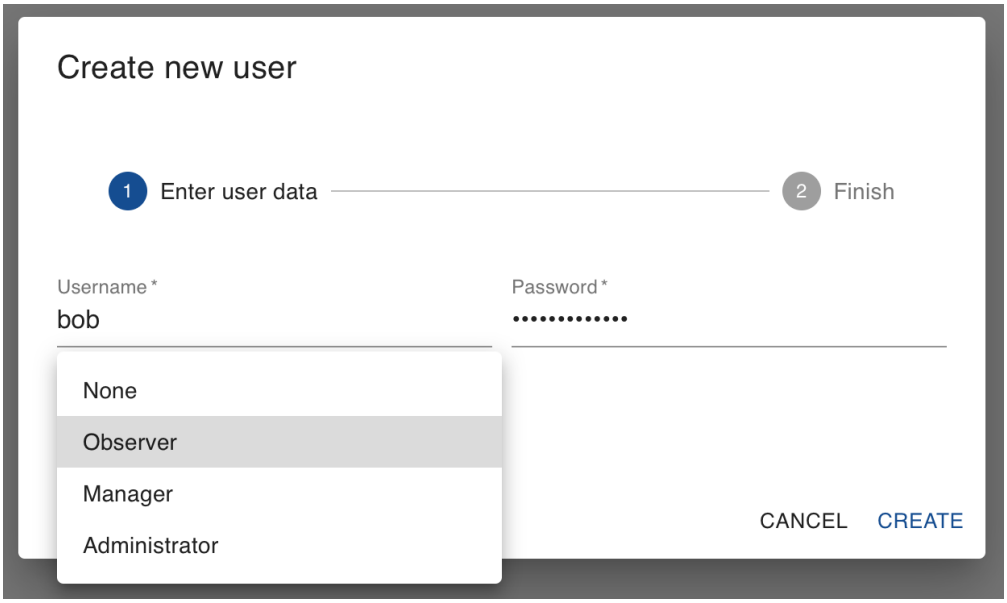


Fig. 27: Dialog Box for Adding Users (Figure 2)

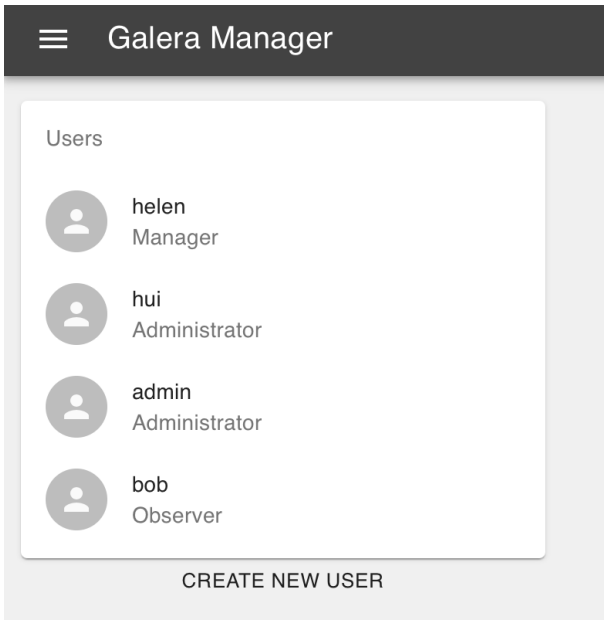


Fig. 28: Galera Manager Users (Figure 3)

You can, of course, add more users. Should you decide to delete a Galera Manager user or to change a user's role, click on that user in the list of users. You'll see in the main panel the same fields you were presented when you created the user, similar to the screenshot below in Figure 4:

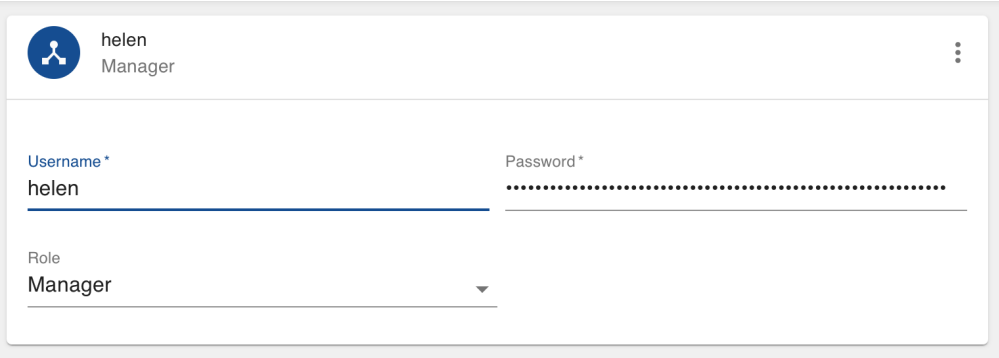


Fig. 29: Galera Manager User Information (Figure 4)

Click the vertical ellipsis at the top right of the panel for the user. Your only choice there will be to delete the user, so click it. To put the user back, click on the text below the list of users where it says, *Create New User* to add them again with new choices or values.

Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)
- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)

- *AWS Ports* (page 157)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)
- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)

Related Articles

- Galera Cluster Backups
- Migrating to Galera Cluster
- *Upgrading GM* (page 199)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

6.3.8 Loading Initial Data

After you create a Galera Cluster and add nodes with Galera Manager, you will probably want to load data from a previous database, to migrate data from another server or cluster. This is not a feature of Galera Manager since its main focus is the logistics and monitoring of clusters and nodes, not the data contained in the databases.

To load the initial data in a new Galera Cluster created within Galera Manager, you will have to use standard methods, of which there are a few. This page of the Codership documentation explains how to log into one of the hosts, and how to use common, basic methods to load data into a node.

If you are unfamiliar with how to make a back-up on an existing Galera Cluster, you could read the *Backup Cluster Data* (page 109) documentation page first. There are also links in the margin to tutorials on making back-ups and loading back-ups to a node of a new cluster—regardless of whether Galera Manager was used to create the cluster.

Methods to Load Initial Data

There are two common methods of loading data into MySQL or MariaDB: restoring from a logical or a physical back-up.

Loading Logically

Logical back-ups are generated with a utility like `mysqldump` and produce text files (that is, dump files) or streams containing SQL statements which may be used to rebuild databases. See the tutorial, Galera Cluster Backups for more details on how to use `mysqldump` to make a back-up. `mysqldump` creates a "dump" file or stream from a source MySQL server which then can be loaded to a new MySQL server using MySQL client utility (`mysql`).

If you will be restoring data by way of a MySQL client, you will need the node's IP address and the root password for the database. To get this information select a node in Galera Manager and then click on the *Configuration* tab for the node. You can see an example of this in the screenshot in Figure 1 below:

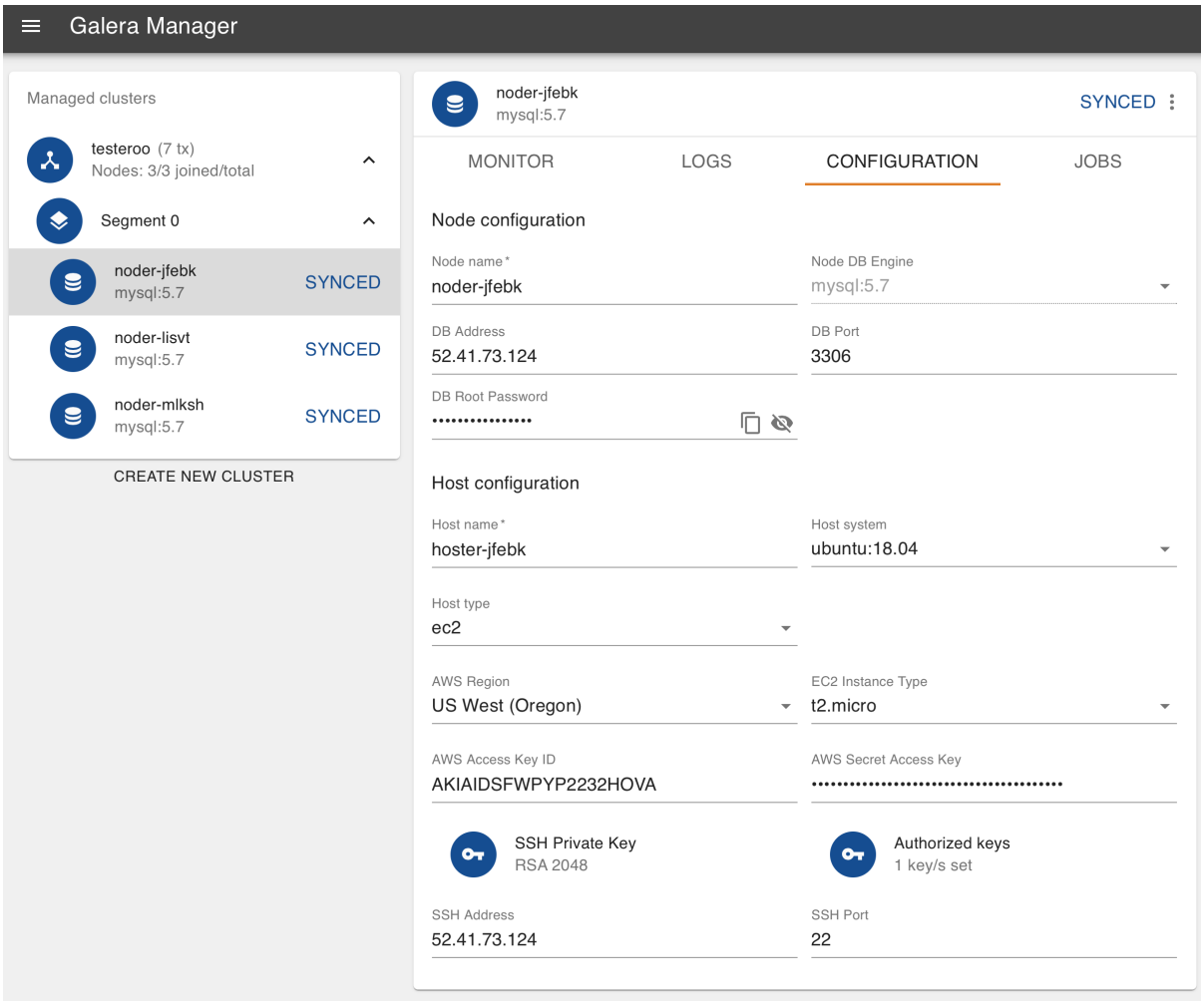


Fig. 30: Node Configuration (Figure 1)

In the main panel shown here, near the top left of the *Configuration* tab, in the *DB Address* field, is the external IP address of the node, which is `52.41.73.124` in this example. You'll also need the MySQL or MariaDB root password. Incidentally, since it is a new installation of MySQL or MariaDB, there's only the root user. To get the *DB Root Password*—as it is labeled here—click on the icon of an eye to reveal it, or click the icon of two sheets of paper to copy the password to your clipboard.

With the node's IP address and the password for root in the database, you can use a MySQL client to load data from a dump file. The example below shows how to restore a dump file made with `mysqldump`:

Listing 24: Load Data from a `mysqldump` File (Example 1)

```
mysql -p -u root -h 52.41.73.124 < company-20200607.sql
```

This line above would be executed on another server where the dump file (that is, `company-20200607.sql`) is located. The host address here is for the node into which it will be loading the data. When it asks for the password, you would give it the one you copied from the node in Galera Manager.

The dump file contains SQL statements that will execute `CREATE DATABASE` and `CREATE TABLE` statements, as well as plenty of `INSERT` statements to recreate the databases, tables, and rows of data on the new node—in the new Galera Cluster. It will run for quite a while, but when it is finished, you might execute an SQL statement like the following from the command-line:

Listing 25: Checking Loaded Data (Example 2)

```
mysql -p -u root -h 52.41.73.124 -e "SHOW TABLES FROM company"

+----------------------+
| Tables_in_company    |
+----------------------+
| clients              |
| clients_addresses    |
| clients_email        |
| clients_telephones   |
| employees            |
| employees_email      |
| employees_salaries   |
| employees_telephones |
| org_departments      |
| org_divisions        |
| org_offices          |
| org_warehouses       |
| ref_job_titles       |
| ref_name_titles      |
| ref_states           |
+----------------------+
```

The results table in Example 2 here shows that the `company` database was created and so were the tables for it. You could check further by executing some `SELECT` statements to ensure the data was inserted.

You might also go back to Galera Manager to see how the activity looks for your nodes. Below is a screenshot of Galera Manager that was taken shortly after loading the dump file above:

Notice the first chart at the top left for the cluster has no activity and then there's a spike of activity. There are three line graphs showing spikes because there are three nodes: one is the data being loaded from the `mysql` client and the other two nodes are replicating data that the first node is receiving.

Loading Physcially

The other common method of making back-ups is to use physical back-ups. This is fairly simple: it is mostly just a copy of MySQL's data directory. Typically, administrators use `rsync`, `xtrabackup` or `mariabackup` to make a back-up copy of the data directory and othere relavent files. Then they use `tar` and `gzip` to make a compressed archive file. See the tutorial, Galera Cluster Backups for more details on this process.

Copying and restoring the data from physical backups is normally much faster than using logical backup, the bigger the volume the bigger the difference. However restoring data from physical backup to Galera Cluster is quite tricky. The problem is that it can't be done on a running node, and as a consequence it goes without cluster being aware of it.

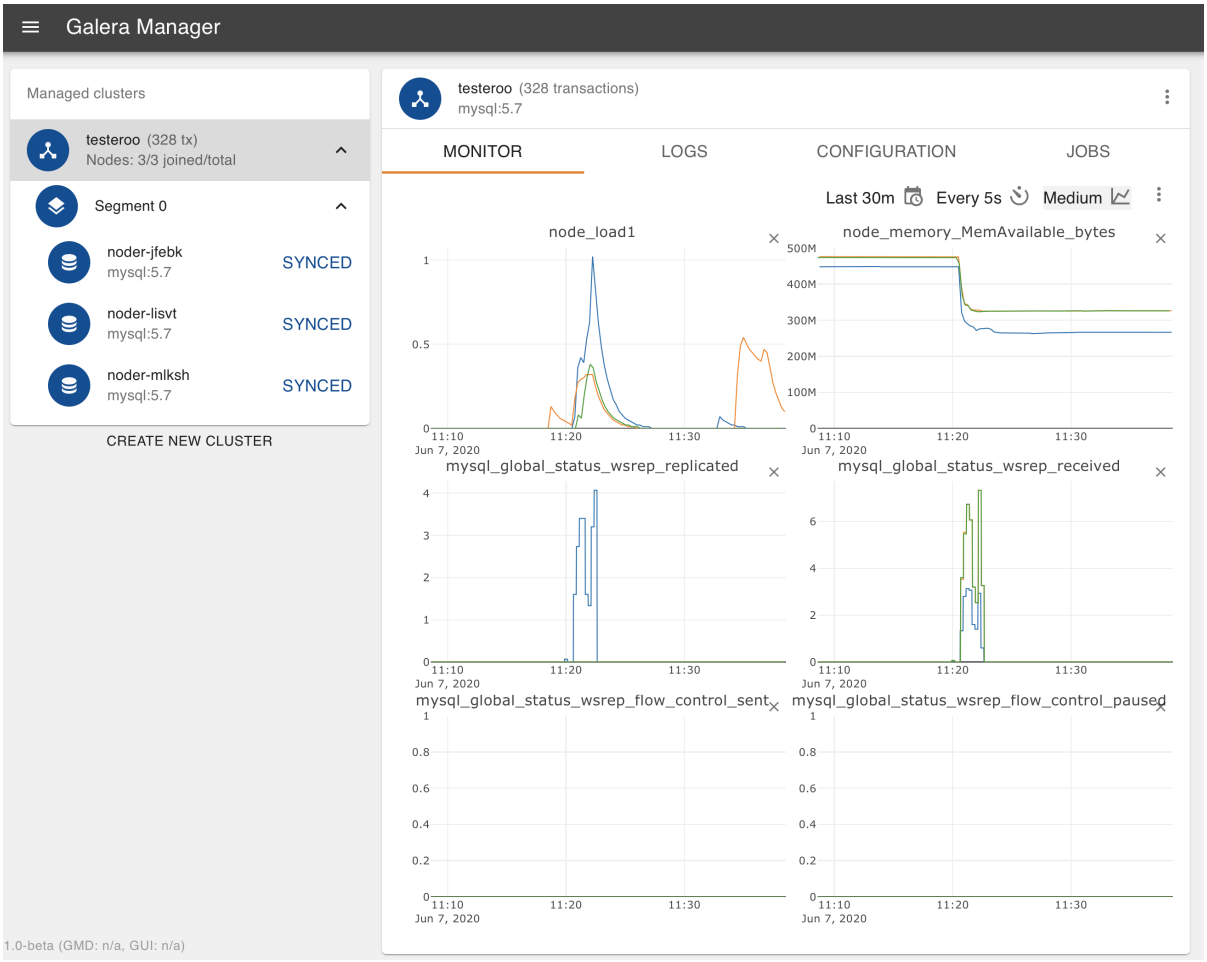


Fig. 31: Monitoring Data Loading on Nodes (Figure 2)

The easiest way to initialize Galera cluster from a physical backup is start with a single node cluster and after restoring the node from physical backup, add other nodes at will.

First, create a cluster and add a single node to it. Make sure to supply your public SSH key in the *Authorized Keys* section. You will need your private SSH key counterpart when accessing the host. When the node reaches `SYNCED` state, stop the node from Galera Manager: click on the node and then the vertical ellipsis at the top right. This will open a dialog box like the one below:

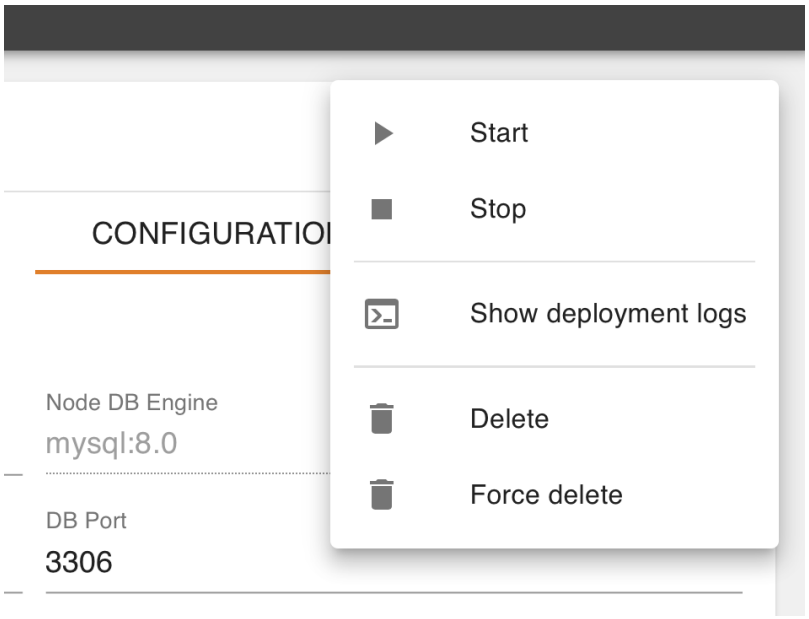


Fig. 32: Stopping a Node in Galera Manager (Figure 3)

When you click on *Stop*, the node process (`mysqld`) will be stopped, but the host will remain online.

To restore from a physical back-up, you will need to copy the back-up data to the host frist. This is where you will need the node's IP address from the node configuration tab mentioned the *Loading Logically* (page 185) section of this page, and a private SSH key that corresponds to the public key you supplied in the node creation box.

To copy the back-up file to the node, you can use `scp` to make a secure copy from the old server where the back-up is located, to the new node. First, you may want to log into the host. You could do that by entering something like the following from the command-line of a local computer:

Listing 26: Logging into a Node to Prepare to Load Data (Example 3)

```
ssh -i ~/.ssh/galera-manager root@52.41.73.124
```

The name of your private key and your node's IP address will be different. Notice it requires you use the user name, root. That's the only user since this is a new host.

Listing 27: Copying Back-Up Data from Remote Server (Example 4)

```
scp -i ~/.ssh/galera-manager /backups/backup-20200607.tgz root@52.41.73.124:/tmp/
```

This line uses `scp` to copy the back-up file from another Ubuntu server to the new node, to the `/tmp` directory. Now you can restore MySQL data directory from that backup. Details depend on how you created the backup. Please refer to the documentation on how to use that particular backup method to recover the data directory. In the most trivial case of backup being simply a tarball of the data directory:

Listing 28: Unzipping and Extracting Back-Up Data (Example 5)

```
tar -xvzf /tmp/backup-20200607.tgz -C /var/lib/mysql
chown -R mysql /var/lib/mysql
```

When you are finished, go back to Galera Manager and start the node. As soon as `mysqld` starts and shows `SYNCED` state, you can add more nodes, they will automatically copy data from the first one. You could execute a few SQL statements on one of the other nodes to see if they have the data, as shown in Example 2.

Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)
- *AWS Ports* (page 157)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)
- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)

Related Articles

- Galera Cluster Backups
- Migrating to Galera Cluster
- *Upgrading GM* (page 199)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)
- *AWS Ports* (page 157)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)

- *Adding Nodes* (page 175)
- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

6.3.9 Monitoring a Cluster with Galera Manager

There are perhaps two aspects of Galera Manager that are its best features and make it worthwhile for database administrators to use: First is the ability to add nodes easily with a graphical interface, and without having to configure each node manually. Second is the ability to monitor a cluster with a set of charts that can track many database and cluster metrics.

If you happened upon this page of the documentation first, without having installed Galera Manager, please read the *Installing Galera Manager* (page 150) page and install it. For those who have installed Galera Manager, but have not yet created a cluster with nodes using Galera Manager, read the *Adding Nodes with Galera Manager* (page 175) page.

Default Charts & Metrics

After first having created a cluster and added nodes, you will see a dashboard containing charts for tracking some metrics. Below in Figure 1 is a screenshot of how a cluster with three nodes would look at the start. However, these charts and monitored metrics are just a few that are loaded by default. You can add and remove any charts, monitor any database or cluster metrics you want.

As a starting point,six charts are configured for new installations. You may remove any or all of these charts. Before you do, you might want to consider what these initial ones track:

- **load_node_1** records the CPU load average. It is a standard metric commonly displayed by most load monitors in Linux. Essentially, it tells you how loaded the system is with tasks, tasks competing for CPU usage.
- **node_memory_MemAvailble_bytes** stores how much memory is available for each node.
- **mysql_global_status_wsrep_replicated** indicates the number of write-sets replicated from that node.
- **mysql_global_status_wsrep_received** is the number of write-sets received. Together with the number replicated, this would equal the total transaction rate on the node.
- **mysql_global_status_wsrep_flow_control_sent** provides the number of flow control events emitted by the node.
- **mysql_global_status_wsrep_flow_control_paused** records how much time replication on the node was paused in nano-seconds per second. A value of 1,000,000,000 would mean it was completely paused. This metric and the previous one are very important to troubleshoot replication performance concerns.

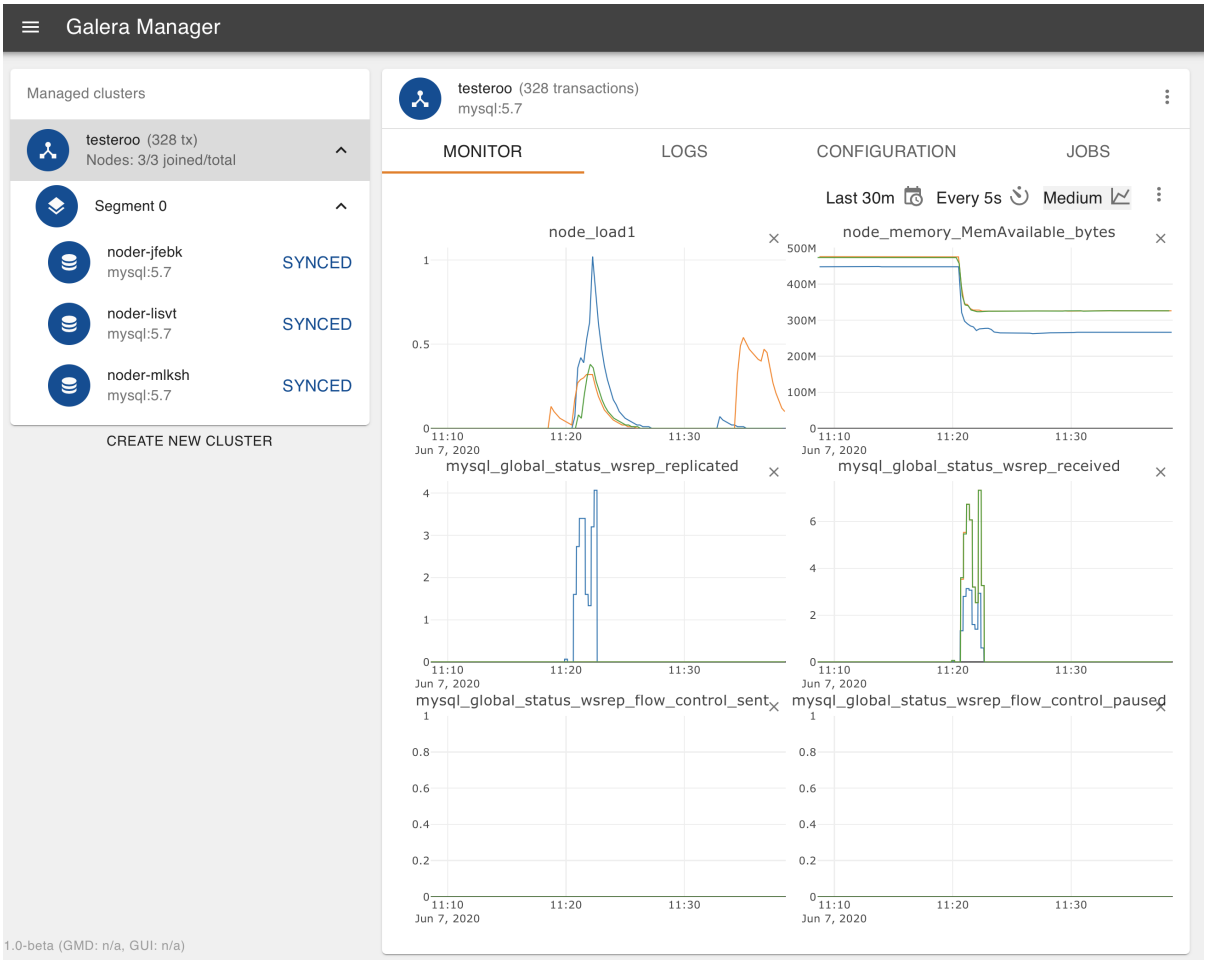


Fig. 33: Monitoring Data Loading (Figure 1)

Metric Names & Associations

The metrics come from the InfuxDB database and have a pattern to the names of metrics. Ones containing the word, *node* (for example, `load_node_1`), track the host metrics; in this context, it is a misnomer.

As for metrics with *mysql_global_status_* as the prefix, the stem is the name of the MySQL or MariaDB global status variable's name. For example, *mysql_global_status_wsrep_replicated* is from the MySQL global status variable, `wsrep_replicated`.

If for some reason you want to access MySQL database directly you would go to the *Configuration* tab for one of the nodes to get the IP address and password for the database system. Then you would enter something like the following from the command-line of a computer on which the `mysql` client is installed:

Listing 29: Checking a MySQL Global Status Variable (Example 1)

```
mysql -p -u root -h 34.216.245.12 /
   -e "SHOW GLOBAL STATUS LIKE 'wsrep_replicated'"

+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| wsrep_replicated | 7     |
+------------------+-------+
```

These results should agree with that of the chart in Galera Manager tracking this status variable. There's no need, though, for you to do this, to enter `SHOW GLOBAL STATUS` for every variable you want to monitor: you now have Galera Manager to do that for you. If there's a status variables you regularly check, you need only to add a chart in Galera Manager to monitor it.

Adding Metrics to Monitor

There are over one-thousand metrics that may be tracked. Some of them measure host operation: CPU usage, free memory, etc. Others come from the node's database engine, the `mysqld` process: the number of transactions committed, the amount of dirty pages in the buffer pool and so on.

To add a chart to the monitor in a cluster, click on the cluster in the left margin. Then in the *Monitor* tab, click on the vertical ellipsis at its top right—not the vertical ellipsis above it, not the one at the top right of the panel. See the screenshot in Figure 2 below:

Click on *Add Chart* and a box will appear like the one in Figure 3 below. There you will find all of the metrics you may track. Most are global status variables from MySQL, others are different host performance metrics—there are several at the bottom of the list. The data come from the `mysqld_exporter` daemon running on each host and gets aggregated locally on Galera Manager host for quick access.

Choosing a Metric

In the screenshot below, you can see the dialog box for choosing metrics to chart. Notice that metrics with the icon of a stack of disks are from the database engine (MySQL or MariaDB). Metrics tracking host performance are represented by the icon of a stack of servers.

You can either scroll through the long list of metrics, or you can enter a variable name in the search line to find it. If you do not remember the precise name of the variable, you may enter part of it (for example, `buffer`). This will present entries that match what was entered. You can then click on the one you want.

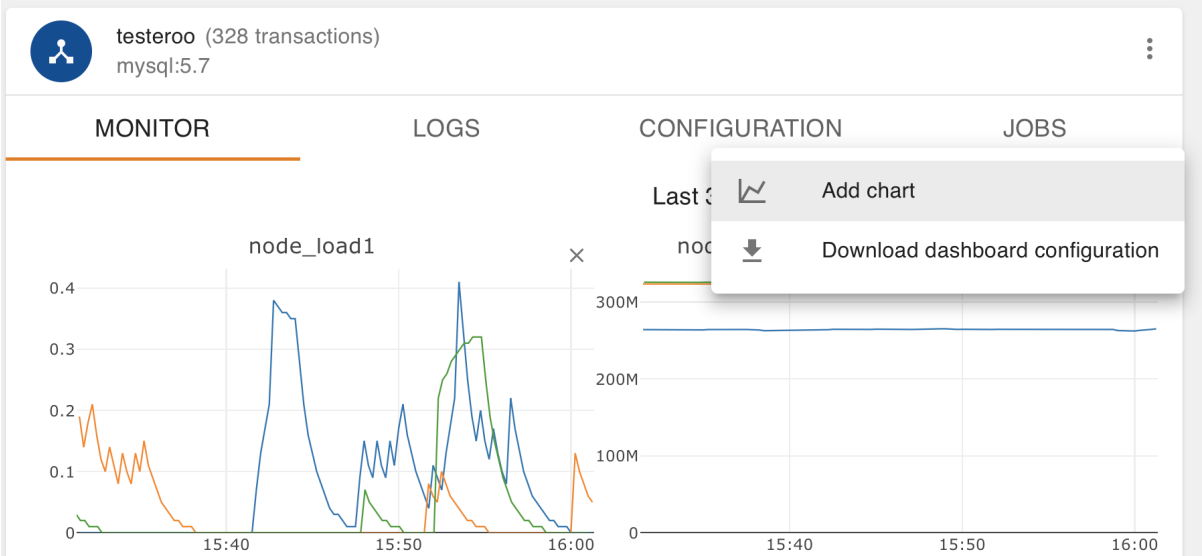


Fig. 34: Adding a Chart (Figure 2)

Cumulative or Differential

Some metrics show the total number of events since the process started (for example, the number of flow control events sent). As a result, its value keeps increasing. This sort of metric is called, *Cumulative*. If you choose such a metric, it will be shown on a chart in values per time interval (that is, per second) over the sampling interval. Other metrics are said to be *Differential* in that they are already in units per second.

Galera Manager is unaware of which metric is cumulative and which is differential. Therefore, you have to mark a chart as such by clicking the appropriate button. It is located in the box for adding a chart as shown in Figure 3 above, but hidden by the list of metrics in that screenshot. Below is the same dialog box, without the list of metrics, and cropped:

After you've chosen a metric and indicated whether it is cumulative or differential, click *Add* at the bottom right corner. You will be taken back to the monitor page and you will see a chart for the metric you added. You can reposition a chart by clicking it and holding down the mouse button, then dragging it where you want.

Changing a Monitor's Perspective

By default, the last fifteen minutes of the metric is shown in each chart, with the data refreshed every five seconds. This is meant to show activities that have just happened and happened recently. However, to determine a trend or a pattern, you may want to change the range of time and the refresh rate. You'll notice in the upper right corner of the main panel, above the charts, some selectors (see Figure 5 below). These may be used to change the perspective of a chart.

Clicking on the first icon of a calendar with a clock will allow you to change the amount of time displayed in the charts. You may choose a block of time in minutes (for example, thirty minutes), or a block in hours (for example, three hours), or for a period of days (for example, seven days). You can see the list of choices in the cropped screenshot in Figure 6 below:

You may also change the refresh rate to every second, or a less frequent amount of time (for example, every minute). You can see a list of choices available in the screenshot shown in Figure 7 below:

Finally, you can choose the relative size of the charts displayed: small, medium, or large. This will affect the arrangement of charts across and down. You might like to keep a separate computer running Galera Manager, continuously.

