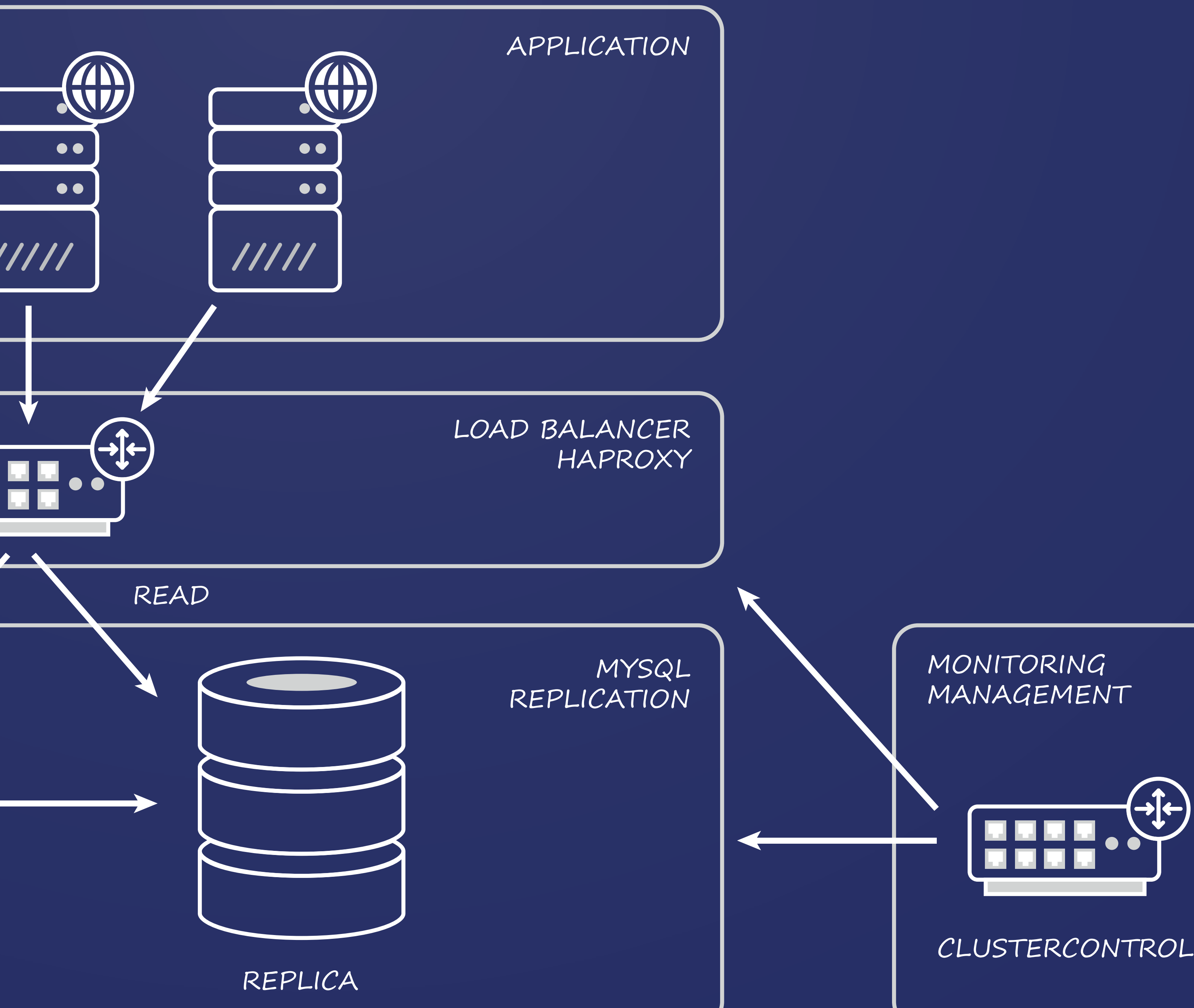
severalnines

MySQL Replication
Blueprint



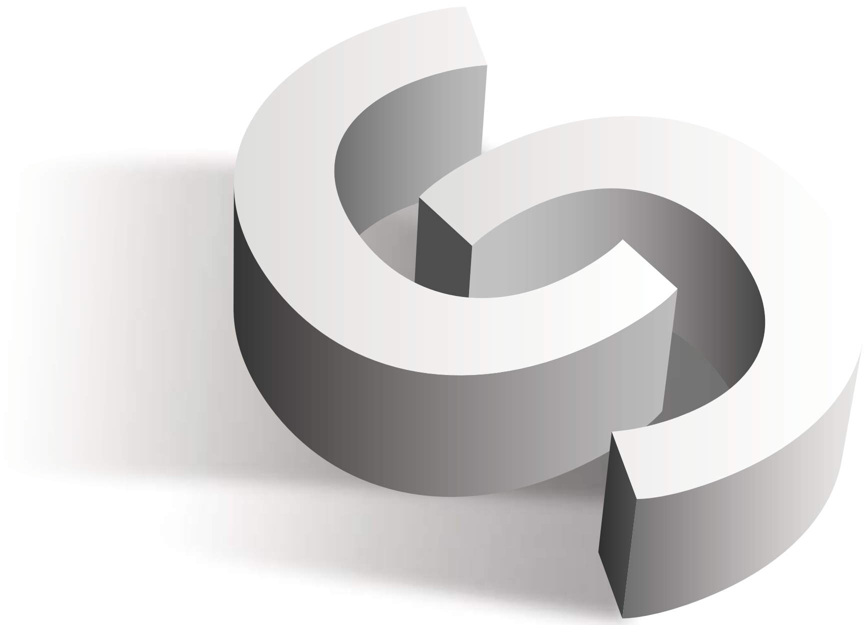


Table of Contents

Introduction

MySQL Replication has become an essential component of scale-out architectures in LAMP environments. When there is a necessity to scale out, MySQL offers a multitude of solutions, the most common being to add read replicas. The major bottleneck for our data is generally not so much oriented around writing our data but more around reading back this data. Therefore the easiest way to scale MySQL is to add replicas for reading.

The traditional master-slave solution comes with a major flaw: the master is a single point of failure. To overcome this, various solutions emerged that run on top of the MySQL replication topology and try to make it highly available. Tools like MySQL Multi Master (MMM), MySQL HA Master and Percona Replication Manager can manage your replication topology but they do have their cons: in general they are quite difficult to set up and in some cases create another single point of failure. Clustering software like Corosync is able to improve this a bit, but introduces even more complexity.

With today's cloud environments, where resources are dynamically allocated and deallocated, systems need the ability to automatically adapt to sudden changes. For MySQL Replication, this includes tasks like detecting failures, promoting a slave to master, failing over slaves, and so on.

A load balancer with Virtual IP can also help mask topology changes from the application, and dispatches read and write traffic appropriately.

Systems also undergo configuration changes or version upgrades. Management procedures for these need to be orchestrated so all the system components are in sync. As we're assembling a system from independent, standalone components, how do we know if something breaks? Therefore, monitoring would be an integral part of any system that goes in production.

Building a production-ready system around MySQL Replication is a major undertaking, but not an impossible one. Facebook, Twitter and Booking.com run thousands of masters and replication slaves in their data centers. These companies invested heavily in building tools to address these issues so they could run replication at scale. The purpose of the Blueprint is to provide an integrated framework for addressing the operational aspects of MySQL Replication.

Why a Blueprint for Replication?

2.1. Replication in the pre-MySQL 5.6 era

MySQL Replication has been around for a very long time. It was introduced 15 years ago, in MySQL 3.23. While MySQL itself went through massive improvements over the years, the replication functionality remained pretty much the same - single threaded, easy to break, hard to retain consistency or change topologies, etc. Although it was easy to set up, it has always been a challenge to support and maintain in production - not great in an age of cloud computing and dynamic infrastructures.