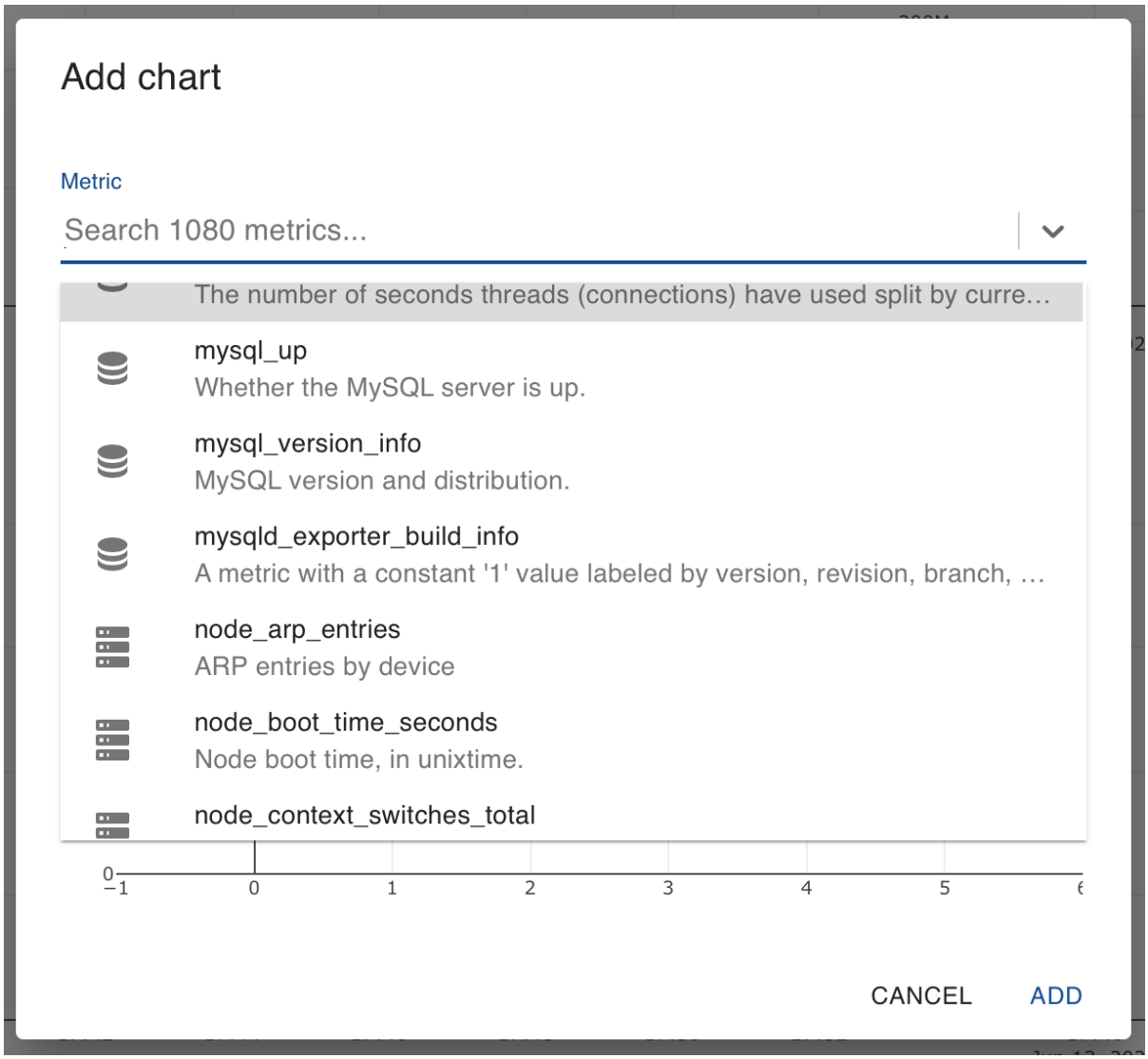


Fig. 35: Adding a Chart - Looking for a Metric (Figure 3)

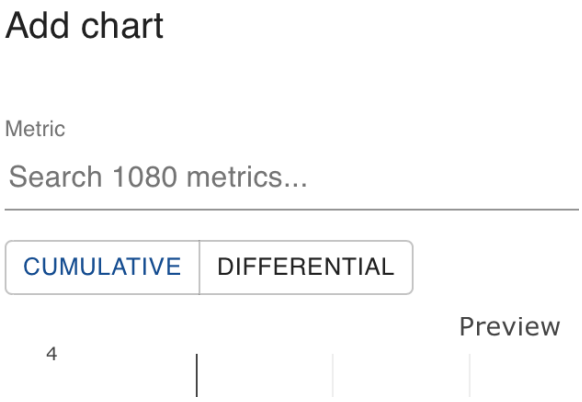


Fig. 36: Adding a Chart - Cumulative or Differential (Figure 4)

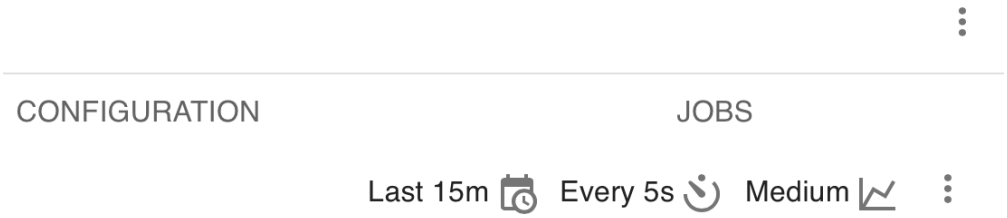


Fig. 37: Changing Perspectives (Figure 5)

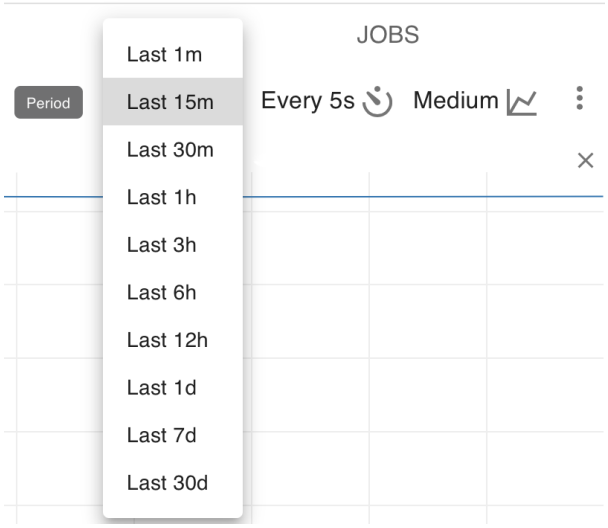


Fig. 38: Changing Time Period Displayed (Figure 6)

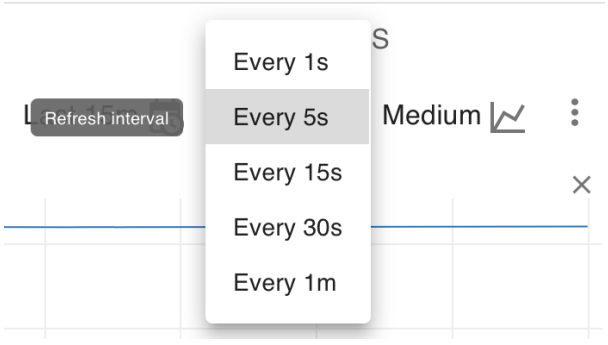


Fig. 39: Changing Refresh Rate of Data (Figure 7)

In which case, switching the web browser to full-screen with the charts set to large format would allow you to fill the screen and view the charts from a distance so as to get your attention quickly if a problem arises.

Preserving Chart Configuration

You may decide to make use of the default charts provided with Galera Manager, but if you decide to make changes, you may want to download a copy of the dashboard configuration. You can spend plenty of a time deciding on which metrics to monitor, and how to configure the charts. It would be frustrating to lose your charts configuration.

To preserve the chart configuration, click on the cluster in the left margin. Then in the *Monitor* tab, click on the vertical ellipsis at the top right within that panel. You'll see this pull-down menu, the same one you used to add charts:

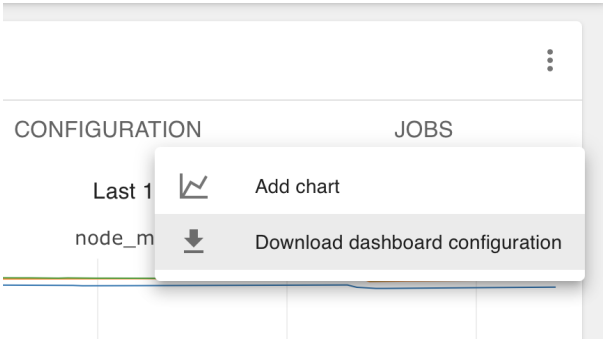


Fig. 40: Downloading Dashboard Configuration (Figure 8)

Click on *Download Dashboard Configuration* and Galera Manager will generate a dashboard configuration file and download it, saving it wherever you have set your web browser to download files. The file will be a json file and named, `cluster-*name*.json`, where *name* is the name of your cluster (for example, `cluster-testeroo.json`). Below is an excerpt from an example file:

Listing 30: Excerpt from an Exported Galera Manager Configuration (Example 2)

```
{"name":"cluster-testeroo","config":
  {"charts":
    [{"id":"301186ce-7b7f-41bb-a457-60696aeabba8",
      "name":"mysql_global_status_wsrep_received",
      "metric":"mysql_global_status_wsrep_received",
      "position":3,
      "resolution":"5s",
      "aggregation":"differential"},
   ...
   ...],
"tileSize":"md","refreshInterval":5000,"period":"15m"}}
```

This file excerpt has been reformatted with hard-returns and spaces to make it easier to view and follow—plus, most of it has been removed and replaced with ellipsis for brevity. But you can get a sense of how the information is stored—especially if you are familiar with json file structures.

This is a nice feature, being able to download the configuration file. However, at this point, the ability to upload a json configuration file is not yet available: Galera Manager is still in its early days. In a future release, though, you should be able to do this from Galera Manager.

Resolving Problems & Making Improvements

Galera Manager is an excellent tool for detecting issues early or potential ones with a Galera Cluster—hopefully, before they can become a problem. It can also assist in determining when and how performance can be improved before there is a slowing of database traffic or a loss of service.

Should you have problems with Galera Manager, you can check its log files. See the *gmd Logs* (page 163) section of the *Galera Manager Daemon (gmd)* (page 163) documentation page for information on those logs.

When you encounter a problem with a Galera Cluster, besides reading Codership's *Documentation* (page 1), you can look through Codership's Knowledge Base. When you can't find a solution on your own, or at least not quickly enough, you can contact Codership Support & Consulting.

Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)
- *AWS Ports* (page 157)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)
- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)
- *AWS Ports* (page 157)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)
- *Adding Users* (page 182)

- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

6.3.10 Upgrading Galera Manager (gmd)

Galera Manager Version

As new releases of Galera Manager software is released by Codership, you may update your installation using `yum` or `apt-get`, depending on your distribution of Linux. When you installed Galera Manager, a repository file will have been added to the repository directory: `galera.repo` in `/etc/yum.repos.d` on servers using `yum`; and `galera-manager.list` in `/etc/apt/sources.list.d` on servers using `apt-get`. These repository files will contain the address of the Codership repository, along with some related information.

To see which version and release of Galera Manager you are using, click on the menu icon, the three horizontal strips at the top left. It will reveal what you see in the screenshot of Figure 1 below:

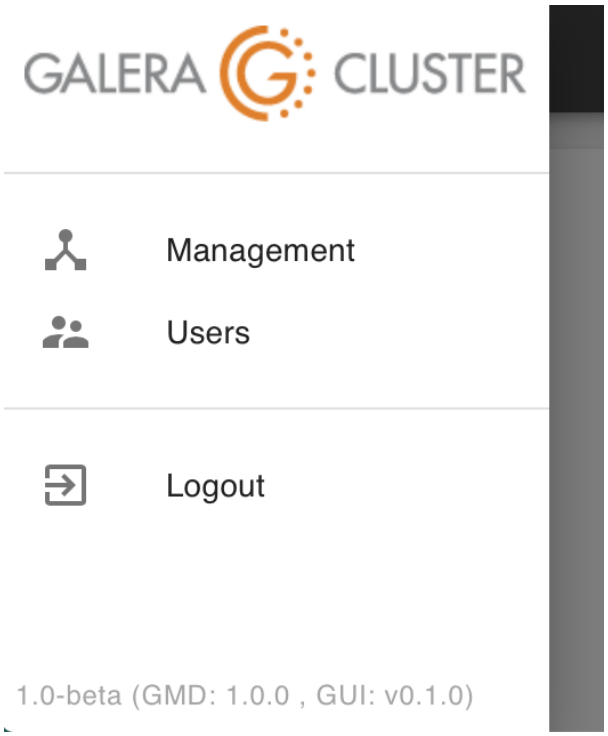


Fig. 41: Galera Manager Menu with Version and Release Number (Figure 1)

In this example screenshot, you can see in the subdued text at the bottom that this installation of Galera Manager is the beta version 1.0. The `gmd` is version 1.0.0, and the graphical user interface is version 0.1.0. You do not need to

keep track of those numbers, but when you read about a new Galera Manager feature offered in these documentation pages, but that you do not have in your installation, you can check your versions to see if maybe you need to upgrade Galera Manager.

Updating Galera Manager

You wouldn't run the *Galera Manager Installer* again to upgrade—reinstalling is not permitted by the *Installer*. Instead, you would use whatever package management utility (that is, `apt-get` or `yum`) is used in Galera Manager host operating system.

When running updates of your server, Galera Manager software will be included. However, if you want to upgrade specifically the Galera Manager software, you can do so like this on a server using the `yum` package management utility:

Listing 31: Method to Upgrade Galera Manager with `yum` (Example 1)

```
yum upgrade galera-manager
```

This will upgrade the Galera Manager software, but you might be asked to upgrade also any related libraries it uses. Unless there would be a problem with those upgrades for other software you are using on your server, cooperate with the requests to upgrade the other packages.

Here's how you would upgrade Galera Manager on a server using `apt-get`:

Listing 32: Method to Upgrade Galera Manager with `apt-get` (Example 2)

```
apt-get update
apt-get --only-upgrade install galera-manager
```

You would do this only on the server running Galera Manager, not on the hosts used for the Galera Cluster nodes.

Once the upgrades are finished, `gmd` will be restarted automatically. You might need to refresh your web browser, though, if you are logged into Galera Manager at the time. All of your settings, as well as your cluster and nodes should remain—including the databases and their data. Your Galera Clusters operation is not affected by `gmd` upgrade or failure.

Galera Manager Documents

- *Getting Started* (page 147)
- *Installing* (page 150)
- *AWS Ports* (page 157)
- *gmd Daemon* (page 163)
- *Deploying Clusters* (page 167)
- *Adding Nodes* (page 175)
- *Adding Users* (page 182)
- *Loading Data* (page 185)
- *Monitoring a Cluster* (page 191)
- *Upgrading* (page 199)

The Library

- *Documentation* (page 1)

- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Notification Script* (page 203)
- *wsrep_node_incoming_address* (page 249)
- *wsrep_node_name* (page 250)
- *wsrep_notify_cmd* (page 250)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

6.4 Notification Command

You can use the database client (for example, `mysql` client) to check the status of a cluster, individual nodes, and the health of replication. However, you may find it counterproductive to log in on each node to run such checks.

As an alternative and better method, Galera Cluster provides a method to call a notification script. Such a script may be customized to automate the monitoring process of a cluster.

Note: For an example of such a custom script and related instructions, see *Notification Script Example* (page 203).

Notification Parameters

When a node registers a change in itself or the cluster, it will trigger the notification script or command. In so doing, it will pass certain parameters to notification script. Below is a list of them and their basic meaning:

- `--status` The node passes a string indicating its current state. For a list of the strings it uses, see *Node Status Strings* (page 202) below.
- `--uuid` The node passes a string, *yes* or *no*, to indicate whether it considers itself part of the *Primary Component*.

- `--members` The node passes a list of the current cluster members. For more information on the format of these, see *Member List Format* (page 202) below.
- `--index` The node passes a string that indicates its index value in the membership list.

You will have to include code in the notificaiton script to capture the values of these parameters and then have the script act as you wish (for example, notify you of certain values).

Only nodes in the `Synced` state will accept connections from the cluster. For more information on node states, see *Node State Changes* (page 24).

Node Status Strings

The notification script may pass one of six values for the `--status` parameter to indicate the current state of the node:

- `Undefined` indicates a starting node that is not part of the Primary Component.
- `Joiner` indicates a node that is part of the Primary Component and is receiving a state snapshot transfer.
- `Donor` indicates a node that is part of the Primary Component and is sending a state snapshot transfer.
- `Joined` indicates a node that is part of the Primary Component and is in a complete state and is catching up with the cluster.
- `Synced` indicates a node that is syncrhonized with the cluster.
- `Error` indicates that an error has occurred. This status string may provide an error code with more information on what occurred.

Again, you will have to prepare your script to capture the value of the `--status` parameter and act accordingly.

Members List Format

The notification script will pass with the `--member` parameter, a list containing entries for each node connected to the cluster component. For each entry in the list the node uses this format:

```
<node UUID> / <node name> / <incoming address>
```

- **Node UUID** refers to the unique identifier the node received from the wsrep Provider.
- **Node Name** refers to the node name, as it is defined with the *wsrep_node_name* (page 250) parameter in the configuration file.
- **Incoming Address** refers to the IP address for client connections, as set with the *wsrep_node_incoming_address* (page 249) parameter in the configuration file. If this is not set, then the default value will be `AUTO`.

Enabling the Notification Script

You can enable your notification script or command through the *wsrep_notify_cmd* (page 250) parameter in the configuration file. Below is an excerpt from that file showing how it might look:

```
wsrep_notify_cmd=/path/wsrep_notify.sh
```

The node will call the script for each change in cluster membership and node status. You can use these status changes in configuring load balancers, raising alerts or scripting for any other situation in which you need your infrastructure to respond to changes to the cluster.

Galera Cluster provides a default script, `wsrep_notify.sh`, for you to use in handling notifications or as a starting point in writing your own custom notification script.

Related Documents

- *Notification Script* (page 203)
- *wsrep_node_incoming_address* (page 249)
- *wsrep_node_name* (page 250)
- *wsrep_notify_cmd* (page 250)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *wsrep_notify_cmd* (page 250)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

6.5 Notification Script Example

Nodes can call a notification script when changes happen in the membership of the cluster, that is when nodes join or leave the cluster. You can specify the name of the script the node calls using the *wsrep_notify_cmd* (page 250). While you can use whatever script meets the particular needs of a deployment, you may find it helpful to consider the example below as a starting point.

```
#!/bin/sh -eu

# This is a simple example of wsrep notification script (wsrep_notify_cmd).
# It will create 'wsrep' schema and two tables in it: 'membership' and 'status'
# and insert data into them on every membership or node status change.
#
# Edit parameters below to specify the address and login to server.

USER=root
PSWD=rootpass
HOST=<host_IP_address>
```

(continues on next page)

(continued from previous page)

```
PORT=3306

SCHEMA="wsrep"
MEMB_TABLE="$SCHEMA.membership"
STATUS_TABLE="$SCHEMA.status"

BEGIN="
   SET wsrep_on=0;
   DROP SCHEMA IF EXISTS $SCHEMA; CREATE SCHEMA $SCHEMA;
   CREATE TABLE $MEMB_TABLE (
      idx  INT UNIQUE PRIMARY KEY,
      uuid CHAR(40) UNIQUE, /* node UUID */
      name VARCHAR(32),     /* node name */
      addr VARCHAR(256)     /* node address */
   ) ENGINE=MEMORY;
   CREATE TABLE $STATUS_TABLE (
      size   INT,      /* component size   */
      idx    INT,      /* this node index  */
      status CHAR(16), /* this node status */
      uuid   CHAR(40), /* cluster UUID */
      prim   BOOLEAN   /* if component is primary */
   ) ENGINE=MEMORY;
   BEGIN;
   DELETE FROM $MEMB_TABLE;
   DELETE FROM $STATUS_TABLE;
"
END="COMMIT;"

configuration_change()
{
   echo "$BEGIN;"

   local idx=0

   for NODE in $(echo $MEMBERS | sed s/,/\ /g)
   do
      echo "INSERT INTO $MEMB_TABLE VALUES ( $idx, "
      # Don't forget to properly quote string values
      echo "'$NODE'" | sed  s/\\//\',\'/g
      echo ");"
      idx=$(( $idx + 1 ))
   done

   echo "
      INSERT INTO $STATUS_TABLE
      VALUES($idx, $INDEX,'$STATUS', '$CLUSTER_UUID', $PRIMARY);
   "

   echo "$END"
}

status_update()
{
   echo "
      SET wsrep_on=0;
      BEGIN;
      UPDATE $STATUS_TABLE SET status='$STATUS';
```

(continues on next page)

(continued from previous page)

```
        COMMIT;
    "
}

COM=status_update # not a configuration change by default

while [ $# -gt 0 ]
do
    case $1 in
        --status)
            STATUS=$2
            shift
            ;;
        --uuid)
            CLUSTER_UUID=$2
            shift
            ;;
        --primary)
            [ "$2" = "yes" ] && PRIMARY="1" || PRIMARY="0"
            COM=configuration_change
            shift
            ;;
        --index)
            INDEX=$2
            shift
            ;;
        --members)
            MEMBERS=$2
            shift
            ;;
            esac
            shift
    done

# Undefined means node is shutting down
if [ "$STATUS" != "Undefined" ]
then
    $COM | mysql -B -u$USER -p$PSWD -h$HOST -P$PORT
fi

exit 0
```

Path and Permissions

After you modify this script to fit your requirements, you need to move it into a directory in the `$PATH` or the binaries directory for your system. On Linux, the binaries directory is typically at `/usr/bin`, while on FreeBSD it is at `/usr/local/bin`.

```
# mv my-wsrep-notify.sh /usr/bin
```

In addition to this, given that the notification command contains your root password, change the ownership to the `mysql` user and make sure the script is executable only by that user.

```
# chown mysql:mysql /usr/bin/my-wsrep-notify.sh
# chmod 700 /usr/bin/my-wsrep-notify.sh.
```

This ensures that only the `mysql` user can execute and read the notification script, preventing all other users from seeing the root password.

Related Documents

- *wsrep_notify_cmd* (page 250)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- Configure Firewall
- Disable SELinux
- *Firewall Settings* (page 208)
- *SELinux Configuration* (page 225)
- *SSL Settings* (page 215)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

CHAPTER

SEVEN

SECURITY

On occasion, you may want or need to enable degrees of security that go beyond the basics of Unix file permissions and secure database management.

For situations such as these, you can secure both node communications and client connections between the application servers and the cluster.

- *Firewall Settings* (page 208)

  In order to use Galera Cluster, nodes must have access to a number of ports to maintain network connectivity with the cluster. While it was touched upon briefly in the Installation section, this section provides more detailed guides on configuring a system firewall using `iptables`, FirewallD and PF.

- *SSL Settings* (page 215)

  To secure communications between nodes and from the application servers, you can enable encryption through the SSL protocol for client connections, replication traffic, State Snapshot Transfers and Incremental State Transfers. This section provides guidance to configuring SSL on Galera Cluster.

- *SELinux Configuration* (page 225)

  Without proper configuration, SELinux can either block nodes from communicating or it can block the database server from starting at all. When it does so, it causes the given process to fail silently, without any notification sent to standard output or error as to why. While you can configure SELinux to permit all activity from the database server, (as was explained in the Installation section, this is not a good long-term solution.

  This section provides a guide to creating an SELinux security policy for Galera Cluster.

Related Documents

- Configure Firewall
- Disable SELinux
- *Firewall Settings* (page 208)
- *SELinux Configuration* (page 225)
- *SSL Settings* (page 215)

The Library

- *Documentation* (page 1)
- Knowledge Base

- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

7.1 Firewall Settings

Galera Cluster requires a number of ports to maintain network connectivity between the nodes. Depending on your deployment, you may not require all of these ports, but a cluster might require all of them on each node. Below is a list of these ports and their purpose:

- `3306` is the default port for MySQL client connections and *State Snapshot Transfer* using `mysqldump` for backups.
- `4567` is reserved for Galera *Cluster Replication* traffic. Multicast replication uses both TCP and UDP transport on this port.
- `4568` is the port for *Incremental State Transfer*.
- `4444` is used for all other *State Snapshot Transfer*.

How these ports are enabled for Galera Cluster can vary depending upon your operating system distribution and what you use to configure the firewall.

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Making Firewall Changes Persistent* (page 210)
- Home
- *Docs* (page 1)

- KB
- Training
- FAQ

7.1.1 Firewall Configuration with `iptables`

Linux provides packet filtering support at the kernel level. Using `iptables` and `ip6tables` you can set up, maintain and inspect tables of IPv4 and IPv6 packet filtering rules.

There are several tables that the kernel uses for packet filtering and within these tables are chains that it match specific kinds of traffic. In order to open the relevant ports for Galera Cluster, you need to append new rules to the `INPUT` chain on the filter table.

Opening Ports for Galera Cluster

Galera Cluster requires four ports for replication. There are two approaches to configuring the firewall to open these `iptables`. The method you use depends on whether you deploy the cluster in a LAN environment, such as an office network, or if you deploy the cluster in a WAN environment, such as on several cloud servers over the internet.

LAN Configuration

When configuring packet filtering rules for a LAN environment, such as on an office network, there are four ports that you need to open to TCP for Galera Cluster and one to UDP transport to enable multicast replication. This means five commands that you must run on each 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 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 tcp --match tcp --dport 4444 \
   --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
```

These commands open the relevant ports to TCP and UDP transport. It assumes that the IP addresses in your network begin with 192.168.0.

> **Warning:** The IP addresses in the example are for demonstration purposes only. Use the real values from your nodes and netmask in your `iptables` configuration.

Galera Cluster can now pass packets through the firewall to the node, but the configuration reverts to default on reboot. In order to update the default firewall configuration, see *Making Firewall Changes Persistent* (page 210).

WAN Configuration

While the configuration shown above for LAN deployments offers the better security, only opening those ports necessary for cluster operation, it does not scale well into WAN deployments. The reason is that in a WAN environment the IP addresses are not in sequence. The four commands to open the relevant ports to TCP would grow to four commands per node on each node. That is, for ten nodes you would need to run four hundred `iptables` commands across the cluster in order to set up the firewall on each node.

Without much loss in security, you can instead open a range of ports between trusted hosts. This reduces the number of commands to one per node on each node. For example, firewall configuration in a three node cluster would look something like:

```
# 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
```

When these commands are run on each node, they set the node to accept TCP connections from the IP addresses of the other cluster nodes.

Warning: The IP addresses in the example are for demonstration purposes only. Use the real values from your nodes and netmask in your `iptables` configuration.

Galera Cluster can now pass packets through the firewall to the node, but the configuration reverts to default on reboot. In order to update the default firewall configuration, see *Making Firewall Changes Persistent* (page 210).

Making Firewall Changes Persistent

Whether you decide to open ports individually for LAN deployment or in a range between trusted hosts for a WAN deployment, the tables you configure in the above sections are not persistent. When the server reboots, the firewall reverts to its default state.

For systems that use `init`, you can save the packet filtering state with one command:

```
# service save iptables
```

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

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

Once you find where your system stores the rules file, use `iptables-save` to update the file:

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

When your system reboots, it now reads this file as the default packet filtering rules.

Related Documents

- *Making Firewall Changes Persistent* (page 210)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

7.1.2 Firewall Configuration with FirewallD

The firewall daemon, or FirewallD, is an interface for dynamically managing firewalls on Linux operating systems. It allows you to set up, maintain and inspect IPv4 and IPv6 firewall rules.

FirewallD includes support for defining zones. This allows you to set the trust level of a given network connection or interface. For example, when deploying nodes that connect to each other over the internet–rather than a private network–you might configure your firewall around the `public` zone. This assumes that other computers on the network are untrusted and only accept designated connections.

For more information on FirewallD, see the Documentation.

Opening Ports for Galera Cluster

Galera Cluster requires four open ports for replication over TCP. To use multicast replication, it also requires one for UDP transport. In order for this to work over FirewallD, you also need to add the database service to the firewall rules.

To enable the database service for FirewallD, you would enter something like the following at the command-line:

```
# firewall-cmd --zone=public --add-service=mysql
```

Next, you will need to open the TCP ports for Galera Cluster. Do this by executing the following from the command-line:

```
# firewall-cmd --zone=public --add-port=3306/tcp
# firewall-cmd --zone=public --add-port=4567/tcp
# firewall-cmd --zone=public --add-port=4568/tcp
# firewall-cmd --zone=public --add-port=4444/tcp
```

Optionally, if you would like to use multicast replication, execute the following from the command-line to open UDP transport on `4567`:

```
# firewall-cmd --zone=public --add-port=4567/udp
```

These commands dynamically configure FirewallD. Your firewall will then permit the rest of the cluster to connect to the node hosted on the server. Repeat the above commands on each server. Keep in mind, changes to the firewall made by this method are not persistent. When the server reboots, FirewallD will return to its default state.

Making Firewall Changes Persistent

The commands given in the above section allow you to configure FirewallD on a running server and update the firewall rules without restarting. However, these changes are not persistent. When the server restarts, FirewallD reverts to its default configuration. To change the default configuration, a somewhat different approach is required:

First, enable the database service for FirewallD by entering the following from the command-line:

```
# firewall-cmd --zone=public --add-service=mysql \
      --permanent
```

Now, you will need to open the TCP ports for Galera Cluster. To do so, enter the following lines from the command-line:

```
# firewall-cmd --zone=public --add-port=3306/tcp \
      --permanent
# firewall-cmd --zone=public --add-port=4567/tcp \
      --permanent
# firewall-cmd --zone=public --add-port=4568/tcp \
      --permanent
# firewall-cmd --zone=public --add-port=4444/tcp \
      --permanent
```

If you would like to use multicast replication, execute the following command. It will open UDP transport on `4567`.

```
# firewall-cmd --zone=public --add-port=4567/udp \
      --permanent
```

Now you just need to reload the firewall rules, maintaining the current state information. To do this, executing the following:

```
# firewall-cmd --reload
```

These commands modify the default FirewallD settings and then cause the new settings to take effect, immediately. FirewallD will then be configured to allow the rest of the cluster to access the node. The configuration remains in effect after reboots. You'll have to repeat these commands on each server.

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos

- FAQ
- search
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

7.1.3 Firewall Configuration with PF

FreeBSD provides packet filtering (that is, *PF*) support at the kernel level. Using PF you can set up, maintain and inspect the packet filtering rule sets.

Warning: Different versions of FreeBSD use different versions of PF. Examples here are from FreeBSD 10.1, which uses the same version of PF as OpenBSD 4.5.

Enabling PF

In order to use PF on FreeBSD, you must first set the system up to load its kernel module. Additionally, you need to set the path to the configuration file for PF.

Using your preferred text editor, add the following lines to `/etc/rc.conf`:

```
pf_enable="YES"
pf_rules="/etc/pf.conf"
```

You may also want to enable logging support for PF and set the path for the log file. This can be done by adding the following lines to `/etc/rc.conf`:

```
pflog_enable="YES"
pflog_logfile="/var/log/pflog"
```

FreeBSD now loads the PF kernel module with logging features at boot.

Configuring PF Rules

In the above section, the configuration file for PF was set to `/etc/pf.conf`. This file allows you to set up the default firewall configuration that you want to use on your server. The settings you add to this file are the same for each cluster node.

There are two variables that you need to define for Galera Cluster in the PF configuration file: a list for the ports it needs open for TCP and a table for the IP addresses of nodes in the cluster.

```
# Galera Cluster Macros
wsrep_ports="{ 3306, 4567, 4568, 4444}"
table <wsrep_cluster_address> persist {192.168.1.1 192.168.1.2 192.168.1.3}"
```

Once you have these defined, you can add the rule to allow cluster packets to pass through the firewall.

```
# Galera Cluster TCP Filter Rule
pass in proto tcp from <wsrep_cluster_address> to any port $wsrep_ports keep state
```

If you deployed a cluster in a LAN environment, you need to also create on additional rule to open port `4567` to UDP transport for multicast replication.

```
# Galera Cluster UDP Filter Rule
pass in proto udp from <wsrep_cluster_address> to any port 4567 keep state
```

This defines the packet filtering rules that Galera Cluster requires. You can test the new rules for syntax errors using `pfctl`, with the `-n` options to prevent it from trying to load the changes.

```
# pfctl -v -nf /etc/pf.conf

wsrep_ports = "{ 3306, 4567, 4568, 4444 }"
table <wsrep_cluster_address> persist { 192.168.1.1 192.168.1.2 192.168.1.3 }
pass in proto tcp from <wsrep_cluster_address> to any port = mysql flags S/A/ keep␣
↪state
pass in proto tcp from <wsrep_cluster_address> to any port = 4567 flags S/SA keep␣
↪state
pass in proto tcp from <wsrep_cluster_address> to any port = 4568 flags S/SA keep␣
↪state
pass in proto tcp from <wsrep_cluster_address> to any port = krb524 falgs S/SA keep␣
↪state
pass in proto udp from <wsrep_cluster_address> to any port = 4567 keep state
```

If there are no syntax errors, `pfctl` prints each of the rules it adds to the firewall, (expanded, as in the example above). If there are syntax errors, it notes the line near where the errors occur.

Warning: The IP addresses in the example are for demonstration purposes only. Use the real values from your nodes and netmask in your PF configuration.

Starting PF

When you finish configuring packet filtering for Galera Cluster and for any other service you may require on your FreeBSD server, you can start the service. This is done with two commands: one to start the service itself and one to start the logging service.

```
# service pf start
# service pflog start
```

In the event that you have PF running already and want to update the rule set to use the settings in the configuration file for PF, (for example, the rules you added for Galera Cluster), you can load the new rules through the `pfctl` command.

```
# pfctl -f /etc/pf.conf
```

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training

- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

7.2 SSL Settings

Galera Cluster supports secure encrypted connections between nodes using SSL (Secure Socket Layer) protocol. This includes connections between database clients and servers through the standard SSL support in MySQL. It also includes encrypting replication traffic particular to Galera Cluster itself.

The SSL implementation is cluster-wide and does not support authentication for replication traffic. You must enable SSL for all nodes in the cluster or none of them.

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *SSL Configuration* (page 218)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

7.2.1 SSL Certificates

Before you can enable encryption for your cluster, you first need to generate the relevant certificates for the nodes to use. This procedure assumes that you are using OpenSSL.

Note: This chapter only covers certificate generation. For information on its use in Galera Cluster, see *SSL Configuration* (page 218).

Generating Certificates

There are three certificates that you need to create in order to secure Galera Cluster: the Certificate Authority (CA) key and cert; the server certificate, to secure `mysqld` activity and replication traffic; and the client certificate to secure the database client and `stunnel` for state snapshot transfers.

Note: When certificates expire there is no way to update the cluster without a complete shutdown. You can minimize the frequency of this downtime by using large values for the `-days` parameter when generating your certificates.

CA Certificate

The node uses the Certificate Authority to verify the signature on the certificates. As such, you need this key and cert file to generate the server and client certificates.

To create the CA key and cert, complete the following steps:

1. Generate the CA key.

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

2. Using the CA key, generate the CA certificate.

```
openssl req -new -x509 -nodes -days 365000 \
      -key ca-key.pem -out ca-cert.pem
```

This creates a key and certificate file for the Certificate Authority. They are in the current working directory as `ca-key.pem` and `ca-cert.pem`. You need both to generate the server and client certificates. Additionally, each node requires `ca-cert.pem` to verify certificate signatures.

Server Certificate

The node uses the server certificate to secure both the database server activity and replication traffic from Galera Cluster.

1. Create the server key.

```
openssl req -newkey rsa:2048 -days 365000 \
      -nodes -keyout server-key.pem -out server-req.pem
```

2. Process the server RSA key.

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

3. Sign the server certificate.

```
openssl x509 -req -in server-req.pem -days 365000 \
      -CA ca-cert.pem -CAkey ca-key.pem -set_serial 01 \
      -out server-cert.pem
```

This creates a key and certificate file for the server. They are in the current working directory as `server-key.pem` and `server-cert.pem`. Each node requires both to secure database server activity and replication traffic.

Client Certificate

The node uses the client certificate to secure client-side activity. In the event that you prefer physical transfer methods for state snapshot transfers, `rsync` for instance, the node also uses this key and certificate to secure `stunnel`.

1. Create the client key.

   ```
   openssl req -newkey rsa:2048 -days 365000 \
         -nodes -keyout client-key.pem -out client-req.pem
   ```

2. Process client RSA key.

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

3. Sign the client certificate.

   ```
   openssl x509 -req -in client-req.pem -days 365000 \
         -CA ca-cert.pem -CAkey ca-key.pem -set_serial 01 \
         -out client-cert.pem
   ```

This creates a key and certificate file for the database client. They are in the current working directory as `client-key.pem` and `client-cert.pem`.

Note: Each node requires both to secure client activity and state snapshot transfers.

Verifying the Certificates

When you finish creating the key and certificate files, use `openssl` to verify that they were generated correctly:

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

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

In the event that this verification fails, repeat the above process to generate replacement certificates.

The Common Name value used for the server and client certificates/keys must each differ from the Common Name value used for the CA certificate. Otherwise, the certificate and key files will not work for servers compiled using OpenSSL.

Once the certificates pass verification, you can send them out to each node. Use a secure method, such as `scp` or `sftp`. The node requires the following files:

- Certificate Authority: `ca-cert.pem`.
- Server Certificate: `server-key.pem` and `server-cert.pem`.

- Client Certificate: `client-key.pem` and `client-cert.pem`.

Place these files in the `/etc/mysql/certs` directory of each node, or a similar location where you can find them later in configuring the cluster to use SSL.

Related Documents

- *SSL Configuration* (page 218)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *SSL Certificates* (page 216)
- *SSL for State Snapshot Transfers* (page 221)
- *Galera Parameters* (page 271)
- *socket.ssl_key* (page 305)
- *socket.ssl_cert* (page 304)
- *socket.ssl_ca* (page 303)
- *wsrep_provider_options* (page 254)
- *socket.checksum* (page 304)
- *socket.ssl_cipher* (page 305)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

7.2.2 SSL Configuration

When you finish generating the SSL certificates for your cluster, you need to enable it for each node. If you have not yet generated the SSL certificates, see *SSL Certificates* (page 216) for a guide on how to do so.

Note: For Galera Cluster, SSL configurations are not dynamic. Since they must be set on every node in the cluster, if you are enabling this feature with a running cluster you need to restart the entire cluster.

Enabling SSL

There are three vectors that you can secure through SSL: traffic between the database server and client, replication traffic within Galera Cluster, and the *State Snapshot Transfer*.

Note: The configurations shown here cover the first two. The procedure for securing state snapshot transfers through SSL varies depending on the SST method you use. For more information, see *SSL for State Snapshot Transfers* (page 221).

Securing the Database

For securing database server and client connections, you can use the internal MySQL SSL support. In the event that you use logical transfer methods for state snapshot transfer, such as `mysqldump`, this is the only step you need to take in securing your state snapshot transfers.

In the configuration file, (`my.cnf`), add the follow parameters to each unit:

```
# MySQL Server
[mysqld]
ssl-ca = /path/to/ca-cert.pem
ssl-key = /path/to/server-key.pem
ssl-cert = /path/to/server-cert.pem

# MySQL Client Configuration
[mysql]
ssl-ca = /path/to/ca-cert.pem
ssl-key = /path/to/client-key.pem
ssl-cert = /path/to/client-cert.pem
```

These parameters tell the database server and client which files to use in encrypting and decrypting their interactions through SSL. The node will begin to use them once it restarts.

Securing Replication Traffic

In order to enable SSL on the internal node processes, you need to define the paths to the key, certificate and certificate authority files that you want the node to use in encrypting replication traffic.

- *socket.ssl_key* (page 305) The key file.
- *socket.ssl_cert* (page 304) The certificate file.
- *socket.ssl_ca* (page 303) The certificate authority file.

You can configure these options through the *wsrep_provider_options* (page 254) parameter in the configuration file, (that is, `my.cnf`).

```
wsrep_provider_options="socket.ssl_key=/path/to/server-key.pem;socket.ssl_cert=/path/
↪to/server-cert.pem;socket.ssl_ca=/path/to/cacert.pem"
```

This tells Galera Cluster which files to use in encrypting and decrypting replication traffic through SSL. The node will begin to use them once it restarts.

Configuring SSL

In the event that you want or need to further configure how the node uses SSL, Galera Cluster provides some additional parameters, including defining the cyclic redundancy check and setting the cryptographic cipher algorithm you want to use.

Note: For a complete list of available configurations available for SSL, see the options with the `socket.` prefix at *Galera Parameters* (page 271).

Configuring the Socket Checksum

Using the *socket.checksum* (page 304) parameter, you can define whether or which cyclic redundancy check the node uses in detecting errors. There are three available settings for this parameter, which are defined by an integer:

- `0` Disables the checksum.
- `1` Enables the CRC-32 checksum.
- `2` Enables the CRC-32C checksum.

The default configuration for this parameter is `1` or `2` depending upon your version. CRC-32C is optimized for and potentially hardware accelerated on Intel CPU's.

```
wsrep_provider_options = "socket.checksum=2"
```

Configuring the Encryption Cipher

Using the *socket.ssl_cipher* (page 305) parameter, one can override the default SSL cipher the node uses to encrypt replication traffic. Galera Cluster uses whatever ciphers are available to the SSL implementation installed on the nodes. For instance, if you install OpenSSL on your node, Galera Cluster can use any cipher supported by OpenSSL, as well as use filters to ensure that "weak" algorithms are not accepted on connection handshake.

```
wsrep_provider_options = "socket.ssl_cipher=ALL:!EXP:!NULL:!ADH:!LOW:!SSLv2:!SSLv3:!
↪MD5:!RC4:!RSA"
```

Related Documents

- *SSL Certificates* (page 216)
- *SSL for State Snapshot Transfers* (page 221)
- *Galera Parameters* (page 271)
- *socket.ssl_key* (page 305)
- *socket.ssl_cert* (page 304)
- *socket.ssl_ca* (page 303)
- *wsrep_provider_options* (page 254)
- *socket.checksum* (page 304)
- *socket.ssl_cipher* (page 305)

The Library

- *Documentation* (page 1)

- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *SSL Configuration* (page 218)
- *Schema Upgrades* (page 83)
- *wsrep_OSU_method* (page 253)
- *wsrep_sst_auth* (page 260)
- *wsrep_sst_method* (page 262)

Related Articles

- Starting a Cluster
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

7.2.3 SSL for State Snapshot Transfers

When you finish generating the SSL certificates for your cluster, you can begin configuring the node for their use. Where *SSL Configuration* (page 218) covers how to enable SSL for replication traffic and the database client, this page covers enabling it for *State Snapshot Transfer* scripts.

The particular method you use to secure the State Snapshot Transfer through SSL depends upon the method you use in state snapshot transfers: `mysqldump`, `clone`, `rsync` or `xtrabackup`.

Note: For Gelera Cluster, SSL configurations are not dynamic. Since they must be set on every node in the cluster, if you want to enable this feature with an existing cluster you need to restart the entire cluster.

Enabling SSL for `mysqldump`

The procedure for securing `mysqldump` is fairly similar to that of securing the database server and client through SSL. Given that `mysqldump` connects through the database client, you can use the same SSL certificates you created for replication traffic.

Before you shut down the cluster, you need to create a user for `mysqldump` on the database server and grant it privileges through the cluster. This ensures that when the cluster comes back up, the nodes have the correct privileges

to execute the incoming state snapshot transfers. In the event that you use the *Total Order Isolation* online schema upgrade method, you only need to execute the following commands on a single node.

1. From the database client, check that you use Total Order Isolation for online schema upgrades.

```
SHOW VARIABLES LIKE 'wsrep_OSU_method';

+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| wsrep_OSU_method | TOI   |
+------------------+-------+
```

   If *wsrep_OSU_method* (page 253) is set to *Rolling Schema Upgrade*, or `ROI`, then you need to execute the following commands on each node individually.

2. Create a user for `mysqldump`.

```
CREATE USER 'sst_user'@'%' IDENTIFIED BY PASSWORD 'sst_password';
```

   Bear in mind that, due to the manner in which the SST script is called, the user name and password must be the same on all nodes.

3. Grant privileges to this user and require SSL.

```
GRANT ALL ON *.* TO 'sst_user'@'%' REQUIRE SSL;
```

4. From the database client on a different node, check to ensure that the user has replicated to the cluster.

```
SELECT User, Host, ssl_type
FROM mysql.user WHERE User='sst_user';

+----------+------+----------+
| User     | Host | ssl_type |
+----------+------+----------+
| sst_user | %    | Any      |
+----------+------+----------+
```

This configures and enables the `mysqldump` user for the cluster.

Note: In the event that you find, *wsrep_OSU_method* (page 253) set to `ROI`, you need to manually create the user on each node in the cluster. For more information on rolling schema upgrades, see *Schema Upgrades* (page 83).

With the user now on every node, you can shut the cluster down to enable SSL for `mysqldump` State Snapshot Transfers.

1. Using your preferred text editor, update the `my.cnf` configuration file to define the parameters the node requires to secure state snapshot transfers.

```
# MySQL Server
[mysqld]
ssl-ca = /path/to/ca-cert.pem
ssl-key = /path/to/server-key.pem
ssl-cert = /path/to/server-cert.pem

# MySQL Client Configuration
[client]
```

(continues on next page)

(continued from previous page)

```
ssl-ca = /path/to/ca-cert.pem
ssl-key = /path/to/client-key.pem
ssl-cert = /path/to/client-cert.pem
```

2. Additionally, configure *wsrep_sst_auth* (page 260) with the SST user authentication information.

```
[mysqld]
# mysqldump SST auth
wsrep_sst_auth = sst_user:sst_password
```

This configures the node to use `mysqldump` for state snapshot transfers over SSL. When all nodes are updated to SSL, you can begin restarting the cluster. For more information on how to do this, see Starting a Cluster.

Enabling SSL for `clone` based SST

Configurations for `clone` are handled through the `my.cnf` configuration file, in the same manner as for `mysqldump`-based SST above. You can use the same SSL certificate files as the node uses on the database server, client and with replication traffic.

```
# clone Configuration
[mysqld]
ssl-cert= /path/to/server-cert.pem
ssl-key= /path/to/server-key.pem
ssl-ca= /path/to/ca.pem

[client] # or [sst]
ssl-cert= /path/to/client-cert.pem
ssl-key= /path/to/client-key.pem
ssl-ca= /path/to/ca.pem
ssl-mode= VERIFY_CA
```

Client SSL configuration on *Donor Node* must match server SSL configuration on Joiner. (That means: mysql client using client SSL configuration from Joiner should be able to connect to server on Donor) Client SSL configuration on Joiner must match CLONE SSL configuration on donor. If CLONE plugin on Donor is not loaded, or if CLONE SSL configuration is empty then server SSL configuration on Donor is used.

If for some reason general client SSL configuration is undesirable, client SSL configuration for `clone` SST can be put into the *[sst]* section of the configuration file. It will be used first.

Enabling SSL for `xtrabackup` and `rsync` based SSTs

The *Physical State Transfer Method* for state snapshot transfers, uses an external script to copy the physical data directly from the file system on one cluster node into another. Before releases 5.7.34 and 8.0.25 only `xtrabackup-v2` SST supported SSL encryption and required custom configuration. Starting with releases 5.7.34 and 8.0.25 both `rsync` and `xtrabackup-v2` scripts can use the standard MySQL SSL configuration and will use it **BY DEFAULT**.

New way SSL configuration for `xtrabackup-v2` and `rsync` SSTs (releases 5.7.34 and 8.0.25 or newer)

If `[mysqld]` or `[server]` section of the configuration contains

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

those credentials will be automatically used for SSL encryption of SST unless explicitly overridden with the same parameters in `[sst]` section.

For backward compatibility no peer/CA authentication is performed unless explicitly requested in the `[sst]` section of the configuration using the standard `ssl-mode` option:

```
[sst]
ssl-mode=VERIFY_CA
```

or

```
[sst]
ssl-mode=VERIFY_IDENTITY
```

This is a backward incompatible option and should be used only on fully upgraded clusters.

```
[sst]
ssl-mode=DISABLED
```

disables SSL encryption for SST regardless server SSL settings

Old way SSL configuration for `xtrabackup-v2` SST

This is deprecated, but for backward compatibility takes precedence if present.

Configurations for `xtrabackup-v2` script are handled through the `my.cnf` configuration file, in the same as the database server and client. Use the `[sst]` unit to configure SSL for the script. You can use the same SSL certificate files as the node uses on the database server, client and with replication traffic.

```
# xtrabackup Configuration
[sst]
encrypt = 3
tca = /path/to/ca.pem
tkey = /path/to/key.pem
tcert = /path/to/cert.pem
```

When you finish editing the configuration file, restart the node to apply the changes. `xtrabackup` now sends and receives state snapshot transfers through SSL.

Note: In order to use SSL with `xtrabackup`, you need to set *wsrep_sst_method* (page 262) to `xtrabackup-v2`, instead of `xtrabackup`.

Related Documents

- *SSL Configuration* (page 218)
- *Schema Upgrades* (page 83)
- *wsrep_OSU_method* (page 253)
- *wsrep_sst_auth* (page 260)

- *wsrep_sst_method* (page 262)

Related Articles

- Starting a Cluster

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *wsrep_notify_cmd* (page 250)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

7.3 SELinux Configuration

Security-Enhanced Linux, or SELinux, is a kernel module for improving security of Linux operating systems. It integrates support for access control security policies, including mandatory access control (MAC), that limit user applications and system daemons access to files and network resources. Some Linux distributions, such as Red Hat Enterprise Linux or CentOS, ship with SELinux enabled by default.

In the context of Galera Cluster, systems with SELinux may block the database server, keeping it from starting or preventing the node from establishing connections with other nodes in the cluster. To prevent this, you need to configure SELinux policies to allow the node to operate.

Generating an SELinux Policy

In order to create an SELinux policy for Galera Cluster, you need to first open ports and set SELinux to permissive mode. Then, after generating various replication events, state transfers and notifications, create a policy from the logs of this activity and reset SELinux from to enforcing mode.

Setting SELinux to Permissive Mode

When SELinux registers a system event, there are three modes that define its response: enforcing, permissive and disabled. While you can set it to permit all activity on the system, this is not a good security practice. Instead, set SELinux to permit activity on the relevant ports and to ignore the database server.

To set SELinux to permissive mode, complete the following steps:

1. Using `semanage`, open the relevant ports:

   ```
   semanage port -a -t mysqld_port_t -p tcp 4567
   semanage port -a -t mysqld_port_t -p tcp 4568
   semanage port -a -t mysqld_port_t -p tcp 4444
   ```

   SELinux already opens the standard MySQL port `3306`. In the event that you use UDP in your cluster, you also need to open `4567` to those connections.

   ```
   semanage port -a -t mysqld_port_t -p udp 4567
   ```

2. Set SELinux to permissive mode for the database server.

   ```
   semanage permissive -a mysqld_t
   ```

SELinux now permits the database server to function on the server and no longer blocks the node from network connectivity with the cluster.

Defining the SELinux Policy

While SELinux remains in permissive mode, it continues to log activity from the database server. In order for it to understand normal operation for the database, you need to start the database and generate routine events for SELinux to see.

For servers that use `init`, start the database with the following command:

```
service mysql start
```

For servers that use `systemd`, instead run this command:

```
systemctl mysql start
```

You can now begin to create events for SELinux to log. There are many ways to go about this, including:

- Stop the node, then make changes on another node before starting it again. Not being that far behind, the node updates itself using an *Incremental State Transfer*.
- Stop the node, delete the `grastate.dat` file in the data directory, then restart the node. This forces a *State Snapshot Transfer*.
- Restart the node, to trigger the notification command as defined by *wsrep_notify_cmd* (page 250).

When you feel you have generated sufficient events for the log, you can begin work creating the policy and turning SELinux back on.

Note: In order to for your policy to work you must generate both State Snapshot and Incremental State transfers.

Enabling an SELinux Policy

Generating an SELinux policy requires that you search log events for the relevant information and pipe it to the `audit2allow` utility, creating a `galera.te` file to load into SELinux.

To generate and load an SELinux policy for Galera Cluster, complete the following steps:

1. Using `fgrep` and `audit2allow`, create a textease file with the policy information.

```
fgrep "mysqld" /var/log/audit/audit.log | audit2allow -m galera -o galera.te
```

This creates a `galera.te` file in your working directory.

2. Compile the audit logs into an SELinux policy module.

```
checkmodule -M -m galera.te -o galera.mod
```

This creates a `galera.mod` file in your working directory.

3. Package the compiled policy module.

```
semodule_package -m galera.mod -o galera.pp
```

This creates a `galera.pp` file in your working directory.

4. Load the package into SELinux.

```
semodule -i galera.pp
```

5. Disable permissive mode for the database server.

```
semanage permissive -d mysqld_t
```

SELinux returns to enforcement mode, now using new policies that work with Galera Cluster.

Related Documents

- *wsrep_notify_cmd* (page 250)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Galera Parameters* (page 271)
- *Galera Status Variables* (page 307)
- *Galera System Tables* (page 77)
- *GLB Parameters* (page 343)
- *MySQL wsrep Options* (page 231)
- *Galera Functions* (page 268)
- *Versioning Information* (page 352)
- *XtraBackup-v2 Parameters* (page 335)
- Home

- *Docs* (page 1)
- KB
- Training
- FAQ

CHAPTER

EIGHT

REFERENCE

In the event that you need more information about particular variables or parameters or status variable or would like a clearer explanation about various terms used in the documentation, these chapters provide general reference material to Galera Cluster configuration and use.

Variable Reference

Defining persistent configurations in Galera Cluster is done through the underlying database server, using the `[mysqld]` unit in the `my.cnf` configuration file. These chapters provide reference guides to the base replication status and configuration variables as well as the specific wsrep Provider options implemented through the Galera Replication Plugin.

- *MySQL wsrep Options* (page 231)

  In addition to the standard configuration variables available through the database server, Galera Cluster implements several that are unique and specific to fine-tuning database replication. This chapter provides a reference guide to these new configuration variables in Galera Cluster

- *Galera Functions* (page 268)

  There are a few Galera specific functions. This page lists and explains them, as well as gives examples of their use.

- *Galera Parameters* (page 271)

  The Galera Replication Plugin, which acts as the wsrep Provider, includes a number of parameters specific to its operation. This chapter provides a reference guide to the various wsrep Provider options available.

- *Galera Status Variables* (page 307)

  In addition to the standard status variables available through the database server, Galera Cluster also implements several that you can use in determining the status and health of the node and the cluster. This chapter provides a reference guide to these new status variables in Galera Cluster.

Utility Reference

In some cases your configuration or implementation may require that you work with external utilities in your deployment of Galera Cluster. These chapters provide reference guides for two such utilities: XtraBackup and Galera Load Balancer.

- *Galera Load Balancer Parameters* (page 343)

  In high availability situations or similar cases where nodes are subject to high traffic situations, you may find it beneficial to set up a load balancer between your application servers and the cluster. This chapter provides a reference guide to the Codership implementation: the Galera Load Balancer.

- *XtraBackup-v2 Parameters* (page 335)

  When you manage State Snapshot Transfers using Percona XtraBackup, it allows you to set various parameters on the state transfer script the node uses from the `my.cnf` configuration file. This chapter provides a reference guide to options available to XtraBackup.

- *Galera System Tables* (page 77)

  This page provides information on the Galera specific system tables. These were added as of version 4 of Galera.

Miscellaneous References

- *Versioning Information* (page 352)

  While the documentation follows a convention of 3.x in speaking of release versions for Galera Cluster, in practice the numbering system is somewhat more complicated: covering releases of the underlying database server, the wsrep Provider and the wsrep API. This chapter provides a more thorough explanation of versioning with Galera Cluster.

- *Legal Notice* (page 354)

  This page provides information on the documentation copyright.

- *Glossary* (page 355)

  In the event that you would like clarification on particular topics, this chapter provides reference information on core concepts in Galera Cluster.

- genindex

  In the event that you would like to check these concepts against related terms to find more information in the docs, this chapter provides a general index of the site.

Related Documents

- *Galera Parameters* (page 271)
- *Galera Status Variables* (page 307)
- *Galera System Tables* (page 77)
- *GLB Parameters* (page 343)
- *MySQL wsrep Options* (page 231)
- *Galera Functions* (page 268)
- *Versioning Information* (page 352)
- *XtraBackup-v2 Parameters* (page 335)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ

- search
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

8.1 MySQL wsrep Options

These are MySQL system variables starting from wsrep API patch version 21.1 for MySQL 5.1.58. Although there were earlier versions of MySQL-wsrep, this was the first one to use consistent versioning scheme as was chosen as the starting point.

Almost all of the variables are global except for a few. Those are session variables. If you click on a particular variable in this table, your web browser will scroll down to the entry for it with more details and an explanation.

Option	Default Value	Global	Dynamic
innodb-wsrep-applier-lock-wait-timeout (page 232)	0	Yes	Yes
plugin_wsrep_provider (page 233)	ON	Yes	
wsrep_applier_FK_failure_retries (page 234)	1	Yes	Yes
wsrep_auto_increment_control (page 234)	ON	Yes	Yes
wsrep_causal_reads (page 235)	OFF	Session	Yes
wsrep_certify_nonPK (page 236)	ON	Yes	Yes
wsrep_certification_rules (page 235)	OPTIMIZED	Yes	Yes
wsrep_cluster_address (page 236)	ON	Yes	No
wsrep_cluster_name (page 237)	example_cluster	Yes	Yes
wsrep_convert_LOCK_to_trx (page 238)	OFF	Yes	Yes
wsrep_data_home_dir (page 239)	/path/to/datadir	Yes	No
wsrep_dbug_option (page 239)		Yes	Yes
wsrep_debug (page 240)	OFF	Yes	Yes
wsrep_desync (page 241)	OFF	Yes	Yes
wsrep_dirty_reads (page 242)	OFF	Yes	Yes
wsrep_drupal_282555_workaround (page 242)	ON	Yes	Yes
wsrep_forced_binlog_format (page 243)	NONE	Yes	Yes
wsrep_ignore_apply_errors (page 244)	7	Yes	Yes
wsrep_info_level (page 244)	0	Yes	Yes
wsrep_load_data_splitting (page 245)	ON	Yes	Yes
wsrep_log_conflicts (page 245)	OFF	Yes	Yes
wsrep_max_ws_rows (page 246)	0	Yes	Yes
wsrep_max_ws_size (page 246)	2G	Yes	Yes
wsrep_mode (page 247)	ON	Yes	Yes

Continued on next page

Table 1 – continued from previous page

Option	Default Value	Global	Dynamic
wsrep_node_address (page 248)	*host address:default port*	Yes	No
wsrep_node_incoming_address (page 249)	*host address:mysqld port*	Yes	No
wsrep_node_name (page 250)	`<hostname>`	Yes	Yes
wsrep_notify_cmd (page 250)	(empty)	Yes	Yes
wsrep_on (page 252)	`ON`	Yes	Yes
wsrep_OSU_method (page 253)	`TOI`		Yes
wsrep_preordered (page 253)	`OFF`	Yes	Yes
wsrep_provider (page 254)	`NONE`	Yes	No
wsrep_provider_options (page 254)	(empty)	Yes	No
wsrep_recover (page 255)	`OFF`	Yes	No
wsrep_reject_queries (page 256)	`NONE`	Yes	Yes
wsrep_restart_replica (page 256)	`OFF`	Yes	Yes
wsrep_restart_slave (page 257)	`OFF`	Yes	Yes
wsrep_retry_autocommit (page 257)	`1`	Yes	No
wsrep_applier_FK_checks (page 257)	`ON`	Yes	Yes
wsrep_slave_FK_checks (page 258)	`ON`	Yes	Yes
wsrep_applier_threads (page 258)	`1`	Yes	Yes
wsrep_slave_threads (page 259)	`1`	Yes	Yes
wsrep_applier_UK_checks (page 259)	`OFF`	Yes	Yes
wsrep_slave_UK_checks (page 259)	`OFF`	Yes	Yes
wsrep_sst_auth (page 260)	(empty)	Yes	Yes
wsrep_sst_donor (page 260)		Yes	Yes
wsrep_sst_donor_rejects_queries (page 261)	`OFF`	Yes	Yes
wsrep_sst_method (page 262)	`mysqldump`	Yes	Yes
wsrep_sst_receive_address (page 263)	*node IP address*	Yes	Yes
wsrep_start_position (page 264)	*see reference entry*	Yes	Yes
wsrep_status_file (page 264)	None	Yes	No
wsrep_sync_server_uuid (page 265)	`0`	Yes	Yes
wsrep_sync_wait (page 265)	`0`	Yes	Yes
wsrep_trx_fragment_size (page 267)	`0`	Yes	Yes
wsrep_trx_fragment_unit (page 267)	`bytes`	Yes	Yes

You can execute the `SHOW VARIABLES` statement with the `LIKE` operator as shown below to get list of all Galera related variables on your server:

```
SHOW VARIABLES LIKE 'wsrep%';
```

The results will vary depending on which version of Galera is running on your server. All of the parameters and variables possible are listed above, but they're listed below with explanations of each.

innodb-wsrep-applier-lock-wait-timeout

The innodb-wsrep-applier-lock-wait-timeout parameter defines the timeout in seconds, after which the wsrepw watchdog starts killing local transactions that are blocking the applier. Value 0 disables the watchdog.

Command-line Format	--innodb-wsrep-applier-lock-wait-timeout
System Variable	innodb-wsrep-applier-lock-wait-timeout
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	0 or timeout in seconds
Default Value	0
Initial Version	MySQL-wsrep 8.0.26-26.8

You can execute the following SHOW VARIABLES statement to see how this variable is set:

```
SHOW VARIABLES LIKE 'innodb-wsrep-applier-lock-wait-timeout';

 +----------------------------------------+-------+
 | Variable_name                          | Value |
 +----------------------------------------+-------+
 | innodb-wsrep-applier-lock-wait-timeout | 10    |
 +----------------------------------------+-------+
```

plugin_wsrep_provider

Defines optional settings the node passes to the wsrep Provider.

Command-line Format	--plugin_wsrep_provider
System Variable	plugin_wsrep_provider
Variable Scope	Global
Dynamic Variable	
Permitted Values	String
Default Value	ON
Valied Values	ON, OFF
Initial Version	MySQL-wsrep: 8.4.2, MariaDB: 11.x

When this option is enabled, the node loads the wsrep Provider configuration options through the plugin_wsrep_provider plugin, instead of using wsrep_provider_options options. wsrep_provider_options options can no longer be used.

The configuration required to use plugin_wsrep_provider:

```
wsrep-on=ON
wsrep-cluster-address=gcomm://
wsrep-provider=@ENV.WSREP_PROVIDER
plugin-wsrep-provider=ON
```

Often it is not optimal to use a system variable, that is, one large string, for a long list of options. Instead, it would be better to separate the different options into separate cluster system variables.

When this option is disabled, the node loads the wsrep Provider configuration options from wsrep_provider_options options.

For more information on the wsrep Provider options, see *Galera Parameters* (page 271) and *wsrep_provider_options* (page 254).

```
SHOW VARIABLES LIKE 'plugin_wsrep_provider';

+------------------------+-------------------------------------------------+
| Variable_name          | Value                                           |
+------------------------+-------------------------------------------------+
| plugin_wsrep_provider  | ON                                              |
+------------------------+-------------------------------------------------+
```

wsrep_applier_FK_failure_retries

Occasionally, foreign key constrains may fail even though the constraints themselves are not violated (for example, if the same transaction inserts in the parent table, and the next insert into the child table fails in FK checks). With this foreign key constraint check retrying implementation, you can control the number of retries. If the constraint check fails despite retires, the final retry prints out a warning with an error code and InnoDB system monitor output for further troubleshooting.

Command-line Format	`--wsrep_applier_FK_failure_retries`
System Variable	`wsrep_applier_FK_failure_retries`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Integer
Default Value	`1`
Initial Version	MySQL-wsrep 8.0.35

You can execute the following `SHOW VARIABLES` statement to see how this variable is set:

```
SHOW VARIABLES LIKE 'wsrep_applier_FK_failure_retries';

 +---------------------------------------+-------+
 | Variable_name                         | Value |
 +---------------------------------------+-------+
 | wsrep_applier_FK_failure_retries      | 1     |
 +---------------------------------------+-------+
```

wsrep_auto_increment_control

This parameter enables the automatic adjustment of auto increment system variables with changes in cluster membership.

Command-line Format	`--wsrep-auto-increment-control`
System Variable	`wsrep_auto_increment_control`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Boolean
Default Value	`ON`
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

The node manages auto-increment values in a table using two variables: `auto_increment_increment` and `auto_increment_offset`. The first relates to the value auto-increment rows count from the offset. The second

relates to the offset it should use in moving to the next position.

The *wsrep_auto_increment_control* (page 234) parameter enables additional calculations to this process, using the number of nodes connected to the *Primary Component* to adjust the increment and offset. This is done to reduce the likelihood that two nodes will attempt to write the same auto-increment value to a table.

It significantly reduces the rate of certification conflicts for `INSERT` statements. You can execute the following `SHOW VARIABLES` statement to see how this variable is set:

```
SHOW VARIABLES LIKE 'wsrep_auto_increment_control';

 +------------------------------+-------+
 | Variable_name                | Value |
 +------------------------------+-------+
 | wsrep_auto_increment_control | ON    |
 +------------------------------+-------+
```

wsrep_causal_reads

This parameter enables the enforcement of strict cluster-wide `READ COMMITTED` semantics on non-transactional reads. It results in larger read latencies.

Command-line Format	`--wsrep-causal-reads`
System Variable	`wsrep_causal_reads`
Variable Scope	Session
Dynamic Variable	Yes
Permitted Values	Boolean
Default Value	`OFF`
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21
Deprecated Version	MySQL-wsrep: 5.5.42-25.12, MariaDB 10.1.3, removed from MariaDB 11.3.0

You can execute the following `SHOW VARIABLES` statement with a `LIKE` operator to see how this variable is set:

```
SHOW VARIABLES LIKE 'wsrep_causal_reads';
```

Warning: The `wsrep_causal_reads` option has been **deprecated**. It has been replaced by *wsrep_sync_wait* (page 265).

wsrep_certification_rules

Certification rules to use in the cluster.

Command-line Format	`--wsrep-certification-rules`
System Variable	`wsrep_certification_rules`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Enumeration
Default Value	`OPTIMIZED`
Valid Value	`OPTIMIZED`, `STRICT`
Initial Version	MySQL-wsrep: 5.5.61-25.24, 5.6.41-25.23, 5.7.23-25.15
Deprecated Version	MySQL-wsrep: 8.0.19-26.3

Controls how certification is done in the cluster. To be more specific, this parameter affects how foreign keys are handled: with the `STRICT` option, two INSERTs that happen at about the same time on two different nodes in a child table, and insert different (non conflicting) rows, but both rows point to the same row in the parent table, could result in certification failure. With the `OPTIMIZED` option, such certification failure is avoided.

Note: The Galera versions where this parameter was supported are no longer in production use, and have reached end-of-life status.

```
SHOW VARIABLES LIKE 'wsrep_certification_rules';

+---------------------------+--------+
| Variable_name             | Value  |
+---------------------------+--------+
| wsrep_certification_rules | STRICT |
+---------------------------+--------+
```

wsrep_certify_nonPK

This parameter is used to define whether the node should generate primary keys on rows without them for the purposes of certification.

Command-line Format	`--wsrep-certify-nonpk`
System Variable	`wsrep_certify_nonpk`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Boolean
Default Value	`ON`
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

Galera Cluster requires primary keys on all tables. The node uses the primary key in replication to allow for the parallel applying of transactions to a table. This parameter tells the node that when it encounters a row without a primary key, it should create one for replication purposes. However, as a rule do not use tables without primary keys.

You can execute the following `SHOW VARIABLES` statement with a `LIKE` operator to see how this variable is set:

```
SHOW VARIABLES LIKE 'wsrep_certify_nonpk';

+---------------------+-------+
| Variable_name       | Value |
+---------------------+-------+
| wsrep_certify_nonpk | ON    |
+---------------------+-------+
```

wsrep_cluster_address

This parameter sets the back-end schema, IP addresses, ports and options the node uses in connecting to the cluster.

Command-line Format	`--wsrep-cluster-address`
System Variable	`wsrep_cluster_address`
Variable Scope	Global
Dynamic Variable	No
Permitted Values	String
Default Value	
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

Galera Cluster uses this parameter to determine the IP addresses for the other nodes in the cluster, the back-end schema to use and additional options it should use in connecting to and communicating with those nodes. Currently, the only back-end schema supported for production is `gcomm`.

Below is the syntax for this the values of this parameter:

```
<backend schema>://<cluster address>[?option1=value1[&option2=value2]]
```

Here's an example of how that might look:

```
wsrep_cluster_address="gcomm://192.168.0.1:4567?gmcast.listen_addr=0.0.0.0:5678"
```

Changing this variable while Galera is running will cause the node to close the connection to the current cluster, and reconnect to the new address. Doing this at runtime may not be possible, though, for all SST methods. As of Galera Cluster 23.2.2, it is possible to provide a comma-separated list of other nodes in the cluster as follows:

```
gcomm://node1:port1,node2:port2,...[?option1=value1&...]
```

Using the string `gcomm://` without any address will cause the node to startup alone, thus initializing a new cluster–that the other nodes can join to. Using `--wsrep-new-cluster` is the newer, preferred way.

Warning: Never use an empty `gcomm://` string with the `wsrep_cluster_address` option in the configuration file. If a node restarts, it will cause the node not to rejoin the cluster. Instead, it will initialize a new one-node cluster and cause a *Split Brain*. To bootstrap a cluster, you should only pass the `--wsrep-new-cluster` string at the command-line–instead of using `--wsrep-cluster-address="gcomm://"`. For more information, see Starting the Cluster.

You can execute the following SQL statement to see how this variable is set:

```
SHOW VARIABLES LIKE 'wsrep_cluster_address';

+-----------------------+-----------------------------------------------+
| Variable_name         | Value                                         |
+-----------------------+-----------------------------------------------+
| wsrep_cluster_address | gcomm://192.168.1.1,192.168.1.2,192.168.1.3 |
+-----------------------+-----------------------------------------------+
```

wsrep_cluster_name

This parameter defines the logical cluster name for the node.

Command-line Format	`--wsrep-cluster-name`
System Variable	`wsrep_cluster_name`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	String
Default Value	`exmaple_cluster`
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

This parameter allows you to define the logical name the node uses for the cluster. When a node attempts to connect to a cluster, it checks the value of this parameter against that of the cluster. The connection is only made if the names match. If they do not match, the connection fails. Because of this, the cluster name must be the same on all nodes.

You can execute the following `SHOW VARIABLES` statement with a `LIKE` operator to see how this variable is set:

```
SHOW VARIABLES LIKE 'wsrep_cluster_name';

+--------------------+-----------------+
| Variable_name      | Value           |
+--------------------+-----------------+
| wsrep_cluster_name | example_cluster |
+--------------------+-----------------+
```

wsrep_convert_lock_to_trx

This parameter is used to set whether the node converts `LOCK/UNLOCK TABLES` statements into `BEGIN/COMMIT` statements.

Command-line Format	`--wsrep-convert-lock-to-trx`
System Variable	`wsrep_convert_lock_to_trx`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Boolean
Default Value	`OFF`
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21
Deprecated Version	MySQL-wsrep: 8.0.19-26.3

This parameter determines how the node handles `LOCK/UNLOCK TABLES` statements, specifically whether or not you want it to convert these statements into `BEGIN/COMMIT` statements. It tells the node to convert implicitly locking sessions into transactions within the database server. By itself, this is not the same as support for locking sections, but it does prevent the database from resulting in a logically inconsistent state.

This parameter may sometimes help to get old applications working in a multi-primary setup.

Note: Loading a large database dump with `LOCK` statements can result in abnormally large transactions and cause an out-of-memory condition.

You can execute the following `SHOW VARIABLES` statement with a `LIKE` operator to see how this variable is set:

```
SHOW VARIABLES LIKE 'wsrep_convert_lock_to_trx';

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

(continues on next page)

(continued from previous page)

```
| Variable_name              | Value |
+----------------------------+-------+
| wsrep_convert_lock_to_trx | OFF   |
+----------------------------+-------+
```

wsrep_data_home_dir

Use this parameter to set the directory the wsrep Provider uses for its files.

Command-line Format	`--wsrep_data_home_dir`
System Variable	`wsrep_data_home_dir`
Variable Scope	Global
Dynamic Variable	No
Permitted Values	Directory
Default Value	/path/mysql_datadir
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

During operation, the wsrep Provider needs to save various files to disk that record its internal state. This parameter defines the path to the directory that you want it to use. If not set, it defaults the MySQL `datadir` path.

You can execute the following `SHOW VARIABLES` statement with a `LIKE` operator to see how this variable is set:

```
SHOW VARIABLES LIKE 'wsrep_data_home_dir';

+---------------------+----------------+
| Variable_name       | Value          |
+---------------------+----------------+
| wsrep_data_home_dir | /var/lib/mysql |
+---------------------+----------------+
```

wsrep_dbug_option

You can set debug options to pass to the wsrep Provider with this parameter.

Command-line Format	`--wsrep-dbug-option`
System Variable	`wsrep_dbug_option`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	String
Default Value	
Initial Version	MySQL-wsrep: 5.5.15-21.1, MariaDB: 5.5.21

You can execute the following `SHOW VARIABLES` statement with a `LIKE` operator to see how this variable is set, if it is set:

```
SHOW VARIABLES LIKE 'wsrep_dbug_option';

+------------------+-------+
| Variable_name     | Value |
+------------------+-------+
```

(continues on next page)

(continued from previous page)

```
| wsrep_dbug_option |       |
+-------------------+-------+
```

wsrep_debug

This parameter enables additional debugging output for the database server error log.

Command-line Format	`--wsrep-debug`
System Variable	`wsrep_debug`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Boolean
Default Value	`OFF`
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

Under normal operation, error events are logged to an error log file for the database server. By default, the name of this file is the server hostname with the `.err` extension. You can define a custom path using the log_error parameter. When you enable *wsrep_debug* (page 240), the database server logs additional events surrounding these errors to help in identifying and correcting problems.

DDL statements are also logged. See below for an example:

```
2024-09-06 14:37:57 13 [Note] WSREP: TOI Begin: CREATE SEQUENCE seq start with 1␣
↪minvalue 1 maxvalue 1000000 increment by 0 cache 1000 nocycle ENGINE=InnoDB
2024-09-06 14:37:57 13 [Note] WSREP: enter_toi_local: enter(13,exec,local,success,0,
↪toi: -1,nbo: -1)
2024-09-06 14:37:57 13 [Note] WSREP: poll_enter_toi: 3,0
2024-09-06 14:37:57 13 [Note] WSREP: enter_toi_local: leave(13,exec,toi,success,0,
↪toi: 3,nbo: -1)
2024-09-06 14:37:57 13 [Note] WSREP: avoiding binlog rotate due to TO isolation: 1
2024-09-06 14:37:57 13 [Note] WSREP: TO END: 3: CREATE SEQUENCE seq start with 1␣
↪minvalue 1 maxvalue 1000000 increment by 0 cache 1000 nocycle ENGINE=InnoDB
```

Warning: In addition to useful debugging information, the `wsrep_debug` parameter also causes the database server to print authentication information (that is, passwords) to the error logs. Do not enable it in production environments. This, however, does not concern MariaDB, as the "wsrep_thd_query()", where the user query is exposed, does not print all information when the "SQL_COMMAND" is "SET" (such as "SET PASSWORD") or "SQLCOM_CREATE_USER", where "CREATE USER" is only logged.

See below for an example of `wsrep_debug` output:

```
2024-09-06 14:26:19 2 [Note] WSREP: open: enter(4,none,high priority,success,0,toi: -
↪1,nbo: -1)
2024-09-06 14:26:19 2 [Note] WSREP: open: leave(4,idle,high priority,success,0,toi: -
↪1,nbo: -1)
2024-09-06 14:26:19 2 [Note] WSREP: before_command: enter(4,idle,high priority,
↪success,0,toi: -1,nbo: -1)
2024-09-06 14:26:19 4 [Note] WSREP: before_command: success(4,exec,high priority,
↪success,0,toi: -1,nbo: -1)
2024-09-06 14:26:19 4 [Note] WSREP: Cluster table is empty, not recovering␣
↪transactions
```

(continues on next page)

(continued from previous page)

```
2024-09-06 14:26:19 2 [Note] WSREP: after_command_before_result: enter(4,exec,high␣
↪priority,success,0,toi: -1,nbo: -1)
2024-09-06 14:26:19 2 [Note] WSREP: after_command_before_result: leave(4,result,high␣
↪priority,success,0,toi: -1,nbo: -1)
2024-09-06 14:26:19 2 [Note] WSREP: after_command_after_result_enter(4,result,high␣
↪priority,success,0,toi: -1,nbo: -1)
2024-09-06 14:26:19 2 [Note] WSREP: after_command_after_result: leave(4,idle,high␣
↪priority,success,0,toi: -1,nbo: -1)
2024-09-06 14:26:19 2 [Note] WSREP: close: enter(4,idle,high priority,success,0,toi: -
↪1,nbo: -1)
2024-09-06 14:26:19 2 [Note] WSREP: close: leave(4,quit,high priority,success,0,toi: -
↪1,nbo: -1)
2024-09-06 14:26:19 4 [Note] WSREP: cleanup: enter(4,quit,local,success,0,toi: -1,
↪nbo: -1)
2024-09-06 14:26:19 4 [Note] WSREP: cleanup: leave(4,none,local,success,0,toi: -1,
↪nbo: -1)
```

The `wsrep_debug` options are:

- `SERVER` - `WSREP_DEBUG` log writes from the source code will be added to the error log.
- `TRANSACTION` - Logging from `wsrep-lib` transactions will be added to the error log.
- `STREAMING` - Logging from streaming transactions in `wsrep-lib` will be added to the error log.
- `CLIENT` - Logging from `wsrep-lib` client state will be added to the error log.