In the meanwhile to solve the problems around Replication, we saw the emergence of various tools that ran on top of the MySQL replication topology. To perform master failover or slave promotion, MySQL Multi Master (MMM), MySQL HA Master and Percona Replication Manager became available. These tools can manage your replication topology but they do have their pros and cons: in general they are not simple to set up, some consist of of chained components that can fail and slave promotion without the concept of a global transaction identifier remains fragile. Also these tools solely focus on retaining the replication topology with a master and do not address the other aspects of a replication topology like making backups or scaling out replicas.

With 5.6 and more recently 5.7, MySQL Replication has gone through a total transformation. Features like Global Transaction Identifiers make it easy to maintain consistency in dynamic changing replication topologies. Which begs the question - how relevant are these tools that were written to address issues that existed in the pre-MySQL 5.6 era?

2.2. Making MySQL Replication Production Ready

What makes a MySQL Replication setup production ready? Is it enough to enable the binlog on a master and have slaves read from it?

Unfortuntely, a production environment requires quite a bit more planning than that. How do we ensure our replication setup is working properly? How do we ensure slaves are in sync with the master? What do we do if replication is slow, or stalls? How do we handle the failure of a master or a slave server? How do we ensure applications write to the right master, and read from the right slave? What if we need to replace a server? How are configuration changes performed, and propagated across the replication chain? What about database upgrades and schema changes, can we do them without service downtime? How do we shield applications from all these changes? Do we need to modify our application so it is aware of which server to write to? What about adding capacity? Disaster recovery and data lost? The list goes on.

A holistic approach on replication means setting up a topology from end to end, with a clear understanding about the workings of the replication topology. Deployment of the nodes can be automated using Puppet, Chef or Ansible but this has the downside that these tools need to understand the workings of the topology as well. Which host would become the master, which host would become a replica? How do we ensure postdeployment configuration changes are applied across all servers, even if the topology has changed from its original form? In other words: your automated deployment has to intelligent, understand your replication topology inside out and keep itself up-to-date with any changes. Are these automation tools designed to be this intelligent?

Failover tools can promote one of the replicas to become the new master and generally use Virtual IP addresses to manage the change instantly. These failover tools often rely on third party tools like Corosync and Pacemaker which are not trivial to set up.

When Virtual IP addresses cannot be used, another way of "advertising" the new master is necessary. A popular choice is to use configuration managers, like Zookeeper and Consul, but these configuration managers need to be able to understand the topology and monitor it closely. Another way to handle this is by adding a load balancer between the application and the database.

Popular load balancers like HAProxy, MaxScale, ProxySQL need to be configured so they understand the topology. Configuration of these tools is usually done separately from your failover tool and this also requires double administration. Keep in mind that not every failover tool is a good combination with another load balancer: MaxScale monitors the master node by itself so it could be the case it reaches a different conclusion than MHA (MySQL Master HA) whether a node is up or down.

As we can see, there are quite a few components here - all of which require the right configuration to understand and maintain topology logic.

As those separate tools may have a different approaches and/or implementations, this could lead to a wrong synergy between the tools. Since they were not designed to work together, any topology change means your have to reconfigure all the various tooling.

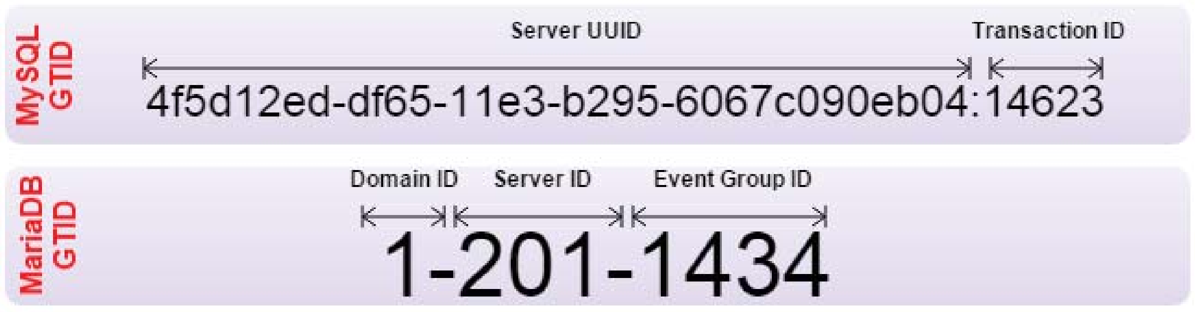
2.3. GTID - A Stronger Foundation for Replication