See also *evs.debug_log_mask* (page 280).

You can execute the following `SHOW VARIABLES` statement with a `LIKE` operator to see if this variable is enabled:

```
SHOW VARIABLES LIKE 'wsrep_debug';

+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| wsrep_debug   | OFF   |
+---------------+-------+
```

wsrep_desync

This parameter is used to set whether or not the node participates in Flow Control.

Command-line Format	`--wsrep_desync`
System Variable	`wsrep_desync`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Boolean
Default Value	`OFF`
Initial Version	MySQL-wsrep: 5.5.33-23.7.6, MariaDB: 5.5.33

When a node receives more write-sets than it can apply, the transactions are placed in a received queue. In the event that the node falls too far behind, it engages Flow Control. The node takes itself out of sync with the cluster and works through the received queue until it reaches a more manageable size.

For more information on Flow Control and how to configure and manage it in a cluster, see *Flow Control* (page 23) and *Managing Flow Control* (page 96).

When set to `ON`, this parameter disables Flow Control for the node. The node will continue to receive write-sets and fall further behind the cluster. The cluster does not wait for desynced nodes to catch up, even if it reaches the `fc_limit` value.

You can execute the following `SHOW VARIABLES` statement with a `LIKE` operator to see if this variable is enabled:

```
SHOW VARIABLES LIKE 'wsrep_desync';

+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| wsrep_desync  | OFF   |
+---------------+-------+
```

wsrep_dirty_reads

This parameter defines whether the node accepts read queries when in a non-operational state.

Command-line Format	`--wsrep-dirty-reads`
System Variable	`wsrep_dirty_reads`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Boolean
Default Value	`OFF`
Initial Version	MySQL-wsrep: 5.6.29-25.14, MariaDB: 10.1.3

When a node loses its connection to the *Primary Component*, it enters a non-operational state. Given that it can't keep its data current while in this state, it rejects all queries with an `ERROR: Unknown command` message. This parameter determines whether or not the node permits reads while in a non-operational state.

Note: Remember that by its nature, data reads from nodes in a non-operational state are stale. Current data in the Primary Component remains inaccessible to these nodes until they rejoin the cluster.

When enabling this parameter, the node only permits reads. It still rejects any command that modifies or updates the database. When in this state, the node allows `USE`, `SELECT`, `LOCK TABLE` and `UNLOCK TABLES` statements. It does not allow DDL statements. It also rejects DML statements (that is, `INSERT`, `DELETE` and `UPDATE`).

You must set the *wsrep_sync_wait* (page 265) parameter to `0` when using this parameter, else it raises a deadlock error.

You can execute the following `SHOW VARIABLES` statement with a `LIKE` operator to see if this variable is enabled:

```
SHOW VARIABLES LIKE 'wsrep_dirty_reads';

+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| wsrep_dirty_reads | ON    |
+-------------------+-------+
```

Note: This is a MySQL wsrep parameter. It was introduced in version 5.6.29.

---

wsrep_drupal_282555_workaround

This parameter enables workaround for a bug in MySQL InnoDB that affects Drupal installations.

Command-line Format	`--wsrep-drupal-282555-workaround`
System Variable	`wsrep_drupal_282555_workaround`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Boolean
Default Value	`ON`
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

Drupal installations using MySQL are subject to a bug in InnoDB, tracked as MySQL Bug 41984 and Drupal Issue 282555. Specifically, inserting a *DEFAULT* value into an *AUTO_INCREMENT* column may return duplicate key errors.

This parameter enables a workaround for the bug on Galera Cluster.

You can execute the following `SHOW VARIABLES` statement with a `LIKE` operator to see if this variable is enabled:

```
SHOW VARIABLES LIKE 'wsrep_drupal_28255_workaround';

+-------------------------------+-------+
| Variable_name                 | Value |
+-------------------------------+-------+
| wsrep_drupal_28255_workaround | ON    |
+-------------------------------+-------+
```

wsrep_forced_binlog_format

This parameter defines the binary log format for all transactions.

Command-line Format	`--wsrep-forced-binlog-format`
System Variable	`wsrep_forced_binlog_format`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Enumeration
Default Value	`NONE`
Valid Values	`ROW`, `STATEMENT`, `MIXED`, `NONE`
Initial Version	MySQL-wsrep: 5.5.17-22.3, MariaDB: 5.5.21

The node uses the format given by this parameter regardless of the client session variable binlog_format. Valid choices for this parameter are: `ROW`, `STATEMENT`, and `MIXED`. Additionally, there is the special value `NONE`, which means that there is no forced format in effect for the binary logs. When set to a value other than `NONE`, this parameter forces all transactions to use a given binary log format.

This variable was introduced to support `STATEMENT` format replication during *Rolling Schema Upgrade*. In most cases, however, `ROW` format replication is valid for asymmetric schema replication.

If you turn on `wsrep_forced_binlog_format`, it is effective only for DML operations, to avoid any possible binlog corruption. In addition, since MySQL-wsrep 8.0.37-26.19, it is also deprecated, as `binlog_format` has been deprecated upstream since MySQL 8.0.34. As the only possible logging format is ROW, it makes this option redundant.

You can execute the following `SHOW VARIABLES` statement with a `LIKE` operator to see how this variable is set:

```
SHOW VARIABLES LIKE 'wsrep_forced_binlog_format';

+-----------------------------+-------+
| Variable_name               | Value |
+-----------------------------+-------+
| wsrep_forced_binlog_format | NONE  |
+-----------------------------+-------+
```

wsrep_ignore_apply_errors

A bitmask defining whether errors are ignored, or reported back to the provider

- 0: No errors are skipped.
- 1: Ignore some DDL errors (DROP DATABASE, DROP TABLE, DROP INDEX, ALTER TABLE).
- 2: Skip DML errors (Only ignores DELETE errors).
- 4: Ignore all DDL errors.

For example, if you want to ignore some DDL errors (option 1) and skip DML errors (option 2), you would calculate 1+2=3, and use `--wsrep-wsrep_ignore_apply_errors=3`.

Command-line Format	`--wsrep-wsrep_ignore_apply_errors`
System Variable	`wsrep_ignore_apply_errors`
Variable Scope	Global
Dynamic Variable	Yes
Data Type	Numeric
Default Value	7
Range	0 to 7
Initial Version	Version 1.0

You can execute the following `SHOW VARIABLES` statement with a `LIKE` operator to see how this variable is set:

```
SHOW VARIABLES LIKE 'wsrep-wsrep_ignore_apply_errors';

+---------------------------------+-------+
| Variable_name                   | Value |
+---------------------------------+-------+
| wsrep-wsrep_ignore_apply_errors |   7   |
+---------------------------------+-------+
```

wsrep_info_level

This parameter defines how to log `INFO`-level wsrep messages.

Command-line Format	`--wsrep_info_level`
System Variable	`wsrep_info_level`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Numeric
Default Value	0
Initial Version	MySQL-wsrep: 8.0.34

`INFO`-level wsrep messages are logged with `SYSTEM_LEVEL` priority by default, as WSREP information level messages are crucial for troubleshooting replication issues. However, if you need to use `INFORMATION_LEVEL` logging, you can use this variable to change the logging priority.

The options are:

- `0` Use `SYSTEM_LEVEL` logging.
- `3` Use `INFORMATION_LEVEL` logging.

You can execute the following `SHOW VARIABLES` statement to see how this variable is set:

```
SHOW VARIABLES LIKE 'wsrep_info_level';

+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| wsrep_info_level | 0     |
+------------------+-------+
```

wsrep_load_data_splitting

This parameter defines whether the node splits large `LOAD DATA` commands into more manageable units.

Command-line Format	`--wsrep-load-data-splitting`
System Variable	`wsrep_load_data_splitting`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Boolean
Default Value	`ON`, defaults to `OFF` from MariaDB 10.4.3
Initial Version	MySQL-wsrep: 5.5.34-25.29, MariaDB: 5.5.32
Deprecated Version	MariaDB: 10.4.2, removed in MariaDB 11.5

When loading huge amounts of data creates problems for Galera Cluster, in that they eventually reach a size that is too large for the node to rollback completely the operation in the event of a conflict and whatever gets committed stays committed.

This parameter tells the node to split `LOAD DATA` commands into transactions of 10,000 rows or less, making the data more manageable for the cluster. This deviates from the standard behavior for MySQL.

You can execute the following `SHOW VARIABLES` statement to see how this variable is set:

```
SHOW VARIABLES LIKE 'wsrep_load_data_splitting';

+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| wsrep_load_data_splitting | ON    |
+--------------------------+-------+
```

wsrep_log_conflicts

This parameter defines whether the node logs additional information about conflicts.

Command-line Format	`--wsrep-log-conflicts`
System Variable	`wsrep_log_conflicts`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Boolean
Default Value	`OFF`
Initial Version	MySQL-wsrep: 5.5.28-23.7, MariaDB: 5.5.27

In Galera Cluster, the database server uses the standard logging features of MySQL, MariaDB and Percona XtraDB. This parameter enables additional information for the logs pertaining to conflicts. You may find this useful in troubleshooting replication problems. You can also log conflict information with the wsrep Provider option *cert.log_conflicts* (page 279).

The additional information includes the table and schema where the conflict occurred, as well as the actual values for the keys that produced the conflict.

You can execute the following `SHOW VARIABLES` statement to see if this feature is enabled:

```
SHOW VARIABLES LIKE 'wsrep_log_conflicts';

+---------------------+-------+
| Variable_name       | Value |
+---------------------+-------+
| wsrep_log_conflicts | OFF   |
+---------------------+-------+
```

wsrep_max_ws_rows

With this parameter you can set the maximum number of rows the node allows in a write-set.

Command-line Format	`--wsrep-max-ws-rows`
System Variable	`wsrep_max_ws_rows`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	String
Default Value	`0`
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

If set to a value greater than `0`, this parameter sets the maximum number of rows that the node allows in a write-set.

You can execute the following `SHOW VARIABLES` statement to see the current value of this parameter:

```
SHOW VARIABLES LIKE 'wsrep_max_ws_rows';

+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| wsrep_max_ws_rows | 128   |
+------------------+-------+
```

wsrep_max_ws_size

You can set the maximum size the node allows for write-sets with this parameter.

Command-line Format	`--wsrep-max-ws-size`
System Variable	`wsrep_max_ws_size`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	String
Default Value	`2G`
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

This parameter sets the maximum size that the node allows for a write-set. Currently, this value limits the supported size of transactions and of `LOAD DATA` statements.

The maximum allowed write-set size is `2G`. You can execute the following `SHOW VARIABLES` statement to see the current value of this parameter:

```
SHOW VARIABLES LIKE 'wsrep_max_ws_size';

+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| wsrep_max_ws_size | 2G    |
+--------------------+-------+
```

wsrep_mode

Extends node behaviour with provided values.

Command-line Format	`--wsrep_mode`
System Variable	`wsrep_mode`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Set
Default Value	See the information below.
Initial Version	MySQL-wsrep: 5.7.32-25.24, 8.0.22-26.5, MariaDB: 10.6.0

The options for MySQL are:

- `IGNORE_NATIVE_REPLICATION_FILTER_RULES` - Ignore native replication filter rules for cluster events. In other words, native asynchronous replication filtering options are honored when applying Galera replication. These options are of format `replicate_*`, and specify if transactions for a table or a database should be applied or not.
- `IGNORE_CASCADING_FK_DELETE_MISSING_ROW_ERROR` - Ignore missing row errors when applying a cascading delete write set. This a workaround for https://bugs.mysql.com/bug.php?id=80821, and is possibly obsolete in the upstream versions.
- `APPLIER_IGNORE_MISSING_TABLE` - MySQL has an anomaly to sometimes add an excessive tablemap event in the binlog. This can happen in use cases related to multi-table updates and trigger definitions to a third table, which is not effectively needed in applying of the replication events. With `wsrep_mode` set to `APPLIER_IGNORE_MISSING_TABLE`, the replication applier will ignore the failure to open such a table, which would not be used in the actual applying. This is the default value for MySQL.
- `APPLIER_SKIP_FK_CHECKS_IN_IST` - In normal operation, appliers must verify foreign key constraints in multi-active topologies. Thus, appliers are configured to enable FK checking. However, during node joining, in IST and latter catch up period, the node is still idle from local connections, and the only source for incoming

transactions is the cluster sending certified write sets for applying. IST happens with parallel applying, and there is a possibility that a foreign key check causes lock conflicts between appliers accessing FK child and parent tables. Also, the excessive FK checking will slow down IST process. When this mode is set, and the node is processing IST or catch up, appliers will skip FK checking.

The options for MariaDB are:

- `BINLOG_ROW_FORMAT_ONLY` - Only `ROW` binlog format is supported.
- **`DISALLOW_LOCAL_GTID` - Nodes can have GTIDs for local transactions in a number of scenarios. If `DISALLOW_LOCAI`**
  - A DDL statement is executed with `wsrep_OSU_method=RSU` set.
  - A DML statement writes to a non-InnoDB table.
  - A DML statement writes to an InnoDB table with `wsrep_on=OFF` set.
- `REPLICATE_ARIA` - Together with `wsrep_mode=REPLICATE_MYISAM`, this parameter enables Galera to replicate both DDL and DML for ARIA and/or MyISAM using TOI. This option requires a primary key for the replicated table. To use this mode, set on `REQUIRED_PRIMARY_KEY,REPLICATE_MYISAM, REPLICATE_ARIA`.
- `REPLICATE_MYISAM` - Together with `wsrep_mode=REPLICATE_ARIA`, this parameter enables Galera to replicate both DDL and DML for ARIA and/or MyISAM using TOI. This option requires a primary key for the replicated table. To use this mode, set on `REQUIRED_PRIMARY_KEY,REPLICATE_MYISAM, REPLICATE_ARIA`.
- `REQUIRED_PRIMARY_KEY` - The table must have a primary key defined.
- `STRICT_REPLICATION` - The same as the old `wsrep_strict_ddl` setting (which was deprecated in 10.6, and removed in 10.7).
- `BF_ABORT_MARIABACKUP` - With this option, backup execution can be aborted if DDL statements take place during the backup execution. Note that node desync and pause operations are still needed, if the node is operating as an SST donor.
- (Empty) - Giving no value does not change the node behavior. This is the default value for MariaDB.

The options for Percona XtraDB Cluster (PXC) are:

- `IGNORE_NATIVE_REPLICATION_FILTER_RULES` - Ignore native replication filter rules for cluster events.
- (Empty) - Giving no value does not change the node behavior. This is the default value for Percona XtraDB Cluster (PXC).

```
SET GLOBAL wsrep_mode = IGNORE_NATIVE_REPLICATION_FILTER_RULES;

SHOW VARIABLES LIKE 'wsrep_mode';

+---------------+-----------------------------------------+
| Variable_name | Value                                   |
+---------------+-----------------------------------------+
| wsrep_mode    | IGNORE_NATIVE_REPLICATION_FILTER_RULES |
+---------------+-----------------------------------------+
```

wsrep_node_address

This parameter is used to note the IP address and port of the node.

Command-line Format	`--wsrep-node-address`
System Variable	`wsrep_node_address`
Variable Scope	Global
Dynamic Variable	No
Permitted Values	String
Default Value	Server IP Address, Port `4567`
Initial Version	MySQL-wsrep: 5.5.20-23.4, MariaDB: 5.5.21

The node passes its IP address and port number to the *Galera Replication Plugin*, where it is used as the base address in cluster communications. By default, the node pulls the address of the first network interface and uses the default port for Galera Cluster. Typically, this is the address of `eth0` or `enp2s0` on port `4567`.

While the default behavior is often sufficient, there are situations in which this auto-guessing function produces unreliable results. Some common reasons are the following:

- 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 with Docker and jails; and
- Cloud deployments, such as with Amazon EC2 and OpenStack.

In these scenarios, since auto-guess of the IP address does not produce the correct result, you will need to provide an explicit value for this parameter.

Note: In addition to defining the node address and port, this parameter also provides the default values for the *wsrep_sst_receive_address* (page 263) parameter and the *ist.recv_addr* (page 296) option.

In some cases, you may need to provide a different value. For example, Galera Cluster running on Amazon EC2 requires that you use the global DNS name instead of the local IP address.

You can execute the `SHOW VARIABLES` statement as shown below to get the current value of this parameter:

```
SHOW VARIABLES LIKE 'wsrep_node_address';

+--------------------+-------------+
| Variable_name      | Value       |
+--------------------+-------------+
| wsrep_node_address | 192.168.1.1 |
+--------------------+-------------+
```

wsrep_node_incoming_address

This parameter is used to provide the IP address and port from which the node should expect client connections.

Command-line Format	`--wsrep-node-incoming-address`
System Variable	`wsrep_node_incoming_address`
Variable Scope	Global
Dynamic Variable	No
Permitted Values	String
Default Value	
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

This parameter defines the IP address and port number at which the node should expect to receive client connections. It is intended for integration with load balancers. For now, it is otherwise unused by the node.

You can execute the `SHOW VARIABLES` statement with the `LIKE` operator as shown below to get the IP address and port setting of this parameter:

```
SHOW VARIABLES LIKE 'wsrep_node_incoming_address';

+----------------------------+------------------+
| Variable_name              | Value            |
+----------------------------+------------------+
| wsrep_node_incoming_address | 192.168.1.1:3306 |
+----------------------------+------------------+
```

wsrep_node_name

You can set the logical name that the node uses for itself with this parameter.

Command-line Format	`--wsrep-node-name`
System Variable	`wsrep_node_name`
Variable Scope	Global
Dynamic Variable	No
Permitted Values	String
Default Value	Server Hostname
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

This parameter defines the logical name that the node uses when referring to itself in logs and in the cluster. It's for convenience, to help you in identifying nodes in the cluster by means other than the node address.

By default, the node uses the server hostname. In some situations, you may need explicitly to set it. You would do this when using container deployments with Docker or FreeBSD jails, where the node uses the name of the container rather than the hostname.

You can execute the `SHOW VARIABLES` statement with the `LIKE` operator as shown below to get the node name:

```
SHOW VARIABLES LIKE 'wsrep_node_name';

+-----------------+-------------+
| Variable_name   | Value       |
+-----------------+-------------+
| wsrep_node_name | GaleraNode1 |
+-----------------+-------------+
```

wsrep_notify_cmd

Defines the command the node runs whenever cluster membership or the state of the node changes.

Command-line Format	`--wsrep-notify-cmd`
System Variable	`wsrep_notify_cmd`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	String
Default Value	(empty)
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

Whenever the node registers changes in cluster membership or its own state, this parameter allows you to send information about that change to an external script defined by the value. You can use this to reconfigure load balancers, raise alerts and so on, in response to node and cluster activity.

Warning: The node will block and wait until the script completes and returns before it can proceed. If the script performs any potentially blocking or long-running operations, such as network communication, you may wish initiate such operations in the background and have the script return immediately.

For an example script that updates two tables on the local node, with changes taking place at the cluster level, see the *Notification Command* (page 201).

When the node calls the command, it passes one or more arguments that you can use in configuring your custom notification script and how it responds to the change. Below are these options and explanations of each:

Option	Purpose	Possible Values
`--status <status str>`	The status of this node.	`Undefined` The node has just started up and is not connected to any *Primary Component*.
		`Joiner` The node is connected to a primary component and now is receiving state snapshot.
		`Donor` The node is connected to primary component and now is sending state snapshot.
		`Joined` The node has a complete state and now is catching up with the cluster.
		`Synced` The node has synchronized itself with the cluster.
		`Error(<error code if available>)` The node is in an error state.
`--uuid <state UUID>`	The cluster state UUID.	
`--primary <yes/no>`	Whether the current cluster component is primary or not.	
`--members <list>`	A comma-separated list of the component member UUIDs.	`<node UUID>`; A unique node ID. The wsrep Provider automatically assigns this ID for each node.
		`<node name>`; The node name as it is set in the `wsrep_node_name` option.
		`<incoming address>`; The address for client connections as it is set in the `wsrep_node_incoming_address` option.
`--index`	The index of this node in the node list.	

```
SHOW VARIABLES LIKE 'wsrep_notify_cmd';

+------------------+--------------------------+
| Variable_name    | Value                    |
+------------------+--------------------------+
| wsrep_notify_cmd | /usr/bin/wsrep_notify.sh |
+------------------+--------------------------+
```

wsrep_on

Defines whether replication takes place for updates from the current session.

Command-line Format	`--wsrep_on`
System Variable	`wsrep_on`
Variable Scope	Session
Dynamic Variable	Yes
Permitted Values	Boolean
Default Value	`ON`
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

This parameter defines whether or not updates made in the current session replicate to the cluster. It does not cause the node to leave the cluster and the node continues to communicate with other nodes. Additionally, it is a session

variable. Defining it through the `SET GLOBAL` syntax also affects future sessions.

```
SHOW VARIABLES LIKE 'wsrep_on';

+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| wsrep_on      | ON    |
+---------------+-------+
```

wsrep_OSU_method

Defines the Online Schema Upgrade method the node uses to replicate DDL statements.

Command-line Format	`--wsrep-OSU-method`
System Variable	`wsrep_OSU_method`
Variable Scope	Global, Session
Dynamic Variable	Yes
Permitted Values	Enumeration
Default Value	`TOI`
Valid Values	`TOI`, `RSU`, `NBO`
Initial Version	MySQL-wsrep: 5.5.17-22.3, MariaDB: 5.5.21
Initial Version, NBO	MariaDB Enterprise Server Version 10.5, MySQL-wsrep 8.0.28-26.10 Enterprise Edition, Percona XtraDB Cluster 8.0.25-15.1

DDL statements are non-transactional and as such do not replicate through write-sets. There are two methods available that determine how the node handles replicating these statements:

- `TOI` In the *Total Order Isolation* method, the cluster runs the DDL statement on all nodes in the same total order sequence, blocking other transactions from committing while the DDL is in progress.
- `RSU` In the *Rolling Schema Upgrade* method, the node runs the DDL statements locally, thus blocking only the one node where the statement was made. While processing the DDL statement, the node is not replicating and may be unable to process replication events due to a table lock. Once the DDL operation is complete, the node catches up and syncs with the cluster to become fully operational again. The DDL statement or its effects are not replicated; the user is responsible for manually executing this statement on each node in the cluster.
- `NBO` In the *Non-Blocking Operations* method, the cluster runs the DDL statement on all nodes in the same total order sequence, blocking other transactions from committing while the DDL is in progress. In comparison with TOI, the NBO method has more efficient locking for several operations, as the NBO method issues metadata locks on all nodes at the start of the DDL operation, to ensure consistency. This prevents the TOI issue of long-running DDL statements, which block cluster updates.

For more information on DDL statements and OSU methods, see *Schema Upgrades* (page 83).

```
SHOW VARIABLES LIKE 'wsrep_OSU_method';

+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| wsrep_OSU_method | TOI   |
+------------------+-------+
```

wsrep_preordered

Defines whether the node uses transparent handling of preordered replication events.

Command-line Format	`--wsrep-preordered`
System Variable	`wsrep_preordered`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Boolean
Default Value	`OFF`
Initial Version	MySQL-wsrep: 5.6.21-25.9
Deprecated Version	MySQL-wsrep: 8.0.19-26.3, MariaDB: 10.1.1

This parameter enables transparent handling of preordered replication events, such as replication events arriving from traditional asynchronous replication. When this option is `ON`, such events will be applied locally first before being replicated to the other nodes of the cluster. This could increase the rate at which they can be processed which would be otherwise limited by the latency between the nodes in the cluster.

Preordered events should not interfere with events that originate on the local node. Therefore, you should not run local update queries on a table that is also being updated through asynchronous replication.

```
SHOW VARIABLES LIKE 'wsrep_preordered';

+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| wsrep_preordered | OFF   |
+------------------+-------+
```

wsrep_provider

Defines the path to the *Galera Replication Plugin*.

Command-line Format	`--wsrep-provider`
System Variable	`wsrep_provider`
Variable Scope	Global
Dynamic Variable	No
Permitted Values	File
Default Value	None
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

When the node starts, it needs to load the wsrep Provider in order to enable replication functions. The path defined in this parameter tells it what file it needs to load and where to find it. In the event that you do not define this path or you give it an invalid value, the node bypasses all calls to the wsrep Provider and behaves as a standard standalone instance of MySQL.

```
SHOW VARIABLES LIKE 'wsrep_provider';

+----------------+----------------------------------+
| Variable_name  | Value                            |
+----------------+----------------------------------+
| wsrep_provider | /usr/lib/galera/libgalera_smm.so |
+----------------+----------------------------------+
```

wsrep_provider_options

Defines optional settings the node passes to the wsrep Provider.

Command-line Format	--wsrep-provider-options
System Variable	wsrep_provider_options
Variable Scope	Global
Dynamic Variable	No
Permitted Values	String
Default Value	(empty)
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

When the node loads the wsrep Provider, there are several configuration options available that affect how it handles certain events. These allow you to fine tune how it handles various situations.

For example, you can use *gcache.size* (page 288) to define how large a write-set cache the node keeps or manage group communications timeouts.

Note: All wsrep_provider_options settings need to be specified on a single line. In case of multiple instances of wsrep_provider_options, only the last one is used.

For more information on the wsrep Provider options, see *Galera Parameters* (page 271) and *plugin_wsrep_provider* (page 233).

```
SHOW VARIABLES LIKE 'wsrep_provider_options';

+-------------------------+-------------------------------------------------+
| Variable_name           | Value                                           |
+-------------------------+-------------------------------------------------+
| wsrep_provider_options  | ... evs.user_send_window=2,gcache.size=128Mb    |
|                         | evs.auto_evict=0,debug=OFF, evs.version=0 ...   |
+-------------------------+-------------------------------------------------+
```

wsrep_recover

If ON, when the server starts, the server will recover the sequence number of the most recent write set applied by Galera, and it will be output to stderr, which is usually redirected to the error log. At that point, the server will exit. This sequence number can be provided to the wsrep_start_position system variable.

Command-line Format	--wsrep-recover
System Variable	wsrep_recover
Variable Scope	Global
Dynamic Variable	No
Permitted Values	0 | 1
Default Value	OFF
Initial Version	MySQL-wsrep: 5.5.23-23.5, MariaDB: 5.5.21

See also Restarting the Cluster and wsrep_recover Script.

```
SHOW VARIABLES LIKE 'wsrep_recover';
```

(continues on next page)

(continued from previous page)

```
+------------------------+-------------------------------------------------+
| Variable_name          | Value                                           |
+------------------------+-------------------------------------------------+
| wsrep_recover          | OFF                                             |
+------------------------+-------------------------------------------------+
```

wsrep_reject_queries

Defines whether the node rejects client queries while participating in the cluster.

Command-line Format	
System Variable	`wsrep_reject_queries`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Array
Default Value	`NONE`
Valid Values	`NONE`, `ALL`, `ALL_KILL`
Initial Version	MySQL-wsrep: 5.6.29-25.14, MariaDB: 10.1.32

When in use, this parameter causes the node to reject queries from client connections. The node continues to participate in the cluster and apply write-sets, but client queries generate `Unknown command` errors. For instance,

```
SELECT * FROM my_table;

Error 1047: Unknown command
```

You may find this parameter useful in certain maintenance situations. In enabling it, you can also decide whether or not the node maintains or kills any current client connections.

- `NONE` The node disables this feature.
- `ALL` The node enables this feature. It rejects all queries, but maintains any existing client connections.
- `ALL_KILL` The node enables this feature. It rejects all queries and kills existing client connections without waiting, including the current connection.

```
SHOW VARIABLES LIKE 'wsrep_reject_queries';

+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| wsrep_reject_queries | NONE  |
+----------------------+-------+
```

Note: This is a MySQL wsrep parameter. It was introduced in version 5.6.29.

wsrep_restart_replica

Defines whether the replica restarts when the node joins the cluster.

Command-line Format	`--wsrep-restart-replica`
System Variable	`wsrep_restart_replica`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Boolean
Default Value	`OFF`
Initial Version	MySQL-wsrep: 8.0.26-26.8

Enabling this parameter tells the node to restart the replica when it joins the cluster.

```
SHOW VARIABLES LIKE 'wsrep_restart_replica';

+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| wsrep_restart_replica | OFF   |
+-----------------------+-------+
```

wsrep_restart_slave

Deprecated as of Galera Cluster 4.10/MySQL-wsrep 8.0.26-26.8 in favor of *wsrep_restart_replica* (page 256).

wsrep_retry_autocommit

Defines the number of retries the node attempts when an autocommit query fails.

Command-line Format	`--wsrep-retry-autocommit`
System Variable	`wsrep_retry_autocommit`
Variable Scope	Global
Dynamic Variable	No
Permitted Values	Integer
Default Value	`1`
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

When an autocommit query fails the certification test due to a cluster-wide conflict, the node can retry it without returning an error to the client. This parameter defines how many times the node retries the query. It is analogous to rescheduling an autocommit query should it go into deadlock with other transactions in the database lock manager.

```
SHOW VARIABLES LIKE 'wsrep_retry_autocommit';

+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| wsrep_retry_autocommit | 1     |
+------------------------+-------+
```

wsrep_applier_FK_checks

Defines whether the node performs foreign key checking for applier threads.

Command-line Format	`--wsrep-applier-FK-checks`
System Variable	`wsrep_applier_FK_checks`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Boolean
Default Value	`ON`
Initial Version	MySQL-wsrep: 8.0.26-26.8

This parameter enables foreign key checking on applier threads.

```
SHOW VARIABLES LIKE 'wsrep_applier_FK_checks';

+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| wsrep_applier_FK_checks | ON    |
+--------------------------+-------+
```

wsrep_slave_FK_checks

Command-line Format	`--wsrep-slave-FK-checks`
System Variable	`wsrep_slave_FK_checks`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Boolean
Default Value	`ON`
Initial Version	MySQL-wsrep: 5.5.42-25.11, MariaDB: 10.0.12
Deprecated Version	MySQL-wsrep: 8.0.26-26.8

Deprecated as of Galera Cluster 4.10/MySQL-wsrep 8.0.26-26.8 in favor of *wsrep_applier_FK_checks* (page 257).

wsrep_applier_threads

Defines the number of threads to use in applying of write-sets.

Command-line Format	`--wsrep-applier-threads`
System Variable	`wsrep_applier_threads`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Integer
Default Value	`1`
Initial Version	MySQL-wsrep: 8.0.26-26.8

This parameter allows you to define how many threads the node uses when applying write-sets. Performance on the underlying system and hardware, the size of the database, the number of client connections, and the load your application puts on the server all factor in the need for threading, but not in a way that makes the scale of that need easy to predict. Because of this, there is no strict formula to determine how many applier threads your node actually needs.

Instead of concrete recommendations, there are some general guidelines that you can use as a starting point in finding the value that works best for your system:

- It is rarely beneficial to use a value that is less than twice the number of CPU cores on your system.
- Similarly, it is rarely beneficial to use a value that is more than one quarter the total number of client connections to the node. While it is difficult to predict the number of client connections, being off by as much as 50% over or under is unlikely to make a difference.
- From the perspective of resource utilization, it's recommended that you keep to the lower end of applier threads.

```
SHOW VARIABLES LIKE 'wsrep_applier_threads';

+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| wsrep_applier_threads | 1     |
+-----------------------+-------+
```

wsrep_slave_threads

Command-line Format	`--wsrep-slave-threads`
System Variable	`wsrep_slave_threads`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Integer
Default Value	`1`
Initial Version	MySQL-wsrep: 5.1.58-25.11, MariaDB: 5.5.21
Deprecated Version	MySQL-wsrep: 8.0.26-26.8

Deprecated as of MySQL-wsrep 8.0.26-26.8 in favor of *wsrep_applier_threads* (page 258). See also Setting Parallel Replica Threads.

wsrep_applier_UK_checks

Defines whether the node performs unique key checking on applier threads.

Command-line Format	`--wsrep-applier-UK-checks`
System Variable	`wsrep_applier_UK_checks`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Boolean
Default Value	`OFF`
Initial Version	MySQL-wsrep: 8.0.26-26.8

This parameter enables unique key checking on applier threads.

```
SHOW VARIABLES LIKE 'wsrep_applier_UK_checks';

+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| wsrep_applier_UK_checks | OFF   |
+--------------------------+-------+
```

wsrep_slave_UK_checks

Command-line Format	`--wsrep-slave-UK-checks`
System Variable	`wsrep_slave_UK_checks`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Boolean
Default Value	`OFF`
Initial Version	MySQL-wsrep: 5.5.42-25.11, MariaDB: 5.5.21
Deprecated Version	MySQL-wsrep: 8.0.26-26.8

Deprecated as of MySQL-wsrep 8.0.26-26.8 in favor of *wsrep_applier_UK_checks* (page 259).

wsrep_sst_auth

Defines the authentication information to use in *State Snapshot Transfer*.

Command-line Format	`--wsrep-sst-auth`
System Variable	`wsrep_sst_auth`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	String
Default Value	(empty)
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

When the node attempts a state snapshot transfer using the *Logical State Transfer Method*, the transfer script uses a client connection to the database server in order to obtain the data it needs to send. This parameter provides the authentication information, (that is, the username and password), that the script uses to access the database servers of both sending and receiving nodes.

Note: Galera Cluster only uses this parameter for State Snapshot Transfers that use the Logical transfer method. Currently, the only method to use the Logical transfer method is `mysqldump`. For all other methods, the node does not need this parameter.

Format this value to the pattern: `username:password`.

```
SHOW VARIABLES LIKE 'wsrep_sst_auth'

+----------------+---------------------------+
| Variable_name  | Value                     |
+----------------+---------------------------+
| wsrep_sst_auth | wsrep_sst_user:mypassword |
+----------------+---------------------------+
```

wsrep_sst_donor

Defines the name of the node that this node uses as a donor in state transfers.

Command-line Format	`--wsrep-sst-donor`
System Variable	`wsrep_sst_donor`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	String
Default Value	
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

When the node requires a state transfer from the cluster, it looks for the most appropriate one available. The group communications module monitors the node state for the purposes of Flow Control, state transfers and *Quorum* calculations. The node can be a donor if it is in the `SYNCED` state. The first node in the `SYNCED` state in the index becomes the donor and is made unavailable for requests while serving as such.

If there are no free `SYNCED` nodes at the moment, the joining node reports in the logs:

```
Requesting state transfer failed: -11(Resource temporarily unavailable).
 Will keep retrying every 1 second(s)
```

It continues retrying the state transfer request until it succeeds. When the state transfer request does succeed, the node makes the following entry in the logs:

```
Node 0 (XXX) requested state transfer from '*any*'. Selected 1 (XXX) as donor.
```

Using this parameter, you can tell the node which cluster node or nodes it should use instead for state transfers. The names used in this parameter must match the names given with *wsrep_node_name* (page 250) on the donor nodes. The setting affects both Incremental State Transfers (IST) and Snapshot State Transfers (SST).

If the list contains a trailing comma, the remaining nodes in the cluster will also be considered if the nodes from the list are not available.

```
SHOW VARIABLES LIKE 'wsrep_sst_donor';

+-----------------+---------------------------------+
| Variable_name   | Value                           |
+-----------------+---------------------------------+
| wsrep_sst_donor | my_donor_node1,my_donor_node2, |
+-----------------+---------------------------------+
```

wsrep_sst_donor_rejects_queries

Defines whether the node rejects blocking client sessions on a node when it is serving as a donor in a blocking state transfer method, such as `mysqldump` and `rsync`.

Command-line Format	`--wsrep-sst-donor-rejects-queries`
System Variable	`wsrep_sst_donor_rejects_queries`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	Boolean
Default Value	`OFF`
Initial Version	MySQL-wsrep: 5.5.28-23.7, MariaDB: 5.5.28

This parameter determines whether the node rejects blocking client sessions while it is sending state transfers using methods that block it as the donor. In these situations, all queries return the error `ER_UNKNOWN_COM_ERROR`, that is they respond with `Unknown command`, just like the joining node does.

Given that a *State Snapshot Transfer* is scriptable, there is no way to tell whether the requested method is blocking or not. You may also want to avoid querying the donor even with non-blocking state transfers. As a result, when this parameter is enabled the *Donor Node* rejects queries regardless the state transfer and even if the initial request concerned a blocking-only transfer, (meaning, it also rejects during `xtrabackup`).

Warning: The `mysqldump` state transfer method does not work with the `wsrep_sst_donor_rejects_queries` parameter, given that `mysqldump` runs queries on the donor and there is no way to differentiate its session from the regular client session.

```
SHOW VARIABLES LIKE 'wsrep_sst_donor_rejects_queries';

+---------------------------------+-------+
| Variable_name                   | Value |
+---------------------------------+-------+
| wsrep_sst_donor_rejects_queries | OFF   |
+---------------------------------+-------+
```

wsrep_sst_method

Defines the method or script the node uses in a *State Snapshot Transfer*.

Command-line Format	`--wsrep-sst-method`
System Variable	`wsrep_sst_method`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	String
Default Value	`rsync`
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

When the node makes a state transfer request it calls on an external shell script to establish a connection a with the donor node and transfer the database state onto the local database server. This parameter allows you to define what script the node uses in requesting state transfers.

Galera Cluster ships with a number of default scripts that the node can use in state snapshot transfers. The supported methods are:

- `mysqldump` This is slow, except for small data-sets, but is the most tested option.
- `rsync` This option is much faster than `mysqldump` on large data-sets.

  Note: You can only use `rsync` when a node is starting. You cannot use it with a running InnoDB storage engine.

- `rsync_wan` This option is almost the same as `rsync`, but uses the `delta-xfer` algorithm to minimize network traffic. On MariaDB Server `rsync_wan` is a symlink to `rsync`.
- `mariabackup` This option uses the Mariabackup utility for performing SSTs. See mariabackup-options.
- `xtrabackup` This option is a fast and practically non-blocking state transfer method based on the Percona `xtrabackup` tool. If you want to use it, the following settings must be present in the `my.cnf` configuration file on all nodes:

```
[mysqld]
wsrep_sst_auth=YOUR_SST_USER:YOUR_SST_PASSWORD
wsrep_sst_method=xtrabackup
datadir=/path/to/datadir

[client]
socket=/path/to/socket
```

- `xtrabackup-v2` This option is a fast and practically non-blocking state transfer method based on the Percona `xtrabackup-v2` tool. If you want to use it, the following settings must be present in the `my.cnf` configuration file on all nodes:

```
[mysqld]
wsrep_sst_auth=YOUR_SST_USER:YOUR_SST_PASSWORD
wsrep_sst_method=xtrabackup-v2
datadir=/path/to/datadir

[client]
socket=/path/to/socket
```

- `wsrep_recover` See Crash Recovery.
- `clone` The fastest SST method. See *Enabling SSL for clone based SST* (page 223). Available as of Galera Cluster version 8.0.
- `common` A common command line parser to be sourced by other SST scripts. On MariaDB Server, available from version 10.11.

In addition to the default scripts provided and supported by Galera Cluster, you can also define your own custom state transfer script. The naming convention that the node expects is for the value of this parameter to match `wsrep_%.sh`. For instance, giving the node a transfer method of `MyCustomSST` causes it to look for `wsrep_MyCustomSST.sh` in `/usr/bin`.

Bear in mind, the cluster uses the same script to send and receive state transfers. If you want to use a custom state transfer script, you need to place it on every node in the cluster.

For more information on scripting state snapshot transfers, see *Scriptable State Snapshot Transfers* (page 74).

```
SHOW VARIABLES LIKE 'wsrep_sst_method';

+------------------+-----------+
| Variable_name    | Value     |
+------------------+-----------+
| wsrep_sst_method | mysqldump |
+------------------+-----------+
```

wsrep_sst_receive_address

Defines the address from which the node expects to receive state transfers.

Command-line Format	`--wsrep-sst-receive-address`
System Variable	`wsrep_sst_receive_address`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	String
Default Value	*wsrep_node_address* (page 248)
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

This parameter defines the address from which the node expects to receive state transfers. It is dependent on the *State Snapshot Transfer* method the node uses.

For example, `mysqldump` uses the address and port on which the node listens, which by default is set to the value of *wsrep_node_address* (page 248).

Note: Check that your firewall allows connections to this address from other cluster nodes.

```
SHOW VARIABLES LIKE 'wsrep_sst_receive_address';

+---------------------------+-------------+
| Variable_name             | Value       |
+---------------------------+-------------+
| wsrep_sst_receive_address | 192.168.1.1 |
+---------------------------+-------------+
```

wsrep_start_position

Defines the node start position.

Command-line Format	`--wsrep-start-position`
System Variable	`wsrep_start_position`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	String
Default Value	`00000000-0000-0000-0000-000000000000:-1/0/0/ 00000000-0000-0000-0000-000000000000`
Initial Version	MySQL-wsrep: 5.1.58-21.1, MariaDB: 5.5.21

This parameter defines the node start position. It contains the wsrep GTID position, local seqno for asynchronous replication, server ID and server UUID all in one, slash-separated argument. It exists for the sole purpose of notifying the joining node of the completion of a state transfer.

For more information on scripting state snapshot transfers, see *Scriptable State Snapshot Transfers* (page 74).

```
SHOW VARIABLES LIKE 'wsrep_start_position';

+----------------------+-------------------------------------------------------------------
↪--------------------+
| Variable_name        | Value
↪                    |
+----------------------+-------------------------------------------------------------------
↪--------------------+
| wsrep_start_position | 00000000-0000-0000-0000-000000000000:-1/0/0/00000000-0000-
↪0000-0000-000000000000 |
+----------------------+-------------------------------------------------------------------
↪--------------------+
```

wsrep_status_file

Defines the file name for node status output.

Command-line Format	`--wsrep-status-file`
System Variable	`wsrep_status_file`
Variable Scope	Global
Dynamic Variable	No
Permitted Values	String
Default Value	None
Initial Version	MySQL-wsrep 8.0.26-26.8

If defined, the file will contain JSON formatted output of node status. The purpose of the file is to provide a machine readable view of the current node status which is available all the time after the node is started.

The contents of the file are subject to change.

```
SHOW VARIABLES LIKE 'wsrep_status_file';

--------------------+--------------------+
| Variable_name      | Value              |
+--------------------+--------------------+
| wsrep_status_file | wsrep-status.json |
+--------------------+--------------------+
```

wsrep_sync_server_uuid

Sets the node to use the server UUID received from the donor node.

Command-line Format	`--wsrep_sync_server_uuid`
System Variable	`wsrep_sync_server_uuid`
Variable Scope	Global
Dynamic Variable	Yes
Permitted Values	String
Default Value	0
Initial Version	MySQL-wsrep 8.0.26-26.8

Unless this variable is set, the wsrep nodes generate individual server UUIDs, which are used on binlog events, such as rolling schema upgrades, that are not replicated through wsrep. This makes individual node histories incomparable and complicates switching asynchronous replica PRIMARY between the nodes in the cluster.

When set, this variable forces the nodes to use the same server UUID (generated on the seed node) to binlog events that are not replicated through wsrep. This makes the histories comparable, provided that the user executes such operations in agreed order on all the nodes..

```
SHOW VARIABLES LIKE 'wsrep_sync_server_uuid';

--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| wsrep_sync_server_uuid  | 1     |
+--------------------------+-------+
```

wsrep_sync_wait

Defines whether the node enforces strict cluster-wide causality checks.

Command-line Format	`--wsrep-sync-wait`
System Variable	`wsrep_sync_wait`
Variable Scope	Session
Dynamic Variable	Yes
Permitted Values	Bitmask
Default Value	`0`
Initial Version	MySQL-wsrep: 5.5.42-25.12, MariaDB: 10.0.13

When you enable this parameter, the node triggers causality checks in response to certain types of queries. During the check, the node blocks new queries while the database server catches up with all updates made in the cluster to the point where the check was begun. Once it reaches this point, the node executes the original query.

Note: Causality checks of any type can result in increased latency.

This value of this parameter is a bitmask, which determines the type of check you want the node to run.

Bitmask	Checks
`0`	Disabled.
`1`	Checks on `READ` statements, including `SELECT`, and `BEGIN` / `START TRANSACTION`. Checks on `SHOW` (up to versions 5.5.54, 5.6.35, 5.7.17)
`2`	Checks made on `UPDATE` and `DELETE` statements.
`3`	Checks made on `READ`, `UPDATE` and `DELETE` statements.
`4`	Checks made on `INSERT` and `REPLACE` statements.
`5`	Checks made on `READ`, `INSERT` and `REPLACE` statements.
`6`	Checks made on `UPDATE`, `DELETE`, `INSERT` and `REPLACE` statements.
`7`	Checks made on `READ`,‘‘UPDATE‘‘, `DELETE`, `INSERT` and `REPLACE` statements.
`8`	Checks made on `SHOW` statements.
`9`	Checks made on `READ` and `SHOW` statements.
`10`	Checks made on `UPDATE`, `DELETE` and `SHOW` statements.
`11`	Checks made on `READ`, `UPDATE`, `DELETE` and `SHOW` statements.
`12`	Checks made on `INSERT`, `REPLACE`, and `SHOW` statements.
`13`	Checks made on `READ`, `INSERT`, `REPLACE`, and `SHOW` statements.
`14`	Checks made on `UPDATE`, `DELETE`, `INSERT`, `REPLACE`, and `SHOW` statements.
`15`	Checks made on `READ`, `UPDATE`, `DELETE`, `INSERT`, `REPLACE`, and `SHOW` statements.

For example, say that you have a web application. At one point in its run, you need it to perform a critical read. That is, you want the application to access the database server and run a `SELECT` query that must return the most up to date information possible.

```
SET SESSION wsrep_sync_wait=1;
SELECT * FROM example WHERE field = "value";
SET SESSION wsrep_sync_wait=0
```

In the example, the application first runs a `SET` command to enable *wsrep_sync_wait* (page 265) for `READ` statements, then it makes a `SELECT` query. Rather than running the query, the node initiates a causality check, blocking incoming queries while it catches up with the cluster. When the node finishes applying the new transaction, it executes the `SELECT` query and returns the results to the application. The application, having finished the critical read, disables *wsrep_sync_wait* (page 265), returning the node to normal operation.

Note: Setting *wsrep_sync_wait* (page 265) to `1` is the same as *wsrep_causal_reads* (page 235) to `ON`. This deprecates

wsrep_causal_reads (page 235).

```
SHOW VARIABLES LIKE 'wsrep_sync_wait';

+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| wsrep_sync_wait | 0     |
+-----------------+-------+
```

wsrep_trx_fragment_size

Defines the number of replication units needed to generate a new fragment in Streaming Replication.

Command-line Format	`--wsrep-trx-fragment-size`
System Variable	`wsrep_trx_fragment_size`
Variable Scope	Session
Dynamic Variable	Yes
Permitted Values	Integer
Default Value	`0`
Initial Version	MySQL-wsrep: 8.0.19-26.3, MariaDB: 10.4.2

In *Streaming Replication*, the node breaks transactions down into fragments, then replicates and certifies them while the transaction is in progress. Once certified, a fragment can no longer be aborted due to conflicting transactions. This parameter determines the number of replication units to include in a fragment. To define what these units represent, use *wsrep_trx_fragment_unit* (page 267). A value of `0` indicates that streaming replication will not be used.

```
SHOW VARIABLES LIKE 'wsrep_trx_fragment_size';

+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| wsrep_trx_fragment_size | 5     |
+-------------------------+-------+
```

wsrep_trx_fragment_unit

Defines the replication unit type to use in Streaming Replication.

Command-line Format	`--wsrep-trx-fragment-unit`
System Variable	`wsrep_trx_fragment_unit`
Variable Scope	Session
Dynamic Variable	Yes
Permitted Values	String
Default Value	`bytes`
Valid Values	`bytes`, `rows`, `statements`
Initial Version	MySQL-wsrep: 8.0.19-26.3, MariaDB: 10.4.2

In *Streaming Replication*, the node breaks transactions down into fragments, then replicates and certifies them while the transaction is in progress. Once certified, a fragment can no longer be aborted due to conflicting transactions. This

parameter determines the unit to use in determining the size of the fragment. To define the number of replication units to use in the fragment, use *wsrep_trx_fragment_size* (page 267).

Supported replication units are:

- **bytes**: Refers to the fragment size in bytes.
- **rows**: Refers to the number of rows updated in the fragment.
- **statements**: Refers to the number of SQL statements in the fragment.

```
SHOW VARIABLES LIKE 'wsrep_trx_fragment_unit';

+--------------------------+--------+
| Variable_name            | Value  |
+--------------------------+--------+
| wsrep_trx_fragment_unit | bytes  |
+--------------------------+--------+
```

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *repl.causal_read_timeout* (page 301)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

8.2 Galera Functions

Starting with version 4 of Galera Cluster, there are several Galera functions available. At this point, the Galera functions related to *Global Transaction ID* (GTID). They return a GTID or have effect on transactions related to a GTID.

Function	Arguments	Initial Version
WSREP_LAST_SEEN_GTID() (page 269)		4.0
WSREP_LAST_WRITTEN_GTID() (page 269)		4.0
WSREP_SYNC_WAIT_UPTO_GTID() (page 269)	`gtid [timeout]`	4.0

WSREP_LAST_SEEN_GTID()

Much like `LAST_INSERT_ID()` for getting the identification number of the last row inserted in MySQL, this function returns the *Global Transaction ID* of the last write transaction observed by the client.

Function	`WSREP_LAST_SEEN_GTID()`
Arguments	None
Initial Version	Version 4.0

This function returns the *Global Transaction ID* of the last write transaction observed by the client. It can be useful in combination with *WSREP_SYNC_WAIT_UPTO_GTID()* (page 269). You can use this parameter to identify the transaction upon which it should wait before unblocking the client.

Below is an example of how you might use the `WSREP_LAST_SEEN_GTID()` function to get the Global Transaction ID of the last write transaction observed:

```
SELECT WSREP_LAST_SEEN_GTID();
```

WSREP_LAST_WRITTEN_GTID()

This function returns the *Global Transaction ID* of the last write transaction made by the client.

Function	`WSREP_LAST_WRITTEN_GTID()`
Arguments	None
Initial Version	Version 4.0

This function returns the Global Transaction ID of the last write transaction made by the client. This can be useful in combination with *WSREP_SYNC_WAIT_UPTO_GTID()* (page 269). You can use this parameter to identify the transaction upon which it should wait before unblocking the client.

Below is an example of how you might use the `WSREP_LAST_SEEN_GTID()` function to get the Global Transaction ID of the last write transaction observed:

```
BEGIN;

UPDATE table_name SET id = 0
WHERE field = 'example';

COMMIT;

SELECT WSREP_LAST_WRITTEN_GTID();
```

WSREP_SYNC_WAIT_UPTO_GTID()

This function blocks the client until the node applies and commits the given transaction.

Function	WSREP_SYNC_WAIT_UPTO_GTID()
Required Arguments	Global Transaction ID
Optional Arguments	timeout
Initial Version	Version 4.0

This function blocks the client until the node applies and commits the given *Global Transaction ID*. Optional argument accepts timeout in seconds. If you do not provide a timeout, it will continue to block indefinitely. It returns the following values:

- `1`: The node applied and committed the given Global Transaction ID.
- `ER_LOCAL_WAIT_TIMEOUT` Error: The function times out before the node can apply the transaction.
- `ER_WRONG_ARGUMENTS` Error: The function is given an incorrect Global Transaction ID.

Below is an example of how you might use the `WSREP_SYNC_WAIT_UPTO_GTID()` function:

```
$transaction_gtid = SELECT WSREP_LAST_SEEN_GTID();
...
SELECT WSREP_SYNC_WAIT_UPTO_GTID($transaction_gtid);
```

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Auto-Eviction* (page 100)
- *SSL Certificates* (page 216)
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

8.3 Galera Parameters

As of version 0.8, Galera Cluster accepts parameters as semicolon-separated key value pair lists, such as `key1 = value1; key2 = value2`. In this way, you can configure an arbitrary number of Galera Cluster parameters in one call. A key consists of parameter group and parameter name: `<group>.<name>`, where `<group>` corresponds roughly to some Galera module.

All `wsrep_provider_options` settings need to be specified on a single line. In case of multiple instances of `wsrep_provider_options`, only the last one is used.

Below is a list of all of the Galera parameters. Each is also a link to further down the page where you may find more information. There are a few things to know about this table:

- For numeric values related to memory size, Galera Cluster accepts the numeric modifiers, `K`, `M`, `G`, and `T` to represent 2^{10}, 2^{20}, 2^{30} and 2^{40}, respectively.
- Galera Cluster accepts the following boolean values: `0`, `1`, `YES`, `NO`, `TRUE`, `FALSE`, `ON`, `OFF`.
- Time periods must be expressed in the ISO8601 format. See some of the examples below.
- The parameters that are noted as for debugging only are strictly for use in troubleshooting problems. You should not implement these in production environments.

Parameter	Default	Dynamic	Debug Only	Initial Version	Version Deprecated
base_dir (page 278)					
base_host (page 278)	detected network address			1.0	
base_port (page 279)	`4567`			1.0	
cert.log_conflicts (page 279)	`NO`	Yes		2.0	
cert.optimistic_pa (page 279)	`YES`	Yes		3.25	
debug (page 279)	`NO`	Yes		2.0	
evs.auto_evict (page 280)	`0`	No		3.8	
evs.causal_keepalive_period (page 280)	`0`	No		1.0	
evs.consensus_timeout (page 280)	`PT30S`	No	Yes	1.0, 2.0	

Continued on next page

Table 2 – continued from previous page

Parameter	Default	Dynamic	Debug Only	Initial Version	Version Deprecated
evs.debug_log_mask (page 280)	`0x1`	Yes		1.0	
evs.delayed_keep_period (page 281)	`PT30S`	No		3.8	
evs.delay_margin (page 281)	`PT1S`	No		3.8	
evs.evict (page 282)		No		3.8	
evs.inactive_check_period (page 282)	`PT1S`	No		1.0	
evs.inactive_timeout (page 282)	`PT15S`	No		1.0	
evs.info_log_mask (page 283)	`0`	No		1.0	
evs.install_timeout (page 283)	`PT7.5S`	Yes		1.0	
evs.join_retrans_period (page 283)	`PT1S`	Yes		1.0	
evs.keepalive_period (page 284)	`PT1S`	No		1.0	
evs.max_install_timeouts (page 284)	`1`	No		1.0	
evs.send_window (page 284)	`4`	Yes		1.0	

Continued on next page

Table 2 – continued from previous page

Parameter	Default	Dynamic	Debug Only	Initial Version	Version Deprecated
evs.stats_report_period (page 285)	PT1M	No		1.0	
evs.suspect_timeout (page 285)	PT5S	No		1.0	
evs.use_aggregate (page 285)	TRUE	No		1.0	
evs.user_send_window (page 285)	2	Yes		1.0	
evs.view_forget_timeout (page 286)	PT5M	No		1.0	
evs.version (page 286)	0	No	Yes	1.0	
gcache.dir (page 287)	working directory	No		1.0	
gcache.name (page 287)	galera.cache	No		1.0	
gcache.keep_pages_size (page 287)	0	No		1.0	
gcache.mem_size (page 287)	0	No			
gcache.page_size (page 288)	128M	No		1.0	
gcache.recover (page 288)	yes	No		3.19	

Continued on next page

Table 2 – continued from previous page

Parameter	Default	Dynamic	Debug Only	Initial Version	Version Deprecated
gcache.size (page 288)	128M	No		1.0	
gcomm.thread_prio (page 289)		No		3.0	
gcs.fc_debug (page 289)	0	No		1.0	
gcs.fc_factor (page 290)	1.0	No		1.0	
gcs.fc_limit (page 290)	16	Yes		1.0	
gcs.fc_master_slave (page 290)	NO	No		1.0	4.10
gcs.fc_single_primary (page 290)	NO	No		4.10	
gcs.max_packet_size (page 291)	64500	No		1.0	
gcs.max_throttle (page 291)	0.25	No		1.0	
gcs.recv_q_hard_limit (page 291)	LLONG_MAX	No		1.0	
gcs.recv_q_soft_limit (page 291)	0.25	No		1.0	
gcs.sync_donor (page 292)	NO	No		1.0	

Continued on next page

Table 2 – continued from previous page

Parameter	Default	Dynamic	Debug Only	Initial Version	Version Deprecated
gcs.vote_policy (page 292)	0	No		1.0	
gmcast.isolate (page 293)	0	Yes	Yes		
gmcast.listen_addr (page 293)	tcp://0.0.0.0:4567	No		1.0	
gmcast.mcast_addr (page 293)		No		1.0	
gmcast.mcast_ttl (page 294)	1	No		1.0	
gmcast.peer_timeout (page 294)	PT3S	No		1.0	
gmcast.segment (page 294)	0	No		3.0	
gmcast.time_wait (page 294)	PT5S	No		1.0	
gmcast.version (page 295)	n/a	No	Yes	1.0	
innodb_flush_log_at_trx_commit (page 295)	1	Yes			
ist.recv_addr (page 296)		No		1.0	
ist.recv_bind (page 296)		No		3.0	

Continued on next page

Table 2 – continued from previous page

Parameter	Default	Dynamic	Debug Only	Initial Version	Version Deprecated
pc.announce_timeout (page 297)	PT3S	No		2.0	
pc.bootstrap (page 297)	n/a	No		2.0	
pc.checksum (page 297)	FALSE	No		1.0	
pc.ignore_sb (page 298)	FALSE	Yes		1.0	
pc.ignore_quorum (page 298)	FALSE	Yes		1.0	
pc.linger (page 298)	PT20S	No		1.0	
pc.npvo (page 299)	FALSE	No		1.0	
pc.recovery (page 296)	TRUE	No		3.0	
pc.version (page 300)	n/a	No	Yes	1.0	
pc.wait_prim (page 299)	TRUE	No		1.0	
pc.wait_prim_timeout (page 299)	PT30S	No		2.0	
pc.weight (page 300)	1	Yes		2.4	

Continued on next page

Table 2 – continued from previous page

Parameter	Default	Dynamic	Debug Only	Initial Version	Version Deprecated
protonet.backend (page 300)	`asio`	No		1.0	4.14
protonet.version (page 301)	n/a	No	Yes	1.0	4.14
repl.causal_read_timeout (page 301)	`PT30S`	No		1.0	
repl.commit_order (page 301)	`3`	No		1.0	
repl.key_format (page 302)	`FLAT8`	No		3.0	
repl.max_ws_size (page 302)	`2147483647`	No		3.0	
repl.proto_max (page 302)	`5`	No		2.0	
socket.recv_buf_size (page 302)	`auto`	Yes		3.17	
socket.send_buf_size (page 303)	`auto`	Yes		3.29	
socket.ssl (page 303)	0	No			
socket.ssl_ca (page 303)		No		1.0	
socket.ssl_cert (page 304)		No		1.0	

Continued on next page

Table 2 – continued from previous page

Parameter	Default	Dynamic	Debug Only	Initial Version	Version Deprecated
socket.checksum (page 304)	`1` (vs. 2); `2` (vs. 3)	No		2.0	
socket.dynamic (page 304)	FALSE	No		4.8	
socket.ssl_cipher (page 305)		No		1.0	
socket.ssl_compression (page 305)	`YES`	No		1.0	4.14
socket.ssl_key (page 305)		No		1.0	
socket.ssl_password_file (page 306)		No		1.0	
socket.ssl_reload (page 306)		No		4.8	

base_dir

Specifies the data directory.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="base_dir=/var/lib/mysql/"
```

or

```
wsrep_provider_options="base_dir=/etc/default/garbd/"
```

base_host

Global variable for internal use.

Default Value	detected network address
Dynamic	
Initial Version	

Warning: Since this is for internal use only, do not manually set the `base_host` variable.

base_port

Global variable for internal use.

Default Value	4567
Dynamic	
Initial Version	

Warning: Since this is for internal use only, do not manually set the `base_port` variable.

cert.log_conflicts

Log details of certification failures.

Default Value	NO
Dynamic	Yes
Initial Version	2.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="cert.log_conflicts=NO"
```

cert.optimistic_pa

Controls parallel applying of replica actions. When enabled allows full range of parallelization as determined by certification algorithm. When disabled limits parallel applying window to not exceed that seen on primary. In other words, the action starts applying no sooner than all actions it has seen on the primary are committed.

Default Value	YES
Dynamic	Yes
Initial Version	3.25

```
wsrep_provider_options="cert.optimistic_pa=NO"
```

debug

Enable debugging.

Default Value	NO
Dynamic	Yes
Initial Version	2.0

```
wsrep_provider_options="debug=NO"
```

evs.auto_evict

Defines how many entries the node allows for given a delayed node before it triggers the Auto Eviction protocol.

Default Value	0
Dynamic	No
Initial Version	3.8

Each cluster node monitors the group communication response times from all other nodes. When the cluster registers delayed response from a given node, it adds an entry for that node to its delayed list. If the majority of the cluster nodes show the node as delayed, the node is permanently evicted from the cluster.

This parameter determines how many entries a given node can receive before it triggers Auto Eviction.

When this parameter is set to `0`, it disables the Auto Eviction protocol for this node. Even when you disable Auto Eviction, though; the node continues to monitor response times from the cluster.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="evs.auto_evict=5"
```

For more information on the Auto Eviction process, see *Auto-Eviction* (page 100).

evs.causal_keepalive_period

For developer use only. Defaults to `evs.keepalive_period`.

Default Value	
Dynamic	No
Initial Version	1.0

evs.consensus_timeout

Timeout on reaching the consensus about cluster membership.

Default Value	PT30S
Dynamic	No
Initial Version	1.0
Deprecated	2.0

This variable is mostly used for troubleshooting purposes and should not be implemented in a production environment.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="evs.consensus_timeout=PT30S"
```

Note: This feature has been **deprecated**. It is succeeded by *evs.install_timeout* (page 283).

evs.debug_log_mask

Control EVS debug logging, only effective when `wsrep_debug` is in use.

Default Value	`0x1`
Dynamic	Yes
Initial Version	1.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="evs.debug_log_mask=0x1"
```

evs.delayed_keep_period

Defines how long this node requires a delayed node to remain responsive before it removes an entry from the delayed list.

Default Value	`PT30S`
Dynamic	No
Initial Version	3.8

Each cluster node monitors the group communication response times from all other nodes. When the cluster registered delayed responses from a given node, it adds an entry for that node to its delayed list. Nodes that remain on the delayed list can trigger Auto Eviction, which removes them permanently from the cluster.

This parameter determines how long a node on the delayed list must remain responsive before it removes one entry. The number of entries on the delayed list and how long it takes before the node removes all entries depends on how long the delayed node was unresponsive.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="evs.delayed_keep_period=PT45S"
```

For more information on the delayed list and the Auto Eviction process, see *Auto-Eviction* (page 100).

evs.delay_margin

Defines how long the node allows response times to deviate before adding an entry to the delayed list.

Default Value	PT1S
Dynamic	No
Initial Version	3.8

Each cluster node monitors group communication response times from all other nodes. When the cluster registers a delayed response from a given node, it adds an entry for that node to its delayed list. Delayed nodes can trigger Auto Eviction, which removes them permanently from the cluster.

This parameter determines how long a delay can run before the node adds an entry to the delayed list. You must set this parameter to a value higher than the round-trip delay time (RTT) between the nodes.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="evs.delay_margin=PT5S"
```

For more information on the delayed list and the Auto Eviction process, see *Auto-Eviction* (page 100).

evs.evict

If set to the gcomm UUID of some node, that node will be evicted from the cluster. Setting this parameter to an empty string causes the eviction list to be cleared on the node where it is set.

Default Value	
Dynamic	No
Initial Version	3.8

For more information on the eviction and Auto Eviction process, see *Auto-Eviction* (page 100).

evs.inactive_check_period

Defines how often you want the node to check for peer inactivity.

Default Value	PT1S
Dynamic	No
Initial Version	1.0

Each cluster node monitors group communication response times from all other nodes. When the cluster registers a delayed response from a given node, it adds an entry for that node to its delayed list, which can lead to the delayed node's eviction from the cluster.

This parameter determines how often you want the node to check for delays in the group communication responses from other cluster nodes.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="evs.inactive_check_period=PT1S"
```

evs.inactive_timeout

Defines a hard limit on node inactivity.

Default Value	PT15S
Dynamic	No
Initial Version	1.0

Hard limit on the inactivity period, after which the node is pronounced dead.

Each cluster node monitors group communication response times from all other nodes. When the cluster registers a delayed response from a given node, it add an entry for that node to its delayed list, which can lead to the delayed node's eviction from the cluster.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="evs.inactive_timeout=PT15S"
```

This parameter sets a hard limit for node inactivity. If a delayed node remains unresponsive for longer than this period, the node pronounces the delayed node as dead.

evs.info_log_mask

Defines additional logging options for the EVS Protocol.

Default Value	0
Dynamic	No
Initial Version	1.0

The EVS Protocol monitors group communication response times and controls the node eviction and auto eviction processes. This parameter allows you to enable additional logging options, through a bitmask value.

- 0x1 Provides extra view change info.
- 0x2 Provides extra state change info
- 0x4 Provides statistics
- 0x8 Provides profiling (only in builds with profiling enabled)

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="evs.info_log_mask=0x4"
```

evs.install_timeout

Defines the timeout for install message acknowledgments.

Default Value	PT7.5S
Dynamic	Yes
Initial Version	1.0

Each cluster node monitors group communication response times from all other nodes, checking whether they are responsive or delayed. This parameter determines how long you want the node to wait on install message acknowledgments.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="evs.install_timeout=PT7.5S"
```

Note: This parameter replaces *evs.consensus_timeout* (page 280).

evs.join_retrans_period

Defines how often the node retransmits EVS join messages when forming cluster membership.

Default Value	PT1S
Dynamic	Yes
Initial Version	1.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="evs.join_retrans_period=PT1S"
```

evs.keepalive_period

Defines how often the node emits keepalive signals.

Default Value	PT1S
Dynamic	No
Initial Version	1.0

Each cluster node monitors group communication response times from all other nodes. When there is no traffic going out for the cluster to monitor, nodes emit keepalive signals so that other nodes have something to measure. This parameter determines how often the node emits a keepalive signal, absent any other traffic.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="evs.keepalive_period=PT1S"
```

evs.max_install_timeouts

Defines the number of membership install rounds to try before giving up.

Default Value	1
Dynamic	No
Initial Version	1.0

This parameter determines the maximum number of times that the node tries for a membership install acknowledgment, before it stops trying. The total number of rounds it tries is this value plus 2.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="evs.max_install_timeouts=1"
```

evs.send_window

Defines the maximum number of packets at a time in replication.

Default Value	4
Dynamic	Yes
Initial Version	1.0

This parameter determines the maximum number of packets the node uses at a time in replication. For clusters implemented over WAN, you can set this value considerably higher, (for example, 512), than for clusters implemented over LAN.

You must use a value that is greater than *evs.user_send_window* (page 285). The recommended value is double *evs.user_send_window* (page 285).

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="evs.send_window=4"
```

evs.stats_report_period

Control period of EVS statistics reporting. The node is pronounced dead.

Default Value	PT1M
Dynamic	No
Initial Version	1.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="evs.stats_report_period=PT1M"
```

evs.suspect_timeout

Defines the inactivity period after which a node is *suspected* as dead.

Default Value	PT5S
Dynamic	No
Initial Version	1.0

Each node in the cluster monitors group communications from all other nodes in the cluster. This parameter determines the period of inactivity before the node suspects another of being dead. If all nodes agree on that, the cluster drops the inactive node.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="evs.suspect_timeout=PT5S"
```

evs.use_aggregate

Defines whether the node aggregates small packets into one when possible.

Default Value	TRUE
Dynamic	No
Initial Version	1.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="evs.use_aggregate=TRUE"
```

evs.user_send_window

Defines the maximum number of data packets at a time in replication.

Default Value	2
Dynamic	Yes
Initial Version	1.0

This parameter determines the maximum number of data packets the node uses at a time in replication. For clusters implemented over WAN, you can set this to a value considerably higher than cluster implementations over LAN, (for example, 512).

You must use a value that is smaller than *evs.send_window* (page 284). The recommended value is half *evs.send_window* (page 284).

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="evs.user_send_window=2"
```

For more information, see *evs.send_window* (page 284).

evs.view_forget_timeout

Defines how long the node saves past views from the view history.

Default Value	PT5M
Dynamic	No
Initial Version	1.0

Each node maintains a history of past views. This parameter determines how long you want the node to save past views before dropping them from the table.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="evs.view_forget_timeout=PT5M"
```

evs.version

Defines the EVS Protocol version.

Default Value	0 (up to Galera Cluster version 3.9) | 1 (as of Galera Cluster version 4.0)
Dynamic	No
Initial Version	1.0

This parameter determines which version of the EVS Protocol the node uses. In order to ensure backwards compatibility, the parameter defaults to 0 on Galera Cluster versions prior to 3.9. Certain EVS Protocol features, such as Auto Eviction, require you to upgrade to more recent versions. As of Galera Cluster version 4.0, the parameter defaults to 1.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="evs.version=1"
```

For more information on the procedure to upgrade from one version to another, see *Upgrading the EVS Protocol* (page 101).

gcache.dir

Defines the directory where the write-set cache places its files.

Default Value	`/path/to/working_dir`
Dynamic	No
Initial Version	1.0

When nodes receive state transfers they cannot process incoming write-sets until they finish updating their state. Under certain methods, the node that sends the state transfer is similarly blocked. To prevent the database from falling further behind, GCache saves the incoming write-sets on memory mapped files to disk.

This parameter determines where you want the node to save these files for write-set caching. By default, GCache uses the working directory for the database server.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gcache.dir=/usr/share/galera"
```

gcache.keep_pages_size

Total size of the page storage pages to keep for caching purposes. If only page storage is enabled, one page is always present.

Default Value	0
Dynamic	No
Initial Version	1.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gcache.keep_pages_size=0"
```

gcache.mem_size

The maximum size of size of the `malloc()` store for setups that have spare RAM.

Default Value	0
Dynamic	No

gcache.name

Defines the filename for the write-set cache.

Default Value	`galera.cache`
Dynamic	No
Initial Version	1.0

When nodes receive state transfers they cannot process incoming write-sets until they finish updating their state. Under certain methods, the node that sends the state transfer is similarly blocked. To prevent the database from falling further behind, GCache saves the incoming write-sets on memory-mapped files to disk.

This parameter determines the name you want the node to use for this ring buffer storage file.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gcache.name=galera.cache"
```

gcache.page_size

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`.

Default Value	`128M`
Dynamic	No
Initial Version	1.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gcache.page_size=128M"
```

gcache.recover

Determines whether Gcache recovery takes place on node startup.

As of wsrep API version 26.4.1, Gcache recovery is performed by default in conjunction with entire cluster restarts. At the startup of the first node, instead of deleting the Gcache, Galera will attempt to recover the Gcache file to a usable state, preserving the node ability to serve as an IST donor to other nodes that are also starting up at the same time. Now all the remaining nodes in the cluster can join over IST, drastically reducing the total time needed to bring up the entire cluster. Furthermore, the first node never has to become an SST donor, so it is never blocked or burdened by having to perform the SST operation once for each other node in the cluster.

Gcache recovery is a best-effort operation, which will complete successfully in cases where the nodes were gracefully shut down prior to restarting. Even in the case of a power outage, a monitoring system can be used to shut down the nodes gracefully.

In certain types of hard crashes, such as a sudden power loss to the entire cluster, Galera may not be able to recover the Gcache sufficiently for it to be used to serve IST. In this case, node startup will proceed as usual and SST will happen when the second and subsequent nodes join the cluster.

Default Value	`yes`
Dynamic	No
Initial Version	3.19

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gcache.recover=yes"
```

gcache.size

Defines the disk space you want to node to use in caching write-sets.

Default Value	128M
Dynamic	No
Initial Version	1.0

When nodes receive state transfers they cannot process incoming write-sets until they finish updating their state. Under certain methods, the node that sends the state transfer is similarly blocked. To prevent the database from falling further behind, GCache saves the incoming write-sets on memory-mapped files to disk.