The Global Transaction Identifier (GTID) was introduced in MySQL 5.6 to identify and associate each transaction to its server of origin. This is important as within a replication topology, not only the transaction identifiers need to be unique per server but also for the whole topology.

Without GTID, a MySQL server would log its slave updates in its own binary log in the exact same order as they were logged on the master. As it writes to its own logs, it used a different numbering for the log position of each transaction. Thus it is very difficult to find the log position of the same transaction between two servers. An automated tool like MySQL HA Master (MHA) would address this type of problem, but without that, you generally need to scan through the binary logs to find the same position.

Why is this important? If the master would fail in a multi node replication topology, the most advanced slave is promoted to become the new master and the other slaves need to realign their replication with the new master. Thanks to the GTID, the new slaves can now fetch the binary logs from the new master and find the correct position by themselves. This retains data integrity after failover.

Keep in mind that the GTIDs between MySQL and MariaDB differ, they cannot be mixed in the same topology. In MySQL, the GTID is composed by using the server's UUID and appending to that a sequence number. In MariaDB, this is done differently by combining the domain, host and sequence number into the GTID. In both cases, this makes it a unique identifier within the replication topology.

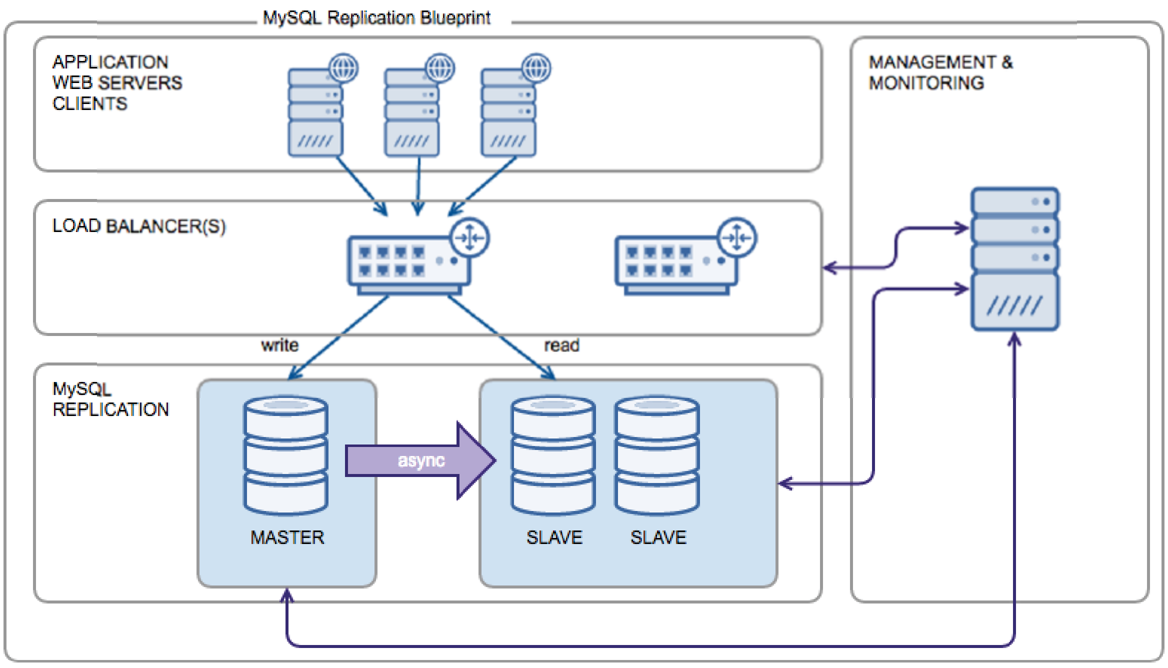


Changing from non-GTID to GTID requires reconfiguration of all the nodes in your topology. You can prepare all your replica nodes to enable GTID, however as the master node is the one that generates the transactions, the master should be reconfigured before the GTID becomes effective. This means you will certainly have downtime for all nodes.

Introducing the MySQL Replication Blueprint

The MySQL Replication Blueprint is about having a complete ops-ready solution from end to end including:

- installation and configuration of master/slave MySQL servers, load balancers, Virtual IP and failover rules
- Management of the topology, including failure detection, failover, repair and subsequent reconfiguration of components
- Managing topology changes when adding, removing or maintaining servers
- Managing configuration changes
- Backups
- Monitoring of all components from one single point



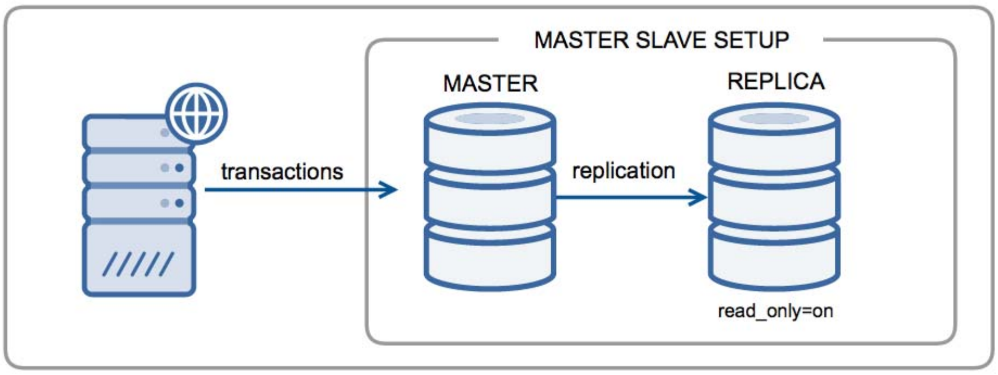
3.1. Deployment and Configuration

Deployment of the topology is your starting point where you deploy all the necessary nodes in your topology: the master, the slaves, the load balancers but also the hosts that will run the monitoring and management software.

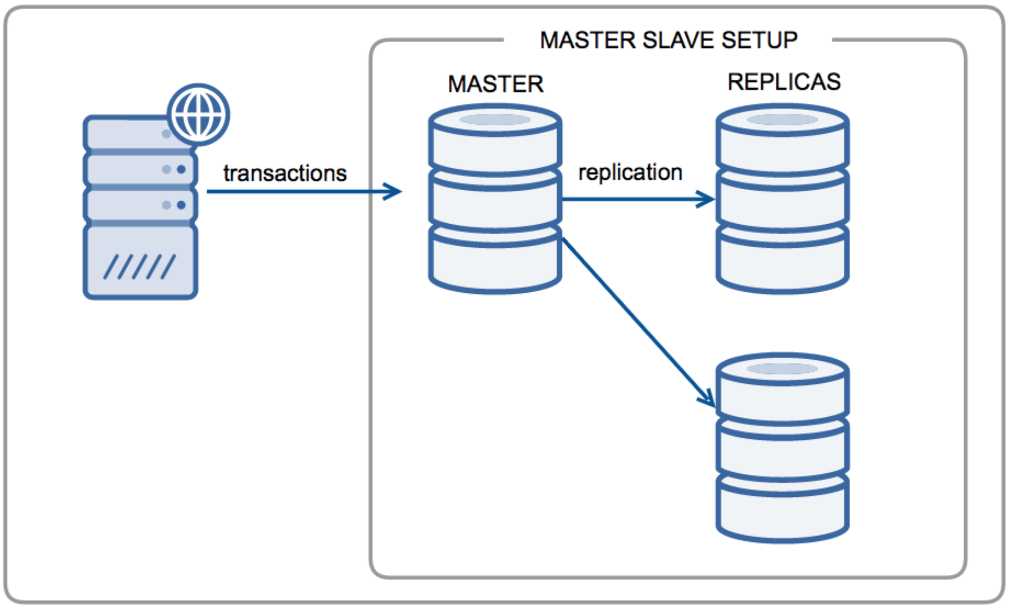
Preferably you would deploy your hosts in an automated way, for instance through Puppet, Chef or Ansible. These deployment tools can also set up replication correctly, but keep in mind that if you do this, the replication should only be set up once during the initial bootstrap of your replication topology: you don't want to have your configuration management tool stop and start replication everytime it changes a configuration value. Replication can be set up in many different ways.