This parameter defines the amount of disk space you want to allocate for the present ring buffer storage. The node allocates this space when it starts the database server.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gcache.size=128M"
```

For more information on customizing the write-set cache, see the Best Practice Articles.

gcomm.thread_prio

Defines the policy and priority for the gcomm thread.

Default Value	
Dynamic	No
Initial Version	3.0

Using this option, you can raise the priority of the gcomm thread to a higher level than it normally uses. You may find this useful in situations where Galera Cluster threads do not receive sufficient CPU time, due to competition with other MySQL threads. In these cases, when the thread scheduler for the operating system does not run the Galera threads frequently enough, timeouts may occur, causing the node to drop from the cluster.

The format for this option is: `<policy>:<priority>`. The priority value is an integer. The policy value supports the following options:

- `other` Designates the default time-sharing scheduling in Linux. They can run until they are blocked by an I/O request or preempted by higher priorities or superior scheduling designations.
- `fifo` Designates first-in out 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` Designates 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.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gcomm.thread_prio=rr:2"
```

gcs.fc_debug

Post debug statistics about replication flow every this number of writesets.

Default Value	0
Dynamic	No
Initial Version	1.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gcs.fc_debug=0"
```

gcs.fc_factor

Resume replication after recv queue drops below this fraction of `gcs.fc_limit`.

Default Value	0.5
Dynamic	Yes
Initial Version	1.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gcs.fc_factor=0.5"
```

gcs.fc_limit

Pause replication if recv queue exceeds this number of writesets. For primary-replica setups this number can be increased considerably.

Default Value	16
Dynamic	Yes
Initial Version	1.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gcs.fc_limit=16"
```

gcs.fc_master_slave

Deprecated as of Galera 4.10 in favor of `gcs.fc_single_primary`.

gcs.fc_single_primary

Defines whether there is more than one source of replication.

As the number of nodes in the cluster grows, the larger the calculated `gcs.fc_limit` gets. At the same time, the number of writes from the nodes increases.

When this parameter value is set to NO (multi-primary), the `gcs.fc_limit` parameter is dynamically modified to give more margin for each node to be a bit further behind applying writes.

The `gcs.fc_limit` parameter is modified by the square root of the cluster size, that is, in a four-node cluster it is two times higher than the base value. This is done to compensate for the increasing replication rate noise.

Default Value	`NO`
Dynamic	No
Initial Version	1.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gcs.fc_single_primary=NO"
```

gcs.max_packet_size

All writesets exceeding that size will be fragmented.

Default Value	`64500`
Dynamic	No
Initial Version	1.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gcs.max_packet_size=64500"
```

gcs.max_throttle

How much to throttle replication rate during state transfer (to avoid running out of memory). Set the value to 0.0 if stopping replication is acceptable for completing state transfer.

Default Value	`0.25`
Dynamic	No
Initial Version	1.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gcs.max_throttle=0.25"
```

gcs.recv_q_hard_limit

Maximum allowed size of recv queue. This should normally be half of (RAM + swap). If this limit is exceeded, Galera Cluster will abort the server.

Default Value	`LLONG_MAX`
Dynamic	No
Initial Version	1.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gcs.recv_q_hard_limit=LLONG_MAX"
```

gcs.recv_q_soft_limit

The fraction of *gcs.recv_q_hard_limit* (page 291) after which replication rate will be throttled.

Default Value	0.25
Dynamic	No
Initial Version	1.0

The degree of throttling is a linear function of recv queue size and goes from 1.0 (`full rate`) at *gcs.recv_q_soft_limit* (page 291) to *gcs.max_throttle* (page 291) at *gcs.recv_q_hard_limit* (page 291) Note that `full rate`, as estimated between 0 and *gcs.recv_q_soft_limit* (page 291) is a very imprecise estimate of a regular replication rate.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gcs.recv_q_soft_limit=0.25"
```

gcs.sync_donor

Should the rest of the cluster keep in sync with the donor? `YES` means that if the donor is blocked by state transfer, the whole cluster is blocked with it.

Default Value	NO
Dynamic	No
Initial Version	1.0

If you choose to use value `YES`, it is theoretically possible that the *Donor Node* cannot keep up with the rest of the cluster due to the extra load from the SST. If the node lags behind, it may send flow control messages stalling the whole cluster. However, you can monitor this using the *wsrep_flow_control_paused* (page 318) status variable.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gcs.sync_donor=NO"
```

gcs.vote_policy

When a cluster node fails to apply a writeset, it initiates voting on the outcome. Every node casts a vote, that is, a hash of the error message or 0, if there was no error. If a node votes "wrong", the node is considered to be inconsistent and it shuts down. `gcs.vote_policy` decides on how the votes are being counted and how to choose the winner:

- `gcs.vote_policy = 0` - The outcome that has more votes is chosen as the winner. In other words, simple majority wins. In the case of a tie between 0 (success) and non-0 (error) outcomes, 0 (success) is preferred.
- `gcs.vote_policy > 0` - If success gets as many as or more votes that the parameter value defines, it is chosen as the winner, even if in minority. For example, if `gcs.vote_policy=1`, only the node that successfully committed a transaction would remain primary. Note that if `gcs.vote_policy=1`, an inconsistent primary may crash all the secondaries.

Default Value	0
Dynamic	No
Initial Version	1.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gcs.vote_policy=0"
```

gmcast.isolate

Warning: This parameter is meant for testing use only, and never meant for production use cases.

Defines how cluster connections are handled.

Default Value	0
Dynamic	Yes
Initial Version	1.0

The options are:

- `0` - Cluster connections are handled as usual.
- `1` - The node closes all cluster connections, does not open new cluster connections and rejects all incoming cluster connections.
- `2` - The node closes all cluster connections and terminates the group communication, moving the node into disconnected state.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gmcast.isolate=1"
```

gmcast.listen_addr

Address at which *Galera Cluster* listens to connections from other nodes. By default the port to listen at is taken from the connection address. This setting can be used to overwrite that.

Default Value	`tcp://0.0.0.0:4567`
Dynamic	No
Initial Version	1.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gmcast.listen_addr=tcp://0.0.0.0:4567"
```

gmcast.mcast_addr

If set, UDP multicast will be used for replication, for example:

Default Value	
Dynamic	No
Initial Version	1.0

The value must be the same on all nodes.

If you are planning to build a large cluster, we recommend using UDP.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gmcast.mcast_addr=239.192.0.11"
```

gmcast.mcast_ttl

Time to live value for multicast packets.

Default Value	1
Dynamic	No
Initial Version	1.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gmcast.mcast_ttl=1"
```

gmcast.peer_timeout

Connection timeout for inactive connections.

Default Value	PT3S
Dynamic	No
Initial Version	1.0

GMCast module monitors liveness of the socket connections on a regular basis. Nodes exchange periodically replication and keepalives messages, which are expected to be received more frequently than `gmcast.peer_timeout` duration. The `gmcast.peer_timeout` defines the timeout after which an idle socket connection between two nodes is considered inactive and will be closed and reopened again. If a socket connection is closed due to timeout, the relaying protocol is activated as a side effect to keep messages delivered to all members in the cluster.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gmcast.peer_timeout=PT3S"
```

gmcast.segment

Define which network segment this node is in. Optimisations on communication are performed to minimise the amount of traffic between network segments including writeset relaying and IST and SST donor selection. The *gmcast.segment* (page 294) value is an integer from `0` to `255`. By default all nodes are placed in the same segment (`0`).

Default Value	0
Dynamic	No
Initial Version	3.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gmcast.segment=0"
```

gmcast.time_wait

Time to wait until allowing peer declared outside of stable view to reconnect.

Default Value	PT5S
Dynamic	No
Initial Version	1.0

In case a node leaves or gets partitioned from the cluster, it is kept isolated from the rest of the cluster for a while to avoid distractions to membership protocol operation. Option `gmcast.time_wait` denotes the time period during which nodes in the primary component refuse connection attempts from nodes not in the primary component in the following way: The nodes which partitioned from the cluster are allowed to reconnect after `gmcast.time_wait/2` seconds, the nodes which left the group completely are allowed to reconnect after `gmcast.time_wait` seconds.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="gmcast.time_wait=PT5S"
```

gmcast.version

This status variable is used to check which gmcast protocol version is used.

Default Value	
Dynamic	No
Initial Version	1.0

This variable is mostly used for troubleshooting purposes and should not be implemented in a production environment.

innodb_flush_log_at_trx_commit

This variable controls the durability/speed trade-off for commits.

The possible values are:

- `0` - Nothing is done on commit; rather the log buffer is written and flushed to the InnoDB redo log once a second. This gives better performance, but a server crash can erase the last second of transactions.
- `1` - The log buffer is written to the InnoDB redo log file, and a flush to disk performed after each transaction. This is required for full ACID compliance.
- `2` - The log buffer is written to the InnoDB redo log after each commit, but flushing takes place every ‘‘innodb_flush_log_at_timeout‘‘(by default once a second). The performance is better, but an operating system crash or a power outage can cause the last second’s transactions to be lost.
- `3` - Flush to disk at prepare and at commit. This option is slower and usually redundant.

Options 0 and 2 can be faster than 1 or 3.

Default Value	1
Dynamic	Yes
Initial Version	1.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
innodb_flush_log_at_trx_commit="1"
```

This variable can also be set dynamically at runtime:

```
SET GLOBAL innodb_flush_log_at_trx_commit=1;
```

If you set `innodb_flush_log_at_trx_commit` dynamically at runtime, its value will be reset the next time the server restarts. To make the value persist on restart, set it also in a configuration file.

Note: If you use MySQL 8 or a later version, you can also use `SET PERSIST` to ensure the value persists upon restart.

---

ist.recv_addr

Address to listen on for Incremental State Transfer. By default this is the `<address>:<port+1>` from *wsrep_node_address* (page 248).

Default Value	
Dynamic	No
Initial Version	2.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="ist.recv_addr=192.168.1.1"
```

ist.recv_bind

Defines the address that the node binds on for receiving an *Incremental State Transfer*.

Default Value	
Dynamic	No
Initial Version	3.16

This option defines the address to which the node will bind in order to receive Incremental State Transfers. When this option is not set, it takes its value from *ist.recv_addr* (page 296) or, in the event that that is also not set, from *wsrep_node_address* (page 248). You may find it useful when the node runs behind a NAT or in similar cases where the public and private addresses differ.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="ist.recv_bind=192.168.1.1"
```

pc.recovery

When set to `TRUE`, the node stores the Primary Component state to disk, in the `gvwstate.dat` file. The Primary Component can then recover automatically when all nodes that were part of the last saved state reestablish communications with each other.

Default Value	TRUE
Dynamic	No
Initial Version	3.0

This allows for:

- Automatic recovery from full cluster crashes, such as in the case of a data center power outage.
- Graceful full cluster restarts without the need for explicitly bootstrapping a new Primary Component.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="pc.recovery=TRUE"
```

Note: In the event that the wsrep position differs between nodes, recovery also requires a full State Snapshot Transfer.

pc.bootstrap

If you set this value to TRUE is a signal to turn a NON-PRIMARY component into PRIMARY.

Default Value	
Dynamic	Yes
Initial Version	2.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="pc.bootstrap=TRUE"
```

pc.announce_timeout

Cluster joining announcements are sent every $\frac{1}{2}$ second for this period of time or less if the other nodes are discovered.

Default Value	PT3S
Dynamic	No
Initial Version	2.0

When a node joins the cluster, it will initially broadcast join messages every half a second to inform other nodes about the joining attempt, but does not handle its own messages to avoid forming membership configuration containing only itself. This initial join message broadcasting will terminate if at least one other node is seen or if the timeout expires, whichever happens first.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="pc.announce_timeout=PT3S"
```

pc.checksum

Checksum replicated messages.

Default Value	FALSE
Dynamic	No
Initial Version	1.0

If set to true, a CRC16 checksum is computed and included into replicated messages on primary component protocol level. This checksum is redundant since the checksum does not take into account all protocol layers, and all messages are checksummed with stronger CRC32C algorithm when transferred between nodes.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="pc.checksum=TRUE"
```

pc.ignore_sb

Should we allow nodes to process updates even in the case of *Split Brain*? This is a dangerous setting in a multi-primary setup, but should simplify things in a primary-replica cluster (especially if only 2 nodes are used).

Default Value	FALSE
Dynamic	Yes
Initial Version	1.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="pc.ignore_sb=FALSE"
```

pc.ignore_quorum

Completely ignore *Quorum* calculations. For example if the primary splits from several replicas it still remains operational. Use with extreme caution even in primary-replica setups, as replicas will not automatically reconnect to primary in this case.

Default Value	FALSE
Dynamic	Yes
Initial Version	1.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="pc.ignore_quorum=FALSE"
```

pc.linger

The period for which the PC protocol waits for the EVS termination.

Default Value	PT20S
Dynamic	No
Initial Version	1.0

When a node leaves the cluster, it will wait up to `pc.linger` duration to get itself removed gracefully from the cluster.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="pc.linger=PT20S"
```

pc.npvo

Control which primary component is allowed to continue in case of conflicting primary components after cluster partitioning.

Default Value	FALSE
Dynamic	No
Initial Version	1.0

If the cluster is configured to ignore quorum or split brain, the nodes may continue processing write sets independently after cluster partitioning, which leads to diverged states. When the cluster merges back to the original configuration, the pc.npvo controls which partition is allowed to continue after the merge. If the value is FALSE, the partition with the lowest view identifier (the most immediate successor to the view before partitioning) is allowed to continue. Otherwise the partition with the highest view identifier (the one which has gone through more configuration changes or has a representative with highest node identifier) is allowed to continue.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="pc.npvo=FALSE"
```

pc.wait_prim

Control whether a joining node is allowed to start in non-primary component or if it should wait for primary component.

Default Value	TRUE
Dynamic	No
Initial Version	1.0

This variable can be used to control if a node waits for connecting to the primary component when joining to the cluster.

With default value TRUE, the joining node waits for the primary component so that the first event delivered by the group communication system is the primary component view event, or times out with error (see *pc.wait_prim_timeout* (page 299)).

With value FALSE, the joining node completes the initialization without waiting for the primary component, and may end up in non-primary component if no other nodes are seen in *pc.announce_timeout* (page 297). This is a useful setting mainly if it is desirable to start the cluster by starting all the nodes at once, and bootstrapping the primary component with *pc.bootstrap* (page 297) after all nodes have been started.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="pc.wait_prim=FALSE"
```

pc.wait_prim_timeout

The timeout for waiting primary component.

Default Value	PT30S
Dynamic	No
Initial Version	2.0

The timeout after which an error is thrown if a primary view event is not seen when joining a new node into the cluster. This value is effective only if *pc.wait_prim* (page 299) is set to TRUE.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="pc.wait_prim_timeout=PT30S"
```

pc.weight

Node weight for quorum calculation.

Default Value	1
Dynamic	Yes
Initial Version	2.4

For detailed description about weighted quorum see https://galeracluster.com/library/documentation/weighted-quorum.html

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="pc.weight=1"
```

pc.version

This variable is used to control which PC protocol version is used.

Default Value	
Dynamic	No
Initial Version	1.0

This variable is mostly used for troubleshooting purposes and should not be implemented in a production environment.

protonet.backend

This parameter is deprecated and will be removed in the future versions.

Which transport backend to use. Currently only ASIO is supported. This variable is deprecated and will be removed in the future versions.

Default Value	asio
Dynamic	No
Initial Version	1.0
Version Deprecated	4.14

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="protonet.backend=asio"
```

protonet.version

This parameter is deprecated and will be removed in the future versions

This status variable is used to check which transport backend protocol version is used.

Default Value	
Dynamic	No
Initial Version	1.0
Version Deprecated	4.14

This variable is mostly used for troubleshooting purposes and should not be implemented in a production environment.

repl.commit_order

Warning: Do not change the default value `3`, as other values may produce inconsistencies or even lead to server crashes.

Whether to allow Out-Of-Order committing (improves parallel applying performance).

Default Value	`3`
Dynamic	No
Initial Version	1.0

Possible settings:

- `0` or `BYPASS` All commit order monitoring is switched off (useful for measuring performance penalty).
- `1` or `OOOC` Allows out of order committing for all transactions.
- `2` or `LOCAL_OOOC` Allows out of order committing only for local transactions.
- `3` or `NO_OOOC` No out of order committing is allowed (strict total order committing)

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="repl.commit_order=2"
```

repl.causal_read_timeout

The default timeout for causal read and sync wait operations.

Default Value	`PT30S`
Dynamic	No
Initial Version	1.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="repl.causal_read_timeout=PT30S"
```

repl.key_format

The hash size to use for key formats (in bytes). An `A` suffix annotates the version.

Default Value	`FLAT8`
Dynamic	No
Initial Version	3.0

Possible settings:

- `FLAT8`
- `FLAT8A`
- `FLAT16`
- `FLAT16A`

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="repl.key_format=FLAT8"
```

repl.max_ws_size

The maximum size of a write-set in bytes. This is limited to 2G.

Default Value	`2147483647`
Dynamic	No
Initial Version	3.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="repl.max_ws_size=2147483647"
```

repl.proto_max

The maximum protocol version in replication. Changes to this parameter will only take effect after a provider restart.

Default Value	`5`
Dynamic	No
Initial Version	2.0

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="repl.proto_max=5"
```

socket.recv_buf_size

The size of the receive buffer that used on the network sockets between nodes. Galera passes the value to the kernel via the `SO_RCVBUF` socket option. The value is either numeric value in bytes or `auto` which allows the kernel to autotune the receive buffer. The default was changed from `212992` to `auto` in 3.29.

Default Value	`auto`
Dynamic	No
Initial Version	3.17

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="socket.recv_buf_size=212992"
```

socket.send_buf_size

The size of the send buffer that used on the network sockets between nodes. Galera passes the value to the kernel via the `SO_SNDBUF` socket option. The value is either numeric value in bytes or `auto` which allows the kernel to autotune the send buffer.

Default Value	`auto`
Dynamic	No
Initial Version	3.29

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="socket.send_buf_size=212992"
```

socket.ssl

Explicitly enables TLS usage by the wsrep provider.

Default Value	`No`
Dynamic	No

socket.ssl_ca

Defines the path to the SSL Certificate Authority (CA) file.

Default Value	
Dynamic	No
Initial Version	1.0

The node uses the CA file to verify the signature on the certificate. You can use either an absolute path or one relative to the working directory. The file must use PEM format.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options='socket.ssl_ca=/path/to/ca-cert.pem'
```

For more information on generating SSL certificate files for your cluster, see *SSL Certificates* (page 216).

socket.ssl_cert

Defines the path to the SSL certificate.

Default Value	
Dynamic	No
Initial Version	1.0

The node uses the certificate as a self-signed public key in encrypting replication traffic over SSL. You can use either an absolute path or one relative to the working directory. The file must use PEM format.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="socket.ssl_cert=/path/to/server-cert.pem"
```

For more information on generating SSL certificate files for your cluster, see *SSL Certificates* (page 216).

socket.checksum

Checksum to use on socket layer.

Default Value	`1` (before vs. 3), `2`
Dynamic	No
Initial Version	2.0

The possible values are:

- `0` - Disable checksum
- `1` - CRC32
- `2` - CRC-32C (optimized and potentially HW-accelerated on Intel CPUs)

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="socket.checksum=2"
```

socket.dynamic

Enable connection engine to accept both SSL and TCP connections.

Default Value	false
Dynamic	No
Initial Version	4.8

By enabling this parameter, it should be possible for Galera to communicate with both SSL and TCP connections. If SSL is enabled it will try to establish/accept SSL connection first and than fallback to TCP connection if necessary.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="socket.dynamic=true"
```

socket.ssl_cipher

Symmetric cipher to use for encrypted connections.

Default Value	
Dynamic	No
Initial Version	1.0

This parameter defines which cipher to use for encrypted SSL connections. If left empty, the SSL library implementation default cipher is used.

The value format depends on used SSL implementation. For OpenSSL, see cipher list format description in https://www.openssl.org/docs/manmaster/man1/openssl-ciphers.html.

The default value was `AES128-SHA` until Galera version 3.24.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="socket.ssl_cipher=AES128-SHA256"
```

socket.ssl_compression

This parameter is deprecated and will be removed in the future versions.

Whether to enable compression on SSL connections.

Default Value	`YES`
Dynamic	No
Initial Version	1.0
Version Deprecated	4.14

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="socket.ssl_compression=YES"
```

socket.ssl_key

Defines the path to the SSL certificate key.

Default Value	
Dynamic	No
Initial Version	1.0

The node uses the certificate key a self-signed private key in encrypting replication traffic over SSL. You can use either an absolute path or one relative to the working directory. The file must use PEM format.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="socket.ssl_key=/path/to/server-key.pem"
```

For more information on generating SSL certificate files for your cluster, see *SSL Certificates* (page 216).

socket.ssl_password_file

Defines a password file for use in SSL connections.

Default Value	
Dynamic	No
Initial Version	1.0

In the event that you have your SSL key file encrypted, the node uses the SSL password file to decrypt the key file.

The excerpt below is an example of how this Galera parameter might look in the configuration file:

```
wsrep_provider_options="socket.ssl_password_file=/path/to/password-file"
```

socket.ssl_reload

Reinitialize SSL context.

Default Value	
Dynamic	Yes
Initial Version	4.8

Parameter used to dynamically reinitialize the Galera SSL context. This is most useful if you need to replace a certificate that is about to expire without restarting the server. You need to change the certificate and key files at the relevant paths defined by SSL variables.

The excerpt below is an example of how this Galera parameter can be triggered from running database:

```
SET GLOBAL wsrep_provider_options = 'socket.ssl_reload=1';
```

8.3.1 Setting Galera Parameters in MySQL

You can set *Galera Cluster* parameters in the `my.cnf` configuration file as follows:

```
wsrep_provider_options="gcs.fc_limit=256;gcs.fc_factor=0.9"
```

This is useful in primary-replica setups.

You can set Galera Cluster parameters through a MySQL client with the following query:

```
SET GLOBAL wsrep_provider_options="evs.send_window=16";
```

This query only changes the *evs.send_window* (page 284) value.

To check which parameters are used in Galera Cluster, enter the following query:

```
SHOW VARIABLES LIKE 'wsrep_provider_options';
```

Related Documents

- *Auto-Eviction* (page 100)
- *SSL Certificates* (page 216)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

8.4 Galera Status Variables

These variables are *Galera Cluster* 0.8.x status variables. There are two types of wsrep-related status variables:

- Galera Cluster-specific variables exported by Galera Cluster
- Variables exported by MySQL. These variables are for the general wsrep provider.

This distinction is of importance for developers only. For convenience, all status variables are presented as a single list below. They're noted as to whether they are exported by Galera or by MySQL.

Status Variable	Exporter	Example Value	Initial Version
wsrep_applier_thread_count (page 309)	Galera	5	MariaDB 10.2.26
wsrep_apply_oooe (page 309)	Galera	0.671120	1.0
wsrep_apply_oool (page 309)	Galera	0.195248	1.0
wsrep_apply_waits (page 310)	Galera	13549	3.34,4.9
wsrep_apply_window (page 310)	Galera	5.163966	1.0
wsrep_cert_deps_distance (page 311)	Galera	23.88889	1.0
wsrep_cert_index_size (page 311)	Galera	30936	1.0
wsrep_cert_interval (page 311)	Galera		1.0
wsrep_cluster_capabilities (page 312)	Galera		1.0
wsrep_cluster_conf_id (page 312)	MySQL	34	1.0

Continued on next page

Table 3 – continued from previous page

Status Variable	Exporter	Example Value	Initial Version
wsrep_cluster_size (page 313)	MySQL		1.0
wsrep_cluster_state_uuid (page 313)	MySQL		1.0
wsrep_cluster_status (page 313)	MySQL	`Primary`	1.0
wsrep_cluster_weight (page 314)	MySQL	`3`	3.24
wsrep_commit_oooe (page 314)	Galera	`0.000000`	1.0
wsrep_commit_oool (page 315)	Galera	`0.000000`	1.0
wsrep_commit_window (page 315)	Galera	`0.000000`	1.0
wsrep_connected (page 315)	Galera	`ON`	1.0
wsrep_desync_count (page 316)	Galera	`0`	3.0
wsrep_evs_delayed (page 316)	Galera		3.8
wsrep_evs_evict_list (page 316)	Galera		3.0
wsrep_evs_repl_latency (page 317)	Galera		3.0
wsrep_evs_state (page 317)	Galera		3.8
wsrep_flow_control_active (page 318)	Galera	`false`	3.31
wsrep_flow_control_paused (page 318)	Galera	`0.184353`	1.0
wsrep_flow_control_paused_ns (page 318)	Galera	`20222491180`	1.0
wsrep_flow_control_recv (page 319)	Galera	`11`	1.0
wsrep_flow_control_requested (page 319)	Galera	`false`	3.31
wsrep_flow_control_sent (page 319)	Galera	`7`	1.0
wsrep_gcomm_uuid (page 320)	Galera		1.0
wsrep_gmcast_seqment (page 320)	Galera	2	3.31
wsrep_incoming_addresses (page 321)	Galera		1.0
wsrep_ist_receive_status (page 321)	Galera		1.0
wsrep_last_committed (page 321)	Galera	`409745`	1.0
wsrep_local_bf_aborts (page 321)	Galera	`960`	1.0
wsrep_local_cached_downto (page 322)	Galera		1.0
wsrep_local_cert_failures (page 322)	Galera	`333`	1.0
wsrep_local_commits (page 323)	Galera	`14981`	1.0
wsrep_local_index (page 323)	Galera	`1`	1.0
wsrep_local_recv_queue (page 323)	Galera	`0`	1.0
wsrep_local_recv_queue_avg (page 324)	Galera	`3.348452`	1.0
wsrep_local_recv_queue_max (page 324)	Galera	`10`	1.0
wsrep_local_recv_queue_min (page 324)	Galera	`0`	1.0
wsrep_local_replays (page 325)	Galera	`0`	1.0
wsrep_local_send_queue (page 325)	Galera	`1`	1.0
wsrep_local_send_queue_avg (page 325)	Galera	`0.145000`	1.0
wsrep_local_send_queue_max (page 326)	Galera	`10`	1.0
wsrep_local_send_queue_min (page 326)	Galera	`0`	1.0
wsrep_local_state (page 327)	Galera	`4`	1.0
wsrep_local_state_comment (page 327)	Galera	`Synced`	1.0
wsrep_local_state_uuid (page 327)	Galera		1.0
wsrep_open_connections (page 328)	Galera	`3`	3.24
wsrep_open_transactions (page 328)	Galera	`25`	3.24
wsrep_protocol_version (page 328)	Galera	`4`	1.0
wsrep_provider_capabilities (page 329)	Galera		1.0
wsrep_provider_name (page 329)	MySQL	`Galera`	1.0
wsrep_provider_vendor (page 330)	MySQL		1.0
wsrep_provider_version (page 330)	MySQL		1.0
wsrep_ready (page 331)	MySQL	`ON`	1.0

Continued on next page

Table 3 – continued from previous page

Status Variable	Exporter	Example Value	Initial Version
wsrep_received (page 331)	Galera	17831	1.0
wsrep_received_bytes (page 331)	Galera	6637093	1.0
wsrep_repl_data_bytes (page 332)	Galera	265035226	1.0
wsrep_repl_keys (page 332)	Galera	797399	1.0
wsrep_repl_keys_bytes (page 332)	Galera	11203721	1.0
wsrep_repl_other_bytes (page 333)	Galera	0	1.0
wsrep_replicated (page 333)	Galera	16109	1.0
wsrep_replicated_bytes (page 333)	Galera	6526788	1.0
wsrep_rollbacker_thread_count (page 334)	Galera	1	MariaDB 10.2.26
wsrep_thread_count (page 334)	Galera	1	1.0

wsrep_applier_thread_count

In MariaDB, stores the current number of applier threads, to indicate how many applier threads of this type there are.

Example Value	5
Location	Galera
Initial Version	MariaDB 10.2.26

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_applier_thread_count';

+------------------------------+-----------+
| Variable_name                | Value     |
+------------------------------+-----------+
| wsrep_applier_thread_count   | 5         |
+------------------------------+-----------+
```

wsrep_apply_oooe

How often applier started write-set applying out-of-order (parallelization efficiency).

Example Value	0.671120
Location	Galera
Initial Version	1.0

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_apply_oooe';

+------------------+----------+
| Variable_name    | Value    |
+------------------+----------+
| wsrep_apply_oooe | 0.671120 |
+------------------+----------+
```

wsrep_apply_oool

How often writesets with a higher sequence number were applied before ones with a lower sequence number. Values closer to 0 refer to a greater gap between slow and fast write-sets.

Example Value	0.195248
Location	Galera
Initial Version	1.0

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_apply_oool';

+------------------+----------+
| Variable_name    | Value    |
+------------------+----------+
| wsrep_apply_oool | 0.195248 |
+------------------+----------+
```

wsrep_apply_waits

Number of times an applier thread has waited for the applying order.

Example Value	13549
Location	Galera
Initial Version	3.34,4.9

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_apply_waits';

+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| wsrep_apply_waits | 13549 |
+-------------------+-------+
```

wsrep_apply_window

Average distance between highest and lowest concurrently applied seqno.

Example Value	5.163966
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_apply_window';

+--------------------+----------+
| Variable_name      | Value    |
```

(continues on next page)

(continued from previous page)

```
+--------------------+----------+
| wsrep_apply_window | 5.163966 |
+--------------------+----------+
```

wsrep_cert_deps_distance

Average distance between the highest and the lowest seqno value that can possibly be applied in parallel (potential degree of parallelization). Note that this is an average measure. You will not see acute changes in this variable.

Example Value	23.888889
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_cert_deps_distance';

+--------------------------+----------+
| Variable_name            | Value    |
+--------------------------+----------+
| wsrep_cert_deps_distance | 23.88889 |
+--------------------------+----------+
```

wsrep_cert_index_size

The number of entries in the certification index.

Example Value	30936
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_cert_index_size';

+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| wsrep_cert_index_size | 30936 |
+------------------------+-------+
```

wsrep_cert_interval

Average number of transactions received while a transaction replicates.

Example Value	1.0
Location	Galera
Initial Version	???

When a node replicates a write-set to the cluster, it can take some time before all the nodes in the cluster receive it. By the time a given node receives, orders and commits a write-set, it may receive and potentially commit others, changing the state of the database from when the write-set was sent and rendering the transaction inapplicable.

To prevent this, Galera Cluster checks write-sets against all write-sets within its certification interval for potential conflicts. Using the *wsrep_cert_interval* (page 311) status variable, you can see the average number of transactions with the certification interval.

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_cert_interval';

+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| wsrep_cert_interval | 1.0   |
+----------------------+-------+
```

This shows you the number of write-sets concurrently replicating to the cluster. In a fully synchronous cluster, with one write-set replicating at a time, *wsrep_cert_interval* (page 311) returns a value of `1.0`.

wsrep_cluster_capabilities

This parameter indicates, which features are supported by the cluster as a whole. `wsrep_cluster_capabilities` is a subset of *wsrep_provider_capabilities* (page 329). Usually they list the same set of features, but during upgrades the feature set may differ.

Example Value	**MULTI_MASTER:CERTIFICATION:PARALLEL_APPLYING**
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_cluster_capabilities';

+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| wsrep_cluster_capabilities |       |
+----------------------------+-------+
```

wsrep_cluster_conf_id

Total number of cluster membership changes happened.

Example Value	34
Location	MySQL
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_cluster_conf_id';

+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| wsrep_cluster_conf_id | 34    |
+-----------------------+-------+
```

wsrep_cluster_size

Current number of members in the cluster.

Example Value	3
Location	MySQL
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_cluster_size';

+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| wsrep_cluster_size | 15    |
+--------------------+-------+
```

wsrep_cluster_state_uuid

Provides the current State UUID. This is a unique identifier for the current state of the cluster and the sequence of changes it undergoes. If it matches the value in *wsrep_local_state_uuid* (page 327), the local and cluster nodes are in sync.

Example Value	e2c9a15e-5485-11e00900-6bbb637e7211
Location	MySQL
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_cluster_state_uuid';

+--------------------------+--------------------------------------+
| Variable_name            | Value                                |
+--------------------------+--------------------------------------+
| wsrep_cluster_state_uuid | e2c9a15e-5485-11e0-0800-6bbb637e7211 |
+--------------------------+--------------------------------------+
```

For more information on the state UUID, see *wsrep API* (page 16).

wsrep_cluster_status

Status of this cluster component. Possible values are:

- `PRIMARY` - The node is part of a `PRIMARY` component. Quorum present.
- `NON_PRIMARY` - The node is part of a `NON_PRIMARY` component. Quorum lost.
- `DISCONNECTED` - The node is not connected to group, but retrying to connect.

Example Value	`Primary`
Location	MySQL
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_cluster_status';

+----------------------+---------+
| Variable_name        | Value   |
+----------------------+---------+
| wsrep_cluster_status | Primary |
+----------------------+---------+
```

wsrep_cluster_weight

The total weight of the current members in the cluster. The value is counted as a sum of of *pc.weight* (page 300) of the nodes in the current *Primary Component*.

Example Value	3
Location	Galera
Initial Version	3.24

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_cluster_weight';

+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| wsrep_cluster_weight | 3     |
+----------------------+-------+
```

wsrep_commit_oooe

How often a transaction was committed out of order.

Example Value	`0.000000`
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_commit_oooe';

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

(continues on next page)

(continued from previous page)

```
| Variable_name     | Value    |
+-------------------+----------+
| wsrep_commit_oooe | 0.000000 |
+-------------------+----------+
```

wsrep_commit_oool

No meaning.

Example Value	0.000000
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_commit_oool';

+-------------------+----------+
| Variable_name     | Value    |
+-------------------+----------+
| wsrep_commit_oool | 0.000000 |
+-------------------+----------+
```

wsrep_commit_window

Average distance between highest and lowest concurrently committed seqno.

Example Value	0.000000
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_commit_window';

+---------------------+----------+
| Variable_name       | Value    |
+---------------------+----------+
| wsrep_commit_window | 0.000000 |
+---------------------+----------+
```

wsrep_connected

If the value is OFF, the node has not yet connected to any of the cluster components. This may be due to misconfiguration. Check the error log for proper diagnostics.

Example Value	ON
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_connected';

+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| wsrep_connected | ON    |
+-----------------+-------+
```

wsrep_desync_count

Returns the number of operations in progress that require the node to temporarily desync from the cluster.

Example Value	0
Location	Galera
Initial Version	3.8

Certain operations, such as DDL statements issued when *wsrep_OSU_method* (page 253) is set to Rolling Schema Upgrade or when you enable *wsrep_desync* (page 241), cause the node to desync from the cluster. This status variable shows how many of these operations are currently running on the node. When all of these operations complete, the counter returns to its default value `0` and the node can sync back to the cluster.

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_desync_count';

+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| wsrep_desync_count | 1     |
+--------------------+-------+
```

wsrep_evs_delayed

Provides a comma separated list of all the nodes this node has registered on its delayed list.

Example Value	
Location	Galera
Initial Version	3.8

The node listing format is as follows:

```
uuid:address:count
```

This refers to the UUID and IP address of the delayed node, with a count of the number of entries it has on the delayed list.

wsrep_evs_evict_list

Lists the UUID's of all nodes evicted from the cluster. Evicted nodes cannot rejoin the cluster until you restart their `mysqld` processes.

Example Value	
Location	Galera
Initial Version	3.8

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_evs_evict_list';

+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| wsrep_evs_evict_list |       |
+----------------------+-------+
```

wsrep_evs_repl_latency

This status variable provides figures for the replication latency on group communication. It measures latency in seconds from the time point when a message is sent out to the time point when a message is received. As replication is a group operation, this essentially gives you the slowest ACK and longest RTT in the cluster.

The format is `min/avg/max/stddev`.

Example Value	`0.00243433/0.144033/0.581963/0.215724/13`
Location	Galera
Initial Version	3.0

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_evs_repl_latency';

+------------------------+-----------------------------------------+
| Variable_name          | Value                                   |
+------------------------+-----------------------------------------+
| wsrep_evs_repl_latency | 0.00243433/0.144022/0.591963/0.215824/13 |
+------------------------+-----------------------------------------+
```

The units are in seconds. The format of the return value is:

```
Minimum / Average / Maximum / Standard Deviation / Sample Size
```

This variable periodically resets. You can control the reset interval using the *evs.stats_report_period* (page 285) parameter. The default value is 1 minute.

wsrep_evs_state

Shows the internal state of the EVS Protocol.

Example Value	
Location	Galera
Initial Version	3.8

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_evs_state';

+-----------------+-------------+
| Variable_name   | Value       |
+-----------------+-------------+
| wsrep_evs_state | OPERATIONAL |
+-----------------+-------------+
```

wsrep_flow_control_active

Whether flow control is currently active (replication paused) in the cluster.

Example Value	`false`
Location	Galera
Initial Version	3.31

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_flow_control_active';

+---------------------------+-----------+
| Variable_name             | Value     |
+---------------------------+-----------+
| wsrep_flow_control_active | true      |
+---------------------------+-----------+
```

wsrep_flow_control_paused

The fraction of time since the last `FLUSH STATUS` command that replication was paused due to flow control.

Example Value	`0.174353`
Location	Galera
Initial Version	

Basically, this is how much the replica lag is slowing down the cluster.

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_flow_control_paused';

+---------------------------+-----------+
| Variable_name             | Value     |
+---------------------------+-----------+
| wsrep_flow_control_paused | 0.184353  |
+---------------------------+-----------+
```

wsrep_flow_control_paused_ns

The total time spent in a paused state measured in nanoseconds.

Example Value	20222491180
Location	Galera
Initial Version	

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_flow_control_paused_ns';

+------------------------------+-------------+
| Variable_name                | Value       |
+------------------------------+-------------+
| wsrep_flow_control_paused_ns | 20222491180 |
+------------------------------+-------------+
```

wsrep_flow_control_recv

Returns the number of FC_PAUSE events the node has received, including those the node has sent. Unlike most status variables, the counter for this one does not reset every time you run the query.

Example Value	11
Location	Galera
Initial Version	

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_flow_control_recv';

+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| wsrep_flow_control_recv | 11    |
+-------------------------+-------+
```

wsrep_flow_control_requested

Whether the node has requested replication pause (received events queue too long).

Example Value	false
Location	Galera
Initial Version	3.31

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_flow_control_requested';

+------------------------------+-------+
| Variable_name                | Value |
+------------------------------+-------+
| wsrep_flow_control_requested | true  |
+------------------------------+-------+
```

wsrep_flow_control_sent

Returns the number of FC_PAUSE events the node has sent. Unlike most status variables, the counter for this one does not reset every time you run the query.

Example Value	7
Location	Galera
Initial Version	

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_flow_control_sent';

+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| wsrep_flow_control_sent | 7     |
+--------------------------+-------+
```

wsrep_gmcast_segment

Returns cluster segment the node belongs to.

Example Value	3
Location	Galera
Initial Version	3.31

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_gmcast_segment';

+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| wsrep_gmcast_segment | 0     |
+----------------------+-------+
```

wsrep_gcomm_uuid

Displays the group communications UUID.

Example Value	7e729708-605f-11e5-8ddd-8319a704b8c4
Location	Galera
Initial Version	1.0

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_gcomm_uuid';

+-----------------+--------------------------------------+
| Variable_name    | Value                                |
```

(continues on next page)

(continued from previous page)

```
+------------------+--------------------------------------+
| wsrep_gcomm_uuid | 7e729708-605f-11e5-8ddd-8319a704b8c4 |
+------------------+--------------------------------------+
```

wsrep_incoming_addresses

Comma-separated list of incoming server addresses in the cluster component.

Example Value	`10.0.0.1:3306,10.0.0.2:3306,undefined`
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_incoming_addresses';

+--------------------------+---------------------------------------+
| Variable_name            | Value                                 |
+--------------------------+---------------------------------------+
| wsrep_incoming_addresses | 10.0.0.1:3306,10.0.02:3306,undefined |
+--------------------------+---------------------------------------+
```

wsrep_ist_receive_status

This variable displays the IST progress for the joiner node. If IST is running, the value is the percentage of transfer completed. If IST is not running, the value is empty.

wsrep_last_committed

The sequence number, or seqno, of the last committed transaction. See *wsrep API* (page 16).

Example Value	`409745`
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_last_committed';

+----------------------+--------+
| Variable_name        | Value  |
+----------------------+--------+
| wsrep_last_committed | 409745 |
+----------------------+--------+
```

For more information, see *wsrep API* (page 16).

wsrep_local_bf_aborts

Total number of local transactions that were aborted by replica transactions while in execution.

Example Value	960
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_local_bf_aborts';

+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| wsrep_local_bf_aborts | 960   |
+-----------------------+-------+
```

wsrep_local_cached_downto

The lowest sequence number, or seqno, in the write-set cache (GCache).

Example Value	18446744073709551615
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_local_cached_downto';

+---------------------------+----------------------+
| Variable_name             | Value                |
+---------------------------+----------------------+
| wsrep_local_cached_downto | 18446744073709551615 |
+---------------------------+----------------------+
```

wsrep_local_cert_failures

Total number of local transactions that failed certification test.

Example Value	333
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_local_cert_failures';

+---------------------------+-------+
| Variable_name             | Value |
+---------------------------+-------+
```

(continues on next page)

(continued from previous page)

```
| wsrep_local_cert_failures | 333    |
+---------------------------+--------+
```

wsrep_local_commits

Total number of local transactions committed.

Example Value	14981
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_local_commits';

+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| wsrep_local_commits | 14981 |
+----------------------+-------+
```

wsrep_local_index

This node's index in the cluster (base 0).

Example Value	1
Location	MySQL
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_local_index';

+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| wsrep_local_index | 1     |
+-------------------+-------+
```

wsrep_local_recv_queue

Current (instantaneous) length of the receive queue, that is, the number of write-sets waiting to be applied.

Example Value	0
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_local_recv_queue';

+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| wsrep_local_recv_queue | 0     |
+-------------------------+-------+
```

wsrep_local_recv_queue_avg

Receive queue length averaged over interval since the last `FLUSH STATUS` command. Values considerably larger than `0.0` mean that the node cannot apply write-sets as fast as they are received and will generate a lot of replication throttling.

Example Value	`3.348452`
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_local_recv_queue_avg';

+----------------------------+----------+
| Variable_name              | Value    |
+----------------------------+----------+
| wsrep_local_recv_queue_avg | 3.348452 |
+----------------------------+----------+
```

wsrep_local_recv_queue_max

The maximum length of the receive queue since the last `FLUSH STATUS` command.

Example Value	`10`
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_local_recv_queue_max';

+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| wsrep_local_recv_queue_max | 10    |
+----------------------------+-------+
```

wsrep_local_recv_queue_min

The minimum length of the receive queue since the last `FLUSH STATUS` command.

Example Value	0
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_local_recv_queue_min';

+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| wsrep_local_recev_queue_min | 0     |
+----------------------------+-------+
```

wsrep_local_replays

Total number of transaction replays due to *asymmetric lock granularity*.

Example Value	0
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_local_replays';

+---------------------+-------+
| Variable_name       | Value |
+---------------------+-------+
| wsrep_lcoal_replays | 0     |
+---------------------+-------+
```

wsrep_local_send_queue

Current (instantaneous) length of the send queue, that is, the number of write-sets waiting to be sent.

Example Value	1
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_local_send_queue';

+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| wsrep_local_send_queue | 1     |
+------------------------+-------+
```

wsrep_local_send_queue_avg

Send queue length averaged over time since the last `FLUSH STATUS` command. Values considerably larger than 0.0 indicate replication throttling or network throughput issue.

Example Value	`0.145000`
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_local_send_queue_avg';

+----------------------------+----------+
| Variable_name              | Value    |
+----------------------------+----------+
| wsrep_local_send_queue_avg | 0.145000 |
+----------------------------+----------+
```

wsrep_local_send_queue_max

The maximum length of the send queue since the last `FLUSH STATUS` command.

Example Value	`10`
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_local_send_queue_max';

+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| wsrep_local_send_queue_max | 10    |
+----------------------------+-------+
```

wsrep_local_send_queue_min

The minimum length of the send queue since the last `FLUSH STATUS` command.

Example Value	`0`
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_local_send_queue_min';

+----------------------------+-------+
| Variable_name              | Value |
```

(continues on next page)

(continued from previous page)

```
+----------------------------+-------+
| wsrep_local_send_queue_min | 0     |
+----------------------------+-------+
```

wsrep_local_state

Internal Galera Cluster FSM state number.

Example Value	4
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_local_state';

+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| wsrep_local_state | 4     |
+-------------------+-------+
```

For more information on the possible node states, see *Node State Changes* (page 24).

wsrep_local_state_comment

Human-readable explanation of the state.

Example Value	Synced
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_local_state_comment';

+---------------------------+--------+
| Variable_name             | Value  |
+---------------------------+--------+
| wsrep_local_state_comment | Synced |
+---------------------------+--------+
```

wsrep_local_state_uuid

The UUID of the state stored on this node. If it matches the value in *wsrep_cluster_state_uuid* (page 313), the local and cluster nodes are in sync.

Example Value	e2c9a15e-5385-11e0-0800-6bbb637e7211
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_local_state_uuid';

+------------------------+--------------------------------------+
| Variable_name          | Value                                |
+------------------------+--------------------------------------+
| wsrep_local_state_uuid | e2c9a15e-5485-11e0-0800-6bbb637e7211 |
+------------------------+--------------------------------------+
```

For more information on the state UUID, see *wsrep API* (page 16).

wsrep_open_connections

The number of open connection objects inside the wsrep provider.

Example Value	1
Location	Galera
Initial Version	3.24

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_open_connections';

+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| wsrep_open_connections | 1     |
+------------------------+-------+
```

wsrep_open_transactions

The number of locally running transactions which have been registered inside the wsrep provider. This means transactions which have made operations which have caused write set population to happen. Transactions which are read only are not counted.

Example Value	6
Location	Galera
Initial Version	3.24

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_open_transactions';

+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| wsrep_open_transactions | 6     |
+------------------------+-------+
```

wsrep_protocol_version

The version of the wsrep Protocol used.

Example Value	4
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_protocol_version';

+------------------------+--------+
| Variable_name          | Value  |
+------------------------+--------+
| wsrep_protocol_version | 11     |
+------------------------+--------+
```

The following table summarizes protocol versions and the Galera version in which they were introduced:

Protocol version	Galera version
11	26.4.17
10	26.4.1
9	25.3.24
8	25.3.23
7	25.3.9
6	25.3.6
5	25.3.5

wsrep_provider_capabilities

This parameter indicates, which features are provided by the wsrep provider on this node. See also *wsrep_cluster_capabilities* (page 312).

Example Value	**MULTI_MASTER:CERTIFICATION:PARALLEL_APPLYING**
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_provider_capabilities';

+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| wsrep_provider_capabilities |       |
+----------------------------+-------+
```

wsrep_provider_name

The name of the wsrep Provider.

Example Value	Galera
Location	MySQL
Initial Version	???

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_provider_name';

+----------------------+---------+
| Variable_name        | Value   |
+----------------------+---------+
| wsrep_provider_name | Galera |
+----------------------+---------+
```

wsrep_provider_vendor

The name of the wsrep Provider vendor.

Example Value	Codership Oy <info@codership.com>
Location	MySQL
Initial Version	???

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_provider_vendor';

+------------------------+-----------------------------------+
| Variable_name          | Value                             |
+------------------------+-----------------------------------+
| wsrep_provider_vendor | Codership Oy <info@codership.com> |
+------------------------+-----------------------------------+
```

wsrep_provider_version

The wsrep Provider version string.

Example Value	25.3.5-wheezy(rXXXX)
Location	MySQL
Initial Version	???

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_provider_version';

+------------------------+----------------------+
| Variable_name          | Value                |
+------------------------+----------------------+
```

(continues on next page)

(continued from previous page)

```
| wsrep_provider_version | 25.3.5-wheezy(rXXXX) |
+------------------------+----------------------+
```

wsrep_ready

Whether the server is ready to accept queries. If this status is `OFF`, almost all of the queries will fail with:

```
ERROR 1047 (08S01) Unknown Command
```

unless the `wsrep_on` session variable is set to `0`.

Example Value	`ON`
Location	MySQL
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_ready';

+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| wsrep_ready   | ON    |
+---------------+-------+
```

wsrep_received

Total number of write-sets received from other nodes.

Example Value	`17831`
Location	MySQL
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_received';

+----------------+-------+
| Variable_name  | Value |
+----------------+-------+
| wsrep_received | 17831 |
+----------------+-------+
```

wsrep_received_bytes

Total byte size of write-sets received from other nodes.

Example Value	`6637093`
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_received_bytes';

+----------------------+---------+
| Variable_name        | Value   |
+----------------------+---------+
| wsrep_received_bytes | 6637093 |
+----------------------+---------+
```

wsrep_repl_data_bytes

Total size of data replicated.

Example Value	6526788
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_repl_data_bytes';

+------------------------+---------+
| Variable_name          | Value   |
+------------------------+---------+
| wsrep_repl_data_bytes | 6526788 |
+------------------------+---------+
```

wsrep_repl_keys

Total number of keys replicated.

Example Value	797399
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the `SHOW STATUS` statement like so:

```
SHOW STATUS LIKE 'wsrep_repl_keys';

+-----------------+--------+
| Variable_name   | Value  |
+-----------------+--------+
| wsrep_repl_keys | 797399 |
+-----------------+--------+
```

wsrep_repl_keys_bytes

Total size of keys replicated.

Example Value	11203721
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_repl_keys_bytes';

+----------------------+----------+
| Variable_name        | Value    |
+----------------------+----------+
| wsrep_repl_keys_bytes | 11203721 |
+----------------------+----------+
```

wsrep_repl_other_bytes

Total size of other bits replicated.

Example Value	0
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_repl_other_bytes';

+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| wsrep_repl_other_bytes | 0     |
+------------------------+-------+
```

wsrep_replicated

Total number of write-sets replicated (sent to other nodes).

Example Value	16109
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_replicated';

+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| wsrep_replicated | 16109 |
+-----------------+-------+
```

wsrep_replicated_bytes

Total size of write-sets replicated.

Example Value	6526788
Location	Galera
Initial Version	???

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_replicated_bytes';

+-----------------------+---------+
| Variable_name         | Value   |
+-----------------------+---------+
| wsrep_replicated_bytes | 6526788 |
+-----------------------+---------+
```

wsrep_rollbacker_thread_count

In MariaDB, stores the current number of rollbacker threads, to indicate how many rollbacker threads of this type there are.

Example Value	1
Location	Galera
Initial Version	MariaDB 10.2.26

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_rollbacker_thread_count';

+--------------------------------+---------+
| Variable_name                  | Value   |
+--------------------------------+---------+
| wsrep_rollbacker_thread_count  | 1       |
+--------------------------------+---------+
```

wsrep_thread_count

Total number of wsrep (applier/rollbacker) threads.

Example Value	2
Location	Galera
Initial Version	1.0

To retrieve the value of this status variable, execute the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'wsrep_thread_count';

+--------------------------------+---------+
| Variable_name                  | Value   |
```

(continues on next page)

(continued from previous page)

```
+--------------------------------+---------+
| wsrep_thread_count             | 2       |
+--------------------------------+---------+
```

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

8.5 XtraBackup-v2 Parameters

When using `xtrabackup-v2` as your *State Snapshot Transfer* method, you can fine tune how the script operates using the `[sst]` unit in the `my.cnf` configuration file.

```
[mysqld]
wsrep_sst_method=xtrabackup-v2

[sst]
compressor="gzip"
decompressor="gzip -dc"
rebuild=ON
compact=ON
encrypt=3
tkey="/path/to/key.pem"
tcert="/path/to/cert.pem"
tca="/path/to/ca.pem"
```

Bear in mind, some XtraBackup parameters require that you match the configuration on donor and joiner nodes, (as designated in the table below).

Option	Default	Match
compressor (page 336)		
cpat (page 337)	0	
decompressor (page 337)		
encrypt (page 337)	0	Yes
encrypt-algo (page 338)		
progress (page 338)		
rebuild (page 338)	0	
rlimit (page 339)		
sst_special_dirs (page 339)	1	
sockopt (page 339)		
streamfmt (page 340)	xbstream	Yes
tca (page 340)		
tcert (page 340)		
time (page 341)	0	
transferfmt (page 341)	socat	Yes
joiner_timeout (page 341)	60	
donor_timeout (page 342)	10	

compressor

Defines the compression utility the *Donor Node* uses to compress the state transfer.

System Variable	Name:	compressor
	Match:	Yes
Permitted Values	Type:	String
	Default Value:	

This parameter defines whether the donor node performs compression on the state transfer stream. It also defines what compression utility it uses to perform the operation. You can use any compression utility which works on a stream, such as `gzip` or `pigz`. Given that the *Joiner Node* must decompress the state transfer before attempting to read it, you must match this parameter with the *decompressor* (page 337) parameter, using the appropriate flags for each.

```
compression="gzip"
```

compact

Defines whether the joiner node performs compaction when rebuilding indexes after applying a *State Snapshot Transfer*.

System Variable	Name:	compact
	Match:	No
Permitted Values	Type:	Boolean
	Default Value:	OFF

This parameter operates on the joiner node with the *rebuild* (page 338) parameter. When enabled, the node performs compaction when rebuilding indexes after applying a state transfer.

```
rebuild=ON
compact=ON
```

cpat

Defines what files to exclude from the clean up from the datadir during state transfers.

System Variable	Name:	cpat
	Match:	No
Permitted Values	Type:	String
	Default Value:	See below

When the donor node begins a *State Snapshot Transfer*, it cleans up various files from the datadir. This ensures that the joiner node can cleanly apply the state transfer. With this parameter, you can define what files you want the node to exclude from being deleted, before the state transfer.

```
cpat='.\*\\.pem$\\|.\*init\\.ok$\\|.\*galera\\.cache$\\|.\*sst_in_progress$\\|.\*\\.
↪sst$\\|.\*gvwstate\\.dat$\\|.\*grastate\\.dat$\\|.\*\\.err$\\|.\*\\.log$\\|.\*RPM_
↪UPGRADE_MARKER$\\|.\*RPM_UPGRADE_HISTORY$'
```

decompressor

Defines the decompression utility the joiner node uses to decompress the state transfer.

System Variable	Name:	decompressor
	Match:	No
Permitted Values	Type:	String
	Default Value:	

This parameter defines whether the joiner node performs decompression on the state transfer stream. It also defines what decompression utility it uses to perform the operation. You can use any compression utility which works on a stream, such as `gzip` or `pigz`. Given that the donor node must compress the state transfer before sending it, you must match this parameter with the *compressor* (page 336) parameter, using the appropriate flags for each.

```
decompressor="gzip -dc"
```

encrypt

Defines whether the node uses SSL encryption for XtraBackup and what kind of encryption it uses.

System Variable	Name:	encrypt
	Match:	Yes
Permitted Values	Type:	Integer
	Default Value:	0

This parameter determines the type of SSL encryption the node uses when sending state transfers through xtrabackup. The recommended type is `2` when using the cluster over WAN.

Value	Description
0	No encryption.
1	The node encrypts State Snapshot Transfers through XtraBackup.
2	The node encrypts State Snapshot Transfers through OpenSSL, using Socat.
3	The node encrypts State Snapshot Transfers through the key and certificate files implemented for Galera Cluster.

```
encrypt=3
tkey="/path/to/key.pem"
tcert="/path/to/cert.pem"
tca="/path/to/ca.pem"
```

encrypt-algo

Defines the SSL encryption type the node uses for XtraBackup state transfers.

System Variable	Name:	encrypt-algo
	Match:	No
Permitted Values	Type:	Integer
	Default Value:	0

When using the *encrypt* (page 337) parameter in both the [xtrabackup] and [sst] units, there is a potential issue in it having different meanings according to the unit under which it occurs. That is, in [xtrabackup], it turns encryption on while in [sst] it both turns it on and specifies the algorithm.

In the event that you need to clarify the meaning, this parameter allows you to define the encryption algorithm separately from turning encryption on. It is only read in the event that *encrypt* (page 337) is set to 1

```
encrypt=1
encrypt-algo=3
```

progress

Defines whether where the node reports *State Snapshot Transfer* progress.

System Variable	Name:	progress
	Match:	No
Permitted Values	Type:	String
	Default Value:	
	Valid Values:	1; /path/to/file

When you set this parameter, the node reports progress on XtraBackup progress in state transfers. If you set the value to 1, the node makes these reports to the database server stderr. If you set the value to a file path, it writes the progress to that file.

Note: Keep in mind, that a 0 value is invalid. If you want to disable this parameter, delete or comment it out.

```
progress="/var/log/mysql/xtrabackup-progress.log"
```

rebuild

Defines whether the joiner node rebuilds indexes during a *State Snapshot Transfer*.

System Variable	Name:	rebuild
	Match:	No
Permitted Values	Type:	Boolean
	Default Value:	OFF

This parameter operates on the joiner node. When enabled, the node rebuilds indexes when applying the state transfer. Bear in mind, this operation is separate from compaction. Due to Bug #1192834, it is recommended that you use this parameter with *compact* (page 336).

```
rebuild=ON
compact=ON
```

rlimit

Defines the rate limit for the donor node.

System Variable	Name:	rlimit
	Match:	No
Permitted Values	Type:	Integer
	Default Value:	

This parameter allows you to define the rate-limit for the donor node. This allows you to keep state transfers from blocking regular cluster operations.

```
rlimit=300M
```

sst_special_dirs

Defines whether the node uses special InnoDB home and log directories.

System Variable	Name:	sst_special_dirs
	Match:	No
Permitted Values	Type:	Boolean
	Default Value:	1

This parameter was deprecated in Percona XtraDB Cluster 5.7.12-5rc1-26.16.

This parameter enables support for innodb_data_home_dir and innodb_log_home_dir parameters for XtraBackup. It requires that you define innodb_data_home_dir and innodb_log_group_home_dir in the [mysqld] unit.

```
[mysqld]
innodb_data_home_dir="/var/mysqld/innodb"
innodb_log_group_home_dir="/var/log/innodb"
wsrep_sst_method="xtrabackup-v2"

[sst]
sst_special_dirs=TRUE
```

sockopt

Defines socket options.

System Variable	Name:	sockopt
	Match:	No
Permitted Values	Type:	String
	Default Value:	

This parameter allows you to define one or more socket options for XtraBackup using the Socat transfer format.

streamfmt

Defines the stream formatting utility.

System Variable	Name:	streamfmt
	Match:	Yes
Permitted Values	Type:	String
	Default Value:	xbstream
	Valid Values:	tar; xbstream

This parameter defines the utility the node uses to archive the node state before the transfer is sent and how to unarchive the state transfers that it receives. There are two methods available: `tar` and `xbstream`. Given that the receiving node needs to know how to read the stream, it is necessary that both nodes use the same values for this parameter.

The default and recommended utility is `xbstream` given that it supports encryption, compression, parallel streaming, incremental backups and compaction. `tar` does not support these features.

```
streamfmt='xbstream'
```

tca

Defines the Certificate Authority (CA) to use in SSL encryption.

System Variable	Name:	tca
	Match:	No
Permitted Values	Type:	Path
	Default Value:	

This parameter defines the Certificate Authority (CA) file that the node uses with XtraBackup state transfers. In order to use SSL encryption with XtraBackup, you must configure the *transferfmt* (page 341) parameter to use `socat`.

For more information on using Socat with encryption, see Securing Traffic between Two Socat Instances using SSL.

```
transferfmt="socat"
tca="/path/to/ca.pem"
```

tcert

Defines the certificate to use in SSL encryption.

System Variable	Name:	tcert
	Match:	No
Permitted Values	Type:	String
	Default Value:	

This parameter defines the SSL certificate file that the node uses with SSL encryption on XtraBackup state transfers. In order to use SSL encryption with XtraBackup, you must configure the *transferfmt* (page 341) parameter to use Socat.

For more information on using Socat with encryption, see Securing Traffic between Two Socat Instances using SSL.

```
transferfmt="socat"
tcert="/path/to/cert.pem"
```

time

Defines whether XtraBackup instruments key stages in the backup and restore process for state transfers.

System Variable	Name:	time
	Match:	No
Permitted Values	Type:	Boolean
	Default Value:	OFF

This parameter instruments key stages of the backup and restore process for state transfers.

```
time=ON
```

transferfmt

Defines the transfer stream utility.

System Variable	Name:	transferfmt
	Match:	YesNo”
Permitted Values	Type:	String
	Default Value:	socat
	Valid Values:	socat; nc

This parameter defines the utility that the node uses to format transfers sent from donor to joiner nodes. There are two methods supported: `socat` and `nc`. Given that the receiving node needs to know how to interpret the transfer, it is necessary that both nodes use the same values for this parameter.

The default and recommended utility is Socat, given that it allows for socket options, such as transfer buffer size. For more information, see the socat Documentation.

```
transferfmt="socat"
```

joiner_timeout

How soon joiner should timeout waiting for SST (seconds).

System Variable	Name:	joiner_timeout
	Match:	No
Permitted Values	Type:	Integer
	Default Value:	60

This parameter determines the initial timeout in seconds for the joiner to receive the first packet in a *State Snapshot Transfer*. This keeps the joiner node from hanging in the event that the donor node crashes while starting the operation.

```
joiner_timeout=120
```

donor_timeout

How soon donor should timeout on connection to joiner (seconds).

System Variable	Name:	donor_timeout
	Match:	No
Permitted Values	Type:	Integer
	Default Value:	10

This parameter determines how soon the donor should timeout on connection to joiner and return to normal operation in case the joiner turns to be unresponsive.

```
donor_timeout=5
```

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- *Configuration Options* (page 345)
- *Destination Policies* (page 124)
- *LISTEN_ADDR* (page 344)
- *OTHER_OPTIONS* (page 344)
- *–watchdog* (page 351)
- Home
- *Docs* (page 1)
- KB
- Training

- FAQ

8.6 Galera Load Balancer Parameters

Galera Load Balancer provides simple TCP connection balancing developed with scalability and performance in mind. It draws on Pen for inspiration, but its functionality is limited to only balancing TCP connections.

It can be run either through the `service` command or the command-line interface of `glbd`. Configuration for Galera Load Balancer depends on which you use to run it.

Configuration Parameters

When Galera Load Balancer starts as a system service, it reads the `glbd.cfg` configuration file for default parameters you want to use. Only the *LISTEN_ADDR* (page 344) parameter is mandatory.

Parameter	Default Configuration
CONTROL_ADDR (page 343)	`127.0.0.1:8011`
CONTROL_FIFO (page 343)	`/var/run/glbd.fifo`
DEFAULT_TARGETS (page 344)	`127.0.0.1:80 10.0.1:80 10.0.0.2:80`
LISTEN_ADDR (page 344)	`8010`
MAX_CONN (page 344)	
OTHER_OPTIONS (page 344)	
THREADS (page 345)	`2`

CONTROL_ADDR

Defines the IP address and port for controlling connections.

Command-line Argument	*–control* (page 345)
Default Configuration	`127.0.0.1:8011`
Mandatory Parameter	No

This is an optional parameter. Use it to define the server used in controlling client connections. When using this parameter you must define the port. In the event that you do not define this parameter, Galera Load Balancer does not open the relevant socket.

```
CONTROL_ADDR="127.0.0.1:8011"
```

CONTROL_FIFO

Defines the path to the FIFO control file.

Command-line Argument	*–fifo* (page 347)
Default Configuration	`/var/run/glbd.fifo`
Mandatory Parameter	No

This is an optional parameter. It defines the path to the FIFO control file as is always opened. In the event that there is already a file at this path, Galera Load Balancer fails to start.

```
CONTROL_FIFO="/var/run/glbd.fifo"
```

DEFAULT_TARGETS

Defines the IP addresses and ports of the destination servers.

Default Configuration	127.0.0.1:80 10.0.0.1:80 10.0.0.2:80:2
Mandatory Parameter	No

This parameter defines that IP addresses that Galera Load Balancer uses as destination servers. Specifically, in this case the Galera Cluster nodes that it routes application traffic onto.

```
DEFAULT_TARGETS="192.168.1.1 192.168.1.2 192.168.1.3"
```

LISTEN_ADDR

Defines the IP address and port used for client connections.

Default Configuration	8010
Mandatory Parameter	Yes

This parameter defines the IP address and port that Galera Load Balancer listens on for incoming client connections. The IP address is optional, the port mandatory. In the event that you define a port without an IP address, Galera Load Balancer listens on that port for all available network interfaces.

```
LISTEN_ADDR="8010"
```

MAX_CONN

Defines the maximum allowed client connections.

Default Configuration	*–max_conn* (page 348)
Mandatory Parameter	No

This parameter defines the maximum number of client connections that you want to allow to Galera Load Balancer. It modifies the system open files limit to accommodate at least this many connections, provided sufficient privileges. It is recommend that you define this parameter if you expect the number of client connections to exceed five hundred.

```
MAX_CONN="135"
```

This option defines the maximum number of client connections that you want allow to Galera Load Balancer. Bear in mind, that it can be operating system dependent.

OTHER_OPTIONS

This parameter defines various additional options that you would like to pass to Galera Load Balancer, such as a destination selection policy or Watchdog configurations. Use the same syntax as you would for the command-line arguments. There is no default configuration, and this is not a mandatory parameter. For more information on the available options, see *Configuration Options* (page 345).

```
OTHER_OPTIONS="--random --watchdog exec:'mysql -utest -ptestpass' --discover"
```

THREADS

Defines the number of threads you want to use.

Default Configuration	*–threads* (page 350)
Mandatory Parameter	No

This parameter allows you to define the number of threads (that is, connection pools), which you want to allow Galera Load Balancer to use. It is advisable that you have at least a few per CPU core.

```
THREADS="6"
```

Configuration Options

When Galera Load Balancer starts as a daemon process, through the `/sbin/glbd` command, it allows you to pass a number of command-line arguments to configure how it operates. It uses the following syntax:

```
/usr/local/sbin/glbd [OPTIONS] LISTEN_ADDRESS [DESTINATION_LIST]
```

In the event that you would like to set any of these options when you run Galera Load Balancer as a service, you can define them through the *OTHER_OPTIONS* (page 344) parameter.

Long Argument	Short	Type	Parameter
–control (page 345)	`-c`	IP address	*CONTROL_ADDR* (page 343)
–daemon (page 346)	`-d`	Boolean	
–defer-accept (page 346)	`-a`	Boolean	
–discover (page 346)	`-D`	Boolean	
–extra (page 347)	`-x`	Decimal	
–fifo (page 347)	`-f`	File Path	*CONTROL_FIFO* (page 343)
–interval (page 347)	`-i`	Decimal	
–keepalive (page 347)	`-K`	Boolean	
–latency (page 348)	`-L`	Integer	
–linger (page 348)	`-l`	Boolean	
–max_conn (page 348)	`-m`	Integer	*MAX_CONN* (page 344)
–nodelay (page 349)	`-n`	Boolean	
–random (page 349)	`-r`	Boolean	
–round (page 349)	`-b`	Boolean	
–single (page 350)	`-S`	Boolean	
–source (page 350)	`-s`	Boolean	
–threads (page 350)	`-t`	Integer	*THREADS* (page 345)
–top (page 351)	`-T`	Boolean	
–verbose (page 351)	`-v`	Boolean	
–watchdog (page 351)	`-w`	String	

--control

Defines the IP address and port for control connections.

Short Argument	-c
Syntax	--control [IP|Hostname:]port
Type	IP Address
Configuration Parameter	*CONTROL_ADDR* (page 343)

For more information on defining the controlling connections, see the *CONTROL_ADDR* (page 343) parameter.

```
# glbd --control 192.168.1.1:80 3306 \
      192.168.1.1 192.168.1.2 192.168.1.3
```

--daemon

Defines whether you want Galera Load Balancer to run as a daemon process.

Short Argument	-d
Syntax	--daemon
Type	Boolean

This option defines whether you want to start `glbd` as a daemon process. That is, if you want it to run in the background, instead of claiming the current terminal session.

```
# glbd --daemon 3306 \
      192.168.1.1 192.168.1.2 192.168.1.3
```

--defer-accept

Enables TCP deferred acceptance on the listening socket.

Short Argument	-a
Syntax	--defer-accept
Type	Boolean

Enabling `TCP_DEFER_ACCEPT` allows Galera Load Balancer to awaken only when data arrives on the listening socket. It is disabled by default.

```
# glbd --defer-accept 3306 \
      192.168.1.1 192.168.1.2 192.168.1.3
```

--discover

Defines whether you want to use watchdog results to discover and set new destinations.

Short Argument	-D
Syntax	--discover
Type	Boolean

When you define the *–watchdog* (page 351) option, this option defines whether Galera Load Balancer uses the return value in discovering and setting new addresses for destination servers. For instance, after querying for the *wsrep_cluster_address* (page 236) parameter.

```
# glbd --discover -w exec:"mysql.sh -utest -ptestpass" 3306 \
      192.168.1.1 192.168.1.2 192.168.1.3
```

--extra

Defines whether you want to perform an extra destination poll on connection attempts.

Short Argument	-x
Syntax	--extra D.DDD
Type	Decimal

This option defines whether and when you want Galera Load Balancer to perform an additional destination poll on connection attempts. The given value indicates how many seconds after the previous poll that you want it to run the extra poll. By default, the extra polling feature is disabled.

```
# glbd --extra 1.35 3306 \
      192.168.1.1 192.168.1.2 192.168.1.3
```

--fifo

Defines the path to the FIFO control file.

Short Argument	-f
Syntax	--fifo /path/to/glbd.fifo
Type	File Path
Configuration Parameter	*CONTROL_FIFO* (page 343)

For more information on using FIFO control files, see the *CONTROL_FIFO* (page 343) parameter.

```
# glbd --fifo /var/run/glbd.fifo 3306 \
      192.168.1.1 192.168.1.2 192.168.1.3
```

--interval

Defines how often to probe destinations for liveliness.

Short Argument	-i
Syntax	--interval D.DDD
Type	Decimal

This option defines how often Galera Load Balancer checks destination servers for liveliness. It uses values given in seconds. By default, it checks liveliness every second.

```
# glbd --interval 2.013 3306 \
      192.168.1.1 192.168.1.2 192.168.1.3
```

--keepalive

Defines whether you want to disable the SO_KEEPALIVE socket option on server-side sockets.

Short Argument	-K
Syntax	--keepalive
Type	Boolean

Linux systems feature the socket option SO_KEEPALIVE, which causes the server to send packets to a remote system in order to maintain the client connection with the destination server. This option allows you to disable SO_KEEPALIVE on server-side sockets. It allows SO_KEEPALIVE by default.

```
# glbd --keepalive 3306 \
      192.168.1.1 192.168.1.2 192.168.1.3
```

--latency

Defines the number of samples to take in calculating latency for watchdog.

Short Argument	-L
Syntax	--latency N
Type	Integer

When the Watchdog module tests a destination server to calculate latency, it sends a number of packets through to measure its responsiveness. This option configures how many packets it sends in sampling latency.

```
# glbd --latency 25 3306 \
      192.168.1.1 192.168.1.2 192.168.1.3
```

--linger

Defines whether Galera Load Balancer disables sockets lingering after they are closed.

Short Argument	-l
Syntax	--linger
Type	Boolean

When Galera Load Balancer sends the close() command, occasionally sockets linger in a TIME_WAIT state. This option defines whether or not you want Galera Load Balancer to disable lingering sockets.

```
# glbd --linger 3306 \
      192.168.1.1 192.168.1.2 192.168.1.3
```

--max_conn

Defines the maximum allowed client connections.

Short Argument	-m
Syntax	--max_conn N
Type	Integer

For more information on defining the maximum client connections, see the *MAX_CONN* (page 344) parameter.

```
# glbd --max_conn 125 3306 \
      192.168.1.1 192.168.1.2 192.168.1.3
```

--nodelay

Defines whether it disables the TCP no-delay socket option.

Short Argument	-n
Syntax	--nodelay
Type	Boolean

Under normal operation, TCP connections automatically concatenate small packets into larger frames through the Nagle algorithm. In the event that you want Galera Load Balancer to disable this feature, this option causes it to open TCP connections with the TCP_NODELAY feature.

```
# glbd --nodelay 3306 \
      192.168.1.1 192.168.1.2 192.168.1.3
```

--random

Defines the destination selection policy as Random.

Short Argument	-r
Syntax	--random
Type	Boolean

The destination selection policy determines how Galera Load Balancer determines which servers to route traffic to. When you set the policy to Random, it randomly chooses a destination from the pool of available servers. You can enable this feature by default through the *OTHER_OPTIONS* (page 344) parameter.

For more information on other policies, see *Destination Selection Policies* (page 124).

```
# glbd --random 3306 \
      192.168.1.1 192.168.1.2 192.168.1.3
```

--round

Defines the destination selection policy as Round Robin.

Short Argument	-b
Syntax	--round
Type	Boolean

The destination selection policy determines how Galera Load Balancer determines which servers to route traffic to. When you set the policy to Round Robin, it directs new connections to the next server in a circular order list. You can enable this feature by default through the *OTHER_OPTIONS* (page 344) parameter.

For more information on other policies, see *Destination Selection Policies* (page 124).

```
# glbd --round 3306 \
      192.168.1.1 192.168.1.2 192.168.1.3
```

--single

Defines the destination selection policy as Single.

Short Argument	-S
Syntax	--single
Type	Boolean

The destination selection policy determines how Galera Load Balancer determines which servers to route traffic to.

When you set the policy to Single, all connections route to the server with the highest weight value. You can enable this by default through the *OTHER_OPTIONS* (page 344) parameter.

```
# glbd --single 3306 \
      192.168.1.1 192.168.1.2 192.168.1.3
```

--source

Defines the destination selection policy as Source Tracking.

Short Argument	-s
Syntax	--source
Type	Boolean

The destination selection policy determines how Galera Load Balancer determines which servers to route traffic to. When you set the policy to Source Tracking, connections that originate from one address are routed to the same destination. That is, you can ensure that certain IP addresses always route to the same destination server. You can enable this by default through the *OTHER_OPTIONS* (page 344) parameter.

Bear in mind, there are some limitations to this selection policy. When the destination list changes, the destination choice for new connections changes as well, while established connections remain in place. Additionally, when a destination is marked as unavailable, all connections that would route to it fail over to another, randomly chosen destination. When the original target becomes available again, routing to it for new connections resumes. In other words, Source Tracking works best with short-lived connections.

For more information on other policies, see *Destination Selection Policies* (page 124).

```
# glbd --source 3306 \
      192.168.1.1 192.168.1.2 192.168.1.3
```

--threads

Defines the number of threads that you want to use.

Short Argument	-t
Syntax	--threads N
Type	Integer

For more information on threading in Galera Load Balancer, see *THREADS* (page 345).

```
# glbd --threads 6 3306 \
      192.168.1.1 192.168.1.2 192.168.1.3
```

--top

Enables balancing to top weights only.

Short Argument	`-T`
Syntax	`--top`
Type	Boolean

This option restricts all balancing policies to a subset of destination servers with the top weight. For instance, if you have servers with weights `1`, `2` and `3`, balancing occurs only on servers with weight `3`, while they remain available.

```
# glbd --top 3306 \
      192.168.1.1 192.168.1.2 192.168.1.3
```

--verbose

Defines whether you want Galera Load Balancer to run as verbose.

Short Argument	`-v`
Syntax	`--verbose`
Type	Boolean

This option enables verbose output for Galera Load Balancer, which you may find useful for debugging purposes.

```
# glbd --verbose 3306 \
      192.168.1.1 192.168.1.2 192.168.1.3
```

--watchdog

Defines specifications for watchdog operations.

Short Argument	`-w`
Syntax	`--watchdog SPEC_STR`
Type	String

Under normal operation, Galera Load Balancer checks destination availability by attempting to establish a TCP connection to the server. For most use cases, this is insufficient. If you want to establish a connection with web server, you need to know if it is able to serve web pages. If you want to establish a connection with a database server, you need to know if it is able to execute queries. TCP connections do not provide that kind of information.

The Watchdog module implements asynchronous monitoring of destination servers through back-ends designed to service availability. This option allows you to enable it by defining the back-end ID string, optionally followed by a colon and the configuration options.

```
# glbd -w exec:"mysql.sh -utest -ptestpass" 3306 \
      192.168.1.1 192.168.1.2 192.168.1.3
```

This initializes the `exec` back-end to execute external programs. It runs the `mysql.sh` script on each destination server in order to determine its availability. You can find the `mysql.sh` in the Galera Load Balancer build directory, under `files/`.

Related Documents

- *Configuration Options* (page 345)
- *Destination Policies* (page 124)
- *LISTEN_ADDR* (page 344)
- *OTHER_OPTIONS* (page 344)
- *–watchdog* (page 351)

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search

Related Documents

- Galera Installation
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

8.7 Versioning Information

Galera Cluster for MySQL is available in binary software packages for several different Linux distributions, as well as in source code for other distributions and other Unix-like operating systems, such as FreeBSD (FreeBSD is available up to Galera Cluster for MySQL 5.7) and Solaris.

For Linux distributions, binary packages in 32-bit and 64-bit for both the MySQL database server with the wsrep API patch and the *Galera Replication Plugin* are available from the Codership Repository. These include support for:

- Red Hat Enterprise Linux
- CentOS
- Debian

- Ubuntu

By installing and configuring the Codership Repository on any of these systems, you can install and update Galera Cluster for MySQL through your package manager. In the event that you use a distribution of Linux that is not supported, or if you use another Unix-like operating system, source files are available on GitHub, at:

- MySQL Server with the wsrep API patch.
- Galera Replication Plugin.
- glb, the Galera Load Balancer.

For users of FreeBSD and similar operating systems, the Galera Replication Plugin is also available in ports, at `/usr/ports/databases/galera`, which corrects for certain compatibility issues with Linux dependencies.

For more information on the installation process, see Galera Installation.

Release Numbering Schemes

Software packages for Galera Cluster have their own release numbering schemas. There are two schemas to consider in version numbering:

- **Galera wsrep Provider** Also, referred to as the *Galera Replication Plugin*. The wsrep Provider uses the following versioning schema: `<wsrep API main version>.<Galera version>`. For example, release 24.2.4 indicates wsrep API version 24.x.x with Galera wsrep Provider version 2.4.
- **MySQL Server with wsrep API patch** The second versioning schema relates to the database server. Here, the MySQL server uses the following versioning schema `<MySQL server version>-<wsrep API version>`. For example, release 5.5.29-23.7.3 indicates a MySQL database server in 5.5.29 with wsrep API version 23.7.3.

For instances of Galera Cluster that use the MariaDB database server, consult the MariaDB documentation for version and release information.

See also Galera Cluster and MySQL Database Server Versions.

Third-party Implementations of Galera Cluster

In addition to the Galera Cluster for MySQL, the reference implementation from Codership Oy, there is a third-party implementation of Galera Cluster - MariaDB Galera Cluster which uses the Galera library for the replication implementation. To interface with the Galera Replication Plugin, MariaDB has been enhanced to support the replication API definition in the wsrep API project. Additionally, releases of MariaDB Server starting from version 10.1 on are packaged with Galera Cluster already included. For more information, see What is MariaDB Galera Cluster.

Related Documents

- Galera Installation

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ

- search
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

8.8 Legal Notice

Copyright (C) 2013 Codership Oy <info@codership.com>

This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported License. To view a copy of this license, visit Creative Commons Attribution-ShareAlike 3.0 Unported License.

Permission is granted to copy, distribute and modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Text, and no Back-Cover Text. To view a copy of that license, visit GNU Free Documentation License.

Any trademarks, logos, and service marks in this document are the property of Codership Oy or other third parties. You are not permitted to use these marks without the prior written consent of Codership Oy or such appropriate third party. Codership, Galera Cluster for MySQL, and the Codership logo are trademarks or registered trademarks of Codership.

All Materials in this document are (and shall continue to be) owned exclusively by Codership Oy or other respective third party owners and are protected under applicable copyrights, patents, trademarks, trade dress and other proprietary rights. Under no circumstances will any ownership rights or other interest in any materials be acquired by or through access or use of the materials. All rights, title and interest not expressly granted is reserved by Codership Oy.

- *"MySQL"* is a registered trademark of Oracle Corporation.
- *"Percona XtraDB Cluster"* and *"Percona Server"* are registered trademarks of Percona LLC.
- *"MariaDB"* and *"MariaDB Galera Cluster"* are registered trademarks of MariaDB Ab.

The Library

- *Documentation* (page 1)
- Knowledge Base
- Training
- Training Courses
- Tutorial Articles
- Training Videos
- FAQ
- search
- Home
- *Docs* (page 1)
- KB
- Training
- FAQ

8.9 Glossary

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.

Donor Node The node elected to provide a state transfer (*SST* or *IST*).

Galera Arbitrator An external process that functions as an additional node in certain cluster operations, such as *Quorum* calculations and generating consistent application state snapshots.

For example, consider a situation where your cluster becomes partitioned due to a loss of network connectivity that results in two components of equal size. Each component initiates quorum calculations to determine which should remain the *Primary Component* and which should become a non-operational component. If the components are of equal size, it risks a split-brain condition. Galera Arbitrator provides an addition vote in the quorum calculation, so that one component registers as larger than the other. The larger component then remains the Primary Component.

Unlike the main `mysqld` process, `garbd` does not generate replication events of its own and does not store replication data. It does, however, acknowledge all replication events. Furthermore, you can route replication through Galera Arbitrator, such as when generating a consistent application state snapshot for backups.

For more information, see *Galera Arbitrator* (page 105) and *Backing Up Cluster Data* (page 109).

Galera Replication Plugin Galera Replication Plugin is a general purpose replication plugin for any transactional system. It can be used to create a synchronous multi-primary replication solution to achieve high availability and scale-out.

See *Galera Replication Plugin* (page 16) for more details.

GCache See *Write-set Cache*.

Global Transaction ID To keep the state identical on all nodes, the *wsrep API* uses global transaction IDs (GTID), which are used to identify the state change and to identify the state itself by the ID of the last state change

The GTID consists of a state UUID, which uniquely identifies the state and the sequence of changes it undergoes, and an ordinal sequence number (seqno, a 64-bit signed integer) to denote the position of the change in the sequence.

For more information on Global Transaction ID's, see *wsrep API* (page 16).

Incremental State Transfer In an Incremental State Transfer (IST) a node only receives the missing write-sets and catches up with the group by replaying them. See also the definition for State Snapshot Transfer (SST).

For more information on IST's, see *Incremental State Transfer (IST)* (page 20).

IST See *Incremental State Transfer*.

Joiner Node The node joining the cluster, usually a state transfer target.

Logical State Transfer Method This is a type of back-end state transfer method that operates through the database server (for example, `mysqldump`).

For more information, see *Logical State Snapshot* (page 68).

NBO See *Non-Blocking Operations*.

Node A cluster node – a single mysql instance that is in the cluster.

Non-Blocking Operations With the NBO method, DDL statements are processed similarly as with the TOI (Total Order Isolation) method, except that the NBO method uses a more efficient locking strategy. Compared with TOI, NBO offers significant flexibility in DDL statement processing.

For more information, see *Non-Blocking Operations* (page 84).

PF PF is packet filter used for firewall software, that has been ported to several operating systems. It can be configured for firewall protection of a Galera Cluster.

Physical State Transfer Method This is another type of back-end state transfer method, but it operates on the physical media in the datadir (for example, `rsync` and `xtrabackup`).

For more information, see *Physical State Snapshot* (page 71).

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.

Primary Component In addition to single-node failures, the cluster may be split into several components due to network failure. In such a situation, only one of the components can continue to modify the database state to avoid history divergence. This component is called the Primary Component (PC).

For more information on the Primary Component, see *Quorum Components* (page 28).

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.

Rolling Schema Upgrade The rolling schema upgrade is a DDL processing method in which the DDL will only be processed locally on the node. The node is desynchronized from the cluster for the duration of the DDL processing in a way that it does not block the other nodes. When the DDL processing is complete, the node applies the delayed replication events and synchronizes with the cluster.

For more information, see *Rolling Schema Upgrade* (page 83).

RSU See *Rolling Schema Upgrade*.

seqno See *Sequence Number*.

Sequence Number This is a 64-bit signed integer that the node uses to denote the position of a given transaction in the sequence. The seqno is second component to the *Global Transaction ID*.

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.

SST See *State Snapshot Transfer*.

State Snapshot Transfer State Snapshot Transfer refers to a full data copy from one cluster node (that is, a donor) to the joining node (that is, a joiner). See also the definition for Incremental State Transfer (IST).

For more information, see *State Snapshot Transfer (SST)* (page 67).

State UUID Unique identifier for the state of a node and the sequence of changes it undergoes. It is the first component of the *Global Transaction ID*.

Streaming Replication This provides an alternative replication method for handling large or long-running write transactions. It is a new feature in version 4.0 of Galera Cluster. In older versions, the feature is unsupported.

Under normal operation, the node performs all replication and certification operations when the transaction commits. With large transactions this can result in conflicts if smaller transactions are committed first. With Streaming Replication, the node breaks the transaction into fragments, then certifies and replicates them to all nodes while the transaction is still in progress. Once certified, a fragment can no longer be aborted by a conflicting transaction.

For more information see *Streaming Replication* (page 33) and *Using Streaming Replication* (page 103).

TOI See *Total Order Isolation*.

Total Order Isolation By default, DDL statements are processed by using the Total Order Isolation (TOI) method. In TOI, the query is replicated to the nodes in a statement form before executing on the primary. The query waits for all preceding transactions to commit and then gets executed in isolation on all nodes, simultaneously.

For more information, see *Total Order Isolation* (page 83).

write-set Transaction commits the node sends to and receives from the cluster.

Write-set Cache Galera stores write-sets in a special cache called, Write-set Cache (GCache). GCache is a memory allocator for write-sets. Its primary purpose is to minimize the write set footprint on the RAM.

For more information, see *Write-Set Cache (GCache)* (page 21).

wsrep API The wsrep API is a generic replication plugin interface for databases. The API defines a set of application callbacks and replication plugin calls.

For more information, see *wsrep API* (page 16).

INDEX

Symbols

.. rubric:: ''wsrep_cluster_capabilities''
    Status Variables, 312
.. rubric:: ''wsrep_provider_capabilities''
    Status Variables, 329

A

Asynchronous replication
    Descriptions, 11

B

base_dir
    wsrep Provider Options, 278
base_host
    wsrep Provider Options, 278
base_port
    wsrep Provider Options, 279

C

cert.log_conflicts
    wsrep Provider Options, 279
cert.optimistic_pa
    wsrep Provider Options, 279
Certification based replication
    Descriptions, 1
Checking
    wsrep Provider Options, 306
Cluster Replication, **355**
Configuration
    pc.recovery, 91

D

Database cluster
    Descriptions, 9
debug
    wsrep Provider Options, 279
Debug log
    Logs, 231
Descriptions
    Asynchronous replication, 11
    Certification based replication, 1
    Database cluster, 9
    Eager replication, 11
    Galera Arbitrator, 105
    GCache, 21
    Global Transaction ID, 16
    Lazy replication, 11
    Non-Blocking Operations, 84
    Rolling Schema Upgrade, 84
    Synchronous replication, 11
    Total Order Isolation, 83
    Virtual Synchrony, 17
    Virtual synchrony, 1
    Weighted Quorum, 29
    Writeset Cache, 21
    wsrep API, 16
DONOR
    Node states, 23
Donor Node, **355**
Drupal, 231

E

Eager replication
    Descriptions, 11
ER_UNKNOWN_COM_ERROR
    Errors, 261
Errors
    ER_UNKNOWN_COM_ERROR, 261
evs.auto_evict
    wsrep Provider Options, 280
evs.causal_keepalive_period
    wsrep Provider Options, 280
evs.consensus_timeout
    Parameters, 26
    wsrep Provider Options, 280
evs.debug_log_mask
    wsrep Provider Options, 281
evs.delay_margin
    wsrep Provider Options, 281
evs.delayed_keep_period
    wsrep Provider Options, 281
evs.evict
    wsrep Provider Options, 282
evs.inactive_check_period

Parameters, 26
wsrep Provider Options, 282
evs.inactive_timeout
Parameters, 26
wsrep Provider Options, 282
evs.info_log_mask
wsrep Provider Options, 283
evs.install_timeout
wsrep Provider Options, 283
evs.join_retrans_period
wsrep Provider Options, 283
evs.keepalive_period
Parameters, 26
wsrep Provider Options, 284
evs.max_install_timeouts
wsrep Provider Options, 284
evs.send_window
wsrep Provider Options, 284
evs.stats_report_period
wsrep Provider Options, 285
evs.suspect_timeout
Parameters, 26
wsrep Provider Options, 285
evs.use_aggregate
wsrep Provider Options, 285
evs.user_send_window
Parameters, 286
evs.version
wsrep Provider Options, 286
evs.view_forget_timeout
wsrep Provider Options, 286

F

Functions
WSREP_LAST_SEE_GTID(), 269
WSREP_LAST_WRITTEN_GTID(), 269
WSREP_SYNC_WAIT_UPTO_GTID(), 270

G

Galera Arbitrator, **355**
Galera Arbitrator, 109
Descriptions, 105
Logs, 105
Galera Cluster 4.x
Streaming Replication, 33, 103, 267
Synchronization Functions, 269, 270
System Tables, 77
Galera Replication Plugin, **355**
GCache, **355**
GCache
Descriptions, 21
gcache.dir
wsrep Provider Options, 287
gcache.keep_pages_size
wsrep Provider Options, 287
gcache.mem_size
wsrep Provider Options, 287
gcache.name
wsrep Provider Options, 287
gcache.page_size
wsrep Provider Options, 288
gcache.recover
wsrep Provider Options, 288
gcache.size
wsrep Provider Options, 289
gcomm.thread_prio
wsrep Provider Options, 289
gcs.fc_debug
wsrep Provider Options, 290
gcs.fc_factor
wsrep Provider Options, 290
gcs.fc_limit
wsrep Provider Options, 290
gcs.fc_master_slave
wsrep Provider Options, 290
gcs.fc_single_primary
wsrep Provider Options, 290
gcs.max_packet_size
wsrep Provider Options, 291
gcs.max_throttle
wsrep Provider Options, 291
gcs.recv_q_hard_limit
wsrep Provider Options, 291
gcs.recv_q_soft_limit
wsrep Provider Options, 292
gcs.sync_donor
wsrep Provider Options, 292
gcs.vote_policy
wsrep Provider Options, 292
Global Transaction ID, **355**
Global Transaction ID
Descriptions, 16
gmcast.isolate
wsrep Provider Options, 293
gmcast.listen_addr
Parameters, 109
wsrep Provider Options, 293
gmcast.mcast_addr
wsrep Provider Options, 293
gmcast.mcast_ttl
wsrep Provider Options, 294
gmcast.peer_timeout
wsrep Provider Options, 294
gmcast.segment
wsrep Provider Options, 294
gmcast.time_wait
wsrep Provider Options, 295
gmcast.version

wsrep Provider Options, 295
gvwstate.dat, 296

I

Incremental State Transfer, **355**
Incremental State Transfer
wsrep Provider Options, 20
innodb-wsrep-applier-lock-wait-timeout
Parameters, 233
innodb_flush_log_at_trx_commit
wsrep Provider Options, 295
IST, **355**
ist.recv_addr
wsrep Provider Options, 296
ist.recv_bind
wsrep Provider Options, 296

J

JOINED
Node states, 23
JOINER
Node states, 23
Joiner Node, **355**

L

Lazy replication
Descriptions, 11
Logical State Transfer Method, **355**
Logs
Debug log, 231
Galera Arbitrator, 105
mysqld error log, 109

M

my.cnf, 236
mysqld error log
Logs, 109

N

NBO, **355**
Node, **355**
Node state changes
Node states, 24
Node states
DONOR, 23
JOINED, 23
JOINER, 23
Node state changes, 24
OPEN, 23
PRIMARY, 23
SYNCED, 23
Non-Blocking Operations, **355**
Non-Blocking Operations
Descriptions, 84

O

OPEN
Node states, 23

P

pairs: Parameters
wsrep_applier_UK_checks, 259
wsrep_slave_UK_checks, 260
Parameters
evs.consensus_timeout, 26
evs.inactive_check_period, 26
evs.inactive_timeout, 26
evs.keepalive_period, 26
evs.suspect_timeout, 26
evs.user_send_window, 286
gmcast.listen_addr, 109
innodb-wsrep-applier-lock-wait-timeout, 233
pc.bootstrap, 93
pc.npvo, 299
pc.weight, 31
plugin_wsrep_provider, 233
socket.ssl_cert, 215
socket.ssl_cipher, 215
socket.ssl_compression, 215
socket.ssl_key, 215
wsrep_applier_FK_checks, 257
wsrep_applier_FK_failure_retries, 234
wsrep_applier_threads, 258
wsrep_auto_increment_control, 234
wsrep_causal_reads, 235, 265
wsrep_cert_deps_distance, 142
wsrep_certification_rules, 235
wsrep_certify_nonPK, 236
wsrep_cluster_address, 89, 141, 236
wsrep_cluster_conf_id, 140
wsrep_cluster_name, 109, 237
wsrep_cluster_size, 140
wsrep_cluster_state_uuid, 140
wsrep_cluster_status, 140
wsrep_connected, 141
wsrep_convert_lock_to_trx, 238
wsrep_data_dir, 65
wsrep_data_home_dir, 239
wsrep_dbug_option, 239
wsrep_debug, 240
wsrep_desync, 241
wsrep_dirty_reads, 242
wsrep_drupal_282555_workaround, 243
wsrep_evs_repl_latency, 317
wsrep_flow_control_paused, 142
wsrep_forced_binlog_format, 243
wsrep_ignore_apply_errors, 244

wsrep_info_level, 244
wsrep_last_committed, 93
wsrep_load_data_splitting, 245
wsrep_local_recv_queue_avg, 142
wsrep_local_recv_queue_max, 142
wsrep_local_recv_queue_min, 142
wsrep_local_send_queue_avg, 144
wsrep_local_send_queue_max, 144
wsrep_local_send_queue_min, 144
wsrep_local_state_comment, 141
wsrep_log_conflicts, 245
wsrep_max_ws_rows, 246
wsrep_max_ws_size, 246
wsrep_mode, 247
wsrep_node_address, 248
wsrep_node_incoming_address, 249
wsrep_node_name, 65, 109, 250
wsrep_notify_cmd, 139, 251
wsrep_on, 252
wsrep_OSU_method, 84, 253
wsrep_preordered, 254
wsrep_provider, 254
wsrep_provider_options, 29, 93, 255
wsrep_ready, 141
wsrep_restart_replica, 256
wsrep_restart_slave, 257
wsrep_retry_autocommit, 257
wsrep_slave_FK_checks, 258
wsrep_slave_threads, 259
wsrep_sst_auth, 260
wsrep_sst_donor, 65, 260
wsrep_sst_donor_rejects_queries, 261
wsrep_sst_method, 20, 262
wsrep_sst_receive_address, 263
wsrep_start_position, 264
wsrep_status_file, 264
wsrep_sync_server_uuid, 265
wsrep_sync_wait, 265
wsrep_trx_fragment_size, 267
wsrep_trx_fragment_unit, 267
pc.announce_timeout
wsrep Provider Options, 297
pc.bootstrap
Parameters, 93
wsrep Provider Options, 297
pc.checksum
wsrep Provider Options, 297
pc.ignore_quorum
wsrep Provider Options, 298
pc.ignore_sb
wsrep Provider Options, 298
pc.linger
wsrep Provider Options, 298
pc.npvo
Parameters, 299
pc.recovery
Configuration, 91
wsrep Provider Options, 296
pc.version
wsrep Provider Options, 300
pc.wait_prim
wsrep Provider Options, 299
pc.wait_prim_timeout
wsrep Provider Options, 299
pc.weight
Parameters, 31
wsrep Provider Options, 300
PF, **356**
Physical State Transfer Method, **356**
plugin_wsrep_provider
Parameters, 233
PRIMARY
Node states, 23
Primary Cluster, **356**
Primary Component, **356**
Primary Component
Nominating, 93
protonet.backend
wsrep Provider Options, 300
protonet.version
wsrep Provider Options, 301

Q

Quorum, **356**

R

repl.causal_read_timeout
wsrep Provider Options, 301
repl.commit_order
wsrep Provider Options, 301
repl.key_format
wsrep Provider Options, 302
repl.max_ws_size
wsrep Provider Options, 302
repl.proto_max
wsrep Provider Options, 302
Rolling Schema Upgrade, **356**
Rolling Schema Upgrade
Descriptions, 84
RSU, **356**

S

seqno, **356**
Sequence Number, **356**
Setting
wsrep Provider Options, 306
socket.checksum
wsrep Provider Options, 304

socket.dynamic
    wsrep Provider Options, 304
socket.recv_buf_size
    wsrep Provider Options, 303
socket.send_buf_size
    wsrep Provider Options, 303
socket.ssl
    wsrep Provider Options, 303
socket.ssl_ca
    wsrep Provider Options, 303
socket.ssl_cert
    Parameters, 215
    wsrep Provider Options, 304
socket.ssl_cipher
    Parameters, 215
    wsrep Provider Options, 305
socket.ssl_compression
    Parameters, 215
    wsrep Provider Options, 305
socket.ssl_key
    Parameters, 215
    wsrep Provider Options, 305
socket.ssl_password_file
    wsrep Provider Options, 306
socket.ssl_reload
    wsrep Provider Options, 306
Split Brain, **356**
Split-brain
    Descriptions, 29
    Prevention, 105
    Recovery, 93
SST, **356**
State Snapshot Transfer, **356**
State Snapshot Transfer
    State Snapshot Transfer methods, 20
State Snapshot Transfer methods
    Incremental State Transfer, 20
    State Snapshot Transfer, 20
State UUID, **356**
Status Variables
    .. rubric:: ``wsrep_cluster_capabilities``
    312
    .. rubric:: ``wsrep_provider_capabilities``
    329
    wsrep_applier_thread_count, 309
    wsrep_apply_oooe, 309
    wsrep_apply_oool, 310
    wsrep_apply_waits, 310
    wsrep_apply_window, 310
    wsrep_cert_deps_distance, 311
    wsrep_cert_index_size, 311
    wsrep_cert_interval, 311
    wsrep_cluster_conf_id, 312
    wsrep_cluster_size, 313
    wsrep_cluster_state_uuid, 313
    wsrep_cluster_status, 313
    wsrep_cluster_weight, 314
    wsrep_commit_oooe, 314
    wsrep_commit_oool, 315
    wsrep_commit_window, 315
    wsrep_connected, 315
    wsrep_desync_count, 316
    wsrep_evs_delayed, 316
    wsrep_evs_evict_list, 316
    wsrep_evs_state, 317
    wsrep_flow_control_active, 318
    wsrep_flow_control_paused, 318
    wsrep_flow_control_paused_ns, 318
    wsrep_flow_control_recv, 319
    wsrep_flow_control_requested, 319
    wsrep_flow_control_sent, 320
    wsrep_gcomm_uuid, 320
    wsrep_gmcast_segment, 320
    wsrep_incoming_addresses, 321
    wsrep_ist_receive_status, 321
    wsrep_last_committed, 321
    wsrep_local_bf_aborts, 322
    wsrep_local_cached_downto, 322
    wsrep_local_cert_failures, 322
    wsrep_local_commits, 323
    wsrep_local_index, 323
    wsrep_local_recv_queue, 323
    wsrep_local_recv_queue_avg, 324
    wsrep_local_recv_queue_max, 324
    wsrep_local_recv_queue_min, 324
    wsrep_local_replays, 325
    wsrep_local_send_queue, 325
    wsrep_local_send_queue_avg, 326
    wsrep_local_send_queue_max, 326
    wsrep_local_send_queue_min, 326
    wsrep_local_state, 327
    wsrep_local_state_comment, 327
    wsrep_local_state_uuid, 327
    wsrep_open_connections, 328
    wsrep_open_transactions, 328
    wsrep_protocol_version, 329
    wsrep_provider_name, 330
    wsrep_provider_vendor, 330
    wsrep_provider_version, 330
    wsrep_ready, 331
    wsrep_received, 331
    wsrep_received_bytes, 331
    wsrep_repl_data_bytes, 332
    wsrep_repl_keys, 332
    wsrep_repl_keys_bytes, 332
    wsrep_repl_other_bytes, 333
    wsrep_replicated, 333
    wsrep_replicated_bytes, 334

wsrep_rollbacker_thread_count, 334
wsrep_thread_count, 334
Streaming Replication, **356**
Streaming Replication
Galera Cluster 4.x, 33, 103, 267
wsrep_trx_fragment_size, 267
wsrep_trx_fragment_unit, 267
SYNCED
Node states, 23
Synchronization Functions
Galera Cluster 4.x, 269, 270
Synchronous replication
Descriptions, 11
System Tables
Galera Cluster 4.x, 77

T

TOI, **356**
Total Order Isolation, **356**
Total Order Isolation, 65
Descriptions, 83

V

Virtual Synchrony
Descriptions, 17
Virtual synchrony
Descriptions, 1

W

Weighted Quorum
Descriptions, 29
write-set, **357**
Write-set Cache, **357**
Writeset Cache
Descriptions, 21
wsrep API, **357**
wsrep API
Descriptions, 16
wsrep Provider Options
base_dir, 278
base_host, 278
base_port, 279
cert.log_conflicts, 279
cert.optimistic_pa, 279
Checking, 306
debug, 279
evs.auto_evict, 280
evs.causal_keepalive_period, 280
evs.consensus_timeout, 280
evs.debug_log_mask, 281
evs.delay_margin, 281
evs.delayed_keep_period, 281
evs.evict, 282
evs.inactive_check_period, 282
evs.inactive_timeout, 282
evs.info_log_mask, 283
evs.install_timeout, 283
evs.join_retrans_period, 283
evs.keepalive_period, 284
evs.max_install_timeouts, 284
evs.send_window, 284
evs.stats_report_period, 285
evs.suspect_timeout, 285
evs.use_aggregate, 285
evs.version, 286
evs.view_forget_timeout, 286
gcache.dir, 287
gcache.keep_pages_size, 287
gcache.mem_size, 287
gcache.name, 287
gcache.page_size, 288
gcache.recover, 288
gcache.size, 289
gcomm.thread_prio, 289
gcs.fc_debug, 290
gcs.fc_factor, 290
gcs.fc_limit, 290
gcs.fc_master_slave, 290
gcs.fc_single_primary, 290
gcs.max_packet_size, 291
gcs.max_throttle, 291
gcs.recv_q_hard_limit, 291
gcs.recv_q_soft_limit, 292
gcs.sync_donor, 292
gcs.vote_policy, 292
gmcast.isolate, 293
gmcast.listen_addr, 293
gmcast.mcast_addr, 293
gmcast.mcast_ttl, 294
gmcast.peer_timeout, 294
gmcast.segment, 294
gmcast.time_wait, 295
gmcast.version, 295
innodb_flush_log_at_trx_commit, 295
ist.recv_addr, 296
ist.recv_bind, 296
pc.announce_timeout, 297
pc.bootstrap, 297
pc.checksum, 297
pc.ignore_quorum, 298
pc.ignore_sb, 298
pc.linger, 298
pc.recovery, 296
pc.version, 300
pc.wait_prim, 299
pc.wait_prim_timeout, 299
pc.weight, 300
protonet.backend, 300

protonet.version, 301
repl.causal_read_timeout, 301
repl.commit_order, 301
repl.key_format, 302
repl.max_ws_size, 302
repl.proto_max, 302
Setting, 306
socket.checksum, 304
socket.dynamic, 304
socket.recv_buf_size, 303
socket.send_buf_size, 303
socket.ssl, 303
socket.ssl_ca, 303
socket.ssl_cert, 304
socket.ssl_cipher, 305
socket.ssl_compression, 305
socket.ssl_key, 305
socket.ssl_password_file, 306
socket.ssl_reload, 306
wsrep_applier_FK_checks
Parameters, 257
wsrep_applier_FK_failure_retries
Parameters, 234
wsrep_applier_thread_count
Status Variables, 309
wsrep_applier_threads
Parameters, 258
wsrep_apply_oooe
Status Variables, 309
wsrep_apply_oool
Status Variables, 310
wsrep_apply_waits
Status Variables, 310
wsrep_apply_window
Status Variables, 310
wsrep_auto_increment_control
Parameters, 234
wsrep_causal_reads
Parameters, 235, 265
wsrep_cert_deps_distance
Parameters, 142
Status Variables, 311
wsrep_cert_index_size
Status Variables, 311
wsrep_cert_interval
Status Variables, 311
wsrep_certification_rules
Parameters, 235
wsrep_certify_nonPK
Parameters, 236
wsrep_cluster_address
Parameters, 89, 141, 236
wsrep_cluster_conf_id
Parameters, 140
Status Variables, 312
wsrep_cluster_name
Parameters, 109, 237
wsrep_cluster_size
Parameters, 140
Status Variables, 313
wsrep_cluster_state_uuid
Parameters, 140
Status Variables, 313
wsrep_cluster_status
Parameters, 140
Status Variables, 313
wsrep_cluster_weight
Status Variables, 314
wsrep_commit_oooe
Status Variables, 314
wsrep_commit_oool
Status Variables, 315
wsrep_commit_window
Status Variables, 315
wsrep_connected
Parameters, 141
Status Variables, 315
wsrep_convert_lock_to_trx
Parameters, 238
wsrep_data_dir
Parameters, 65
wsrep_data_home_dir
Parameters, 239
wsrep_dbug_option
Parameters, 239
wsrep_debug
Parameters, 240
wsrep_desync
Parameters, 241
wsrep_desync_count
Status Variables, 316
wsrep_dirty_reads
Parameters, 242
wsrep_drupal_282555_workaround
Parameters, 243
wsrep_evs_delayed
Status Variables, 316
wsrep_evs_evict_list
Status Variables, 316
wsrep_evs_repl_latency
Parameters, 317
wsrep_evs_state
Status Variables, 317
wsrep_flow_control_active
Status Variables, 318
wsrep_flow_control_paused
Parameters, 142
Status Variables, 318

wsrep_flow_control_paused_ns
    Status Variables, 318
wsrep_flow_control_recv
    Status Variables, 319
wsrep_flow_control_requested
    Status Variables, 319
wsrep_flow_control_sent
    Status Variables, 320
wsrep_forced_binlog_format
    Parameters, 243
wsrep_gcomm_uuid
    Status Variables, 320
wsrep_gmcast_segment
    Status Variables, 320
wsrep_ignore_apply_errors
    Parameters, 244
wsrep_incoming_addresses
    Status Variables, 321
wsrep_info_level
    Parameters, 244
wsrep_ist_receive_status
    Status Variables, 321
wsrep_last_committed
    Parameters, 93
    Status Variables, 321
WSREP_LAST_SEE_GTID()
    Functions, 269
WSREP_LAST_WRITTEN_GTID()
    Functions, 269
wsrep_load_data_splitting
    Parameters, 245
wsrep_local_bf_aborts
    Status Variables, 322
wsrep_local_cached_downto
    Status Variables, 322
wsrep_local_cert_failures
    Status Variables, 322
wsrep_local_commits
    Status Variables, 323
wsrep_local_index
    Status Variables, 323
wsrep_local_recv_queue
    Status Variables, 323
wsrep_local_recv_queue_avg
    Parameters, 142
    Status Variables, 324
wsrep_local_recv_queue_max
    Parameters, 142
    Status Variables, 324
wsrep_local_recv_queue_min
    Parameters, 142
    Status Variables, 324
wsrep_local_replays
    Status Variables, 325
wsrep_local_send_queue
    Status Variables, 325
wsrep_local_send_queue_avg
    Parameters, 144
    Status Variables, 326
wsrep_local_send_queue_max
    Parameters, 144
    Status Variables, 326
wsrep_local_send_queue_min
    Parameters, 144
    Status Variables, 326
wsrep_local_state
    Status Variables, 327
wsrep_local_state_comment
    Parameters, 141
    Status Variables, 327
wsrep_local_state_uuid
    Status Variables, 327
wsrep_log_conflicts
    Parameters, 245
wsrep_max_ws_rows
    Parameters, 246
wsrep_max_ws_size
    Parameters, 246
wsrep_mode
    Parameters, 247
wsrep_node_address
    Parameters, 248
wsrep_node_incoming_address
    Parameters, 249
wsrep_node_name
    Parameters, 65, 109, 250
wsrep_notify_cmd
    Parameters, 139, 251
wsrep_on
    Parameters, 252
wsrep_open_connections
    Status Variables, 328
wsrep_open_transactions
    Status Variables, 328
wsrep_OSU_method
    Parameters, 84, 253
wsrep_preordered
    Parameters, 254
wsrep_protocol_version
    Status Variables, 329
wsrep_provider
    Parameters, 254
wsrep_provider_name
    Status Variables, 330
wsrep_provider_options
    Parameters, 29, 93, 255
wsrep_provider_vendor
    Status Variables, 330

wsrep_provider_version
    Status Variables, 330
wsrep_ready
    Parameters, 141
    Status Variables, 331
wsrep_received
    Status Variables, 331
wsrep_received_bytes
    Status Variables, 331
wsrep_repl_data_bytes
    Status Variables, 332
wsrep_repl_keys
    Status Variables, 332
wsrep_repl_keys_bytes
    Status Variables, 332
wsrep_repl_other_bytes
    Status Variables, 333
wsrep_replicated
    Status Variables, 333
wsrep_replicated_bytes
    Status Variables, 334
wsrep_restart_replica
    Parameters, 256
wsrep_restart_slave
    Parameters, 257
wsrep_retry_autocommit
    Parameters, 257
wsrep_rollbacker_thread_count
    Status Variables, 334
wsrep_slave_FK_checks
    Parameters, 258
wsrep_slave_threads
    Parameters, 259
wsrep_sst_auth
    Parameters, 260
wsrep_sst_donor
    Parameters, 65, 260
wsrep_sst_donor_rejects_queries
    Parameters, 261
wsrep_sst_method
    Parameters, 20, 262
wsrep_sst_receive_address
    Parameters, 263
wsrep_start_position
    Parameters, 264
wsrep_status_file
    Parameters, 264
wsrep_sync_server_uuid
    Parameters, 265
wsrep_sync_wait
    Parameters, 265
WSREP_SYNC_WAIT_UPTO_GTID()
    Functions, 270
wsrep_thread_count
    Status Variables, 334
wsrep_trx_fragment_size
    Parameters, 267
    Streaming Replication, 267
wsrep_trx_fragment_unit
    Parameters, 267
    Streaming Replication, 267