3.2. Master/Slave

This is the most basic topology possible: one node acts as the master and another acts as the replica node (slave). The replica node will receive all transactions written on the master.



As shown in the diagram, the master node will receive all the write operations while both the master and replica(s) can be used to receive read operations. The strength in this topology lies in the fact that the read operations can be scaled by adding additional replicas of the master.

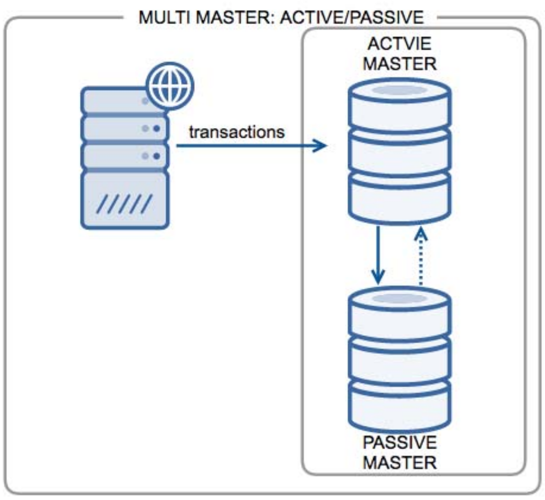


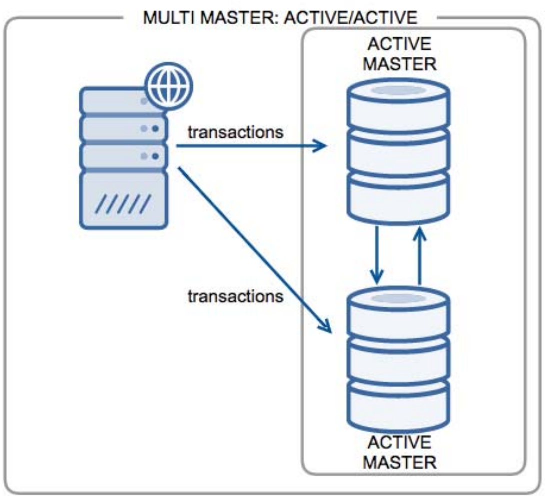
In theory you could perform write operations on the replica nodes. However as the replication stream goes from the master to the replica nodes, these writes will never be propagated back to the master node. You would get inconsistency of data between the master and replica nodes. Therefore these replica nodes should be set to read_only=ON in MySQL to ensure this can never happen.

3.3. Multi Master

Multi Master is similar to the Master/Slave topology, with the difference that both nodes are both master and replica at the same time. This means there will be circular replication between the nodes. It is advisable to configure both servers to log the transactions from the replication thread (log-slave-updates) but ignore its own already replicated transactions (set replicate-same-server-id to 0) to prevent infinite loops in the replication. This needs to be configured even with GTID enabled.

Multi master topologies can be configured to have either a so called active/passive setup where only one node is writable and the other node is a hot standby. Then there is the active/active setup where both nodes are writable.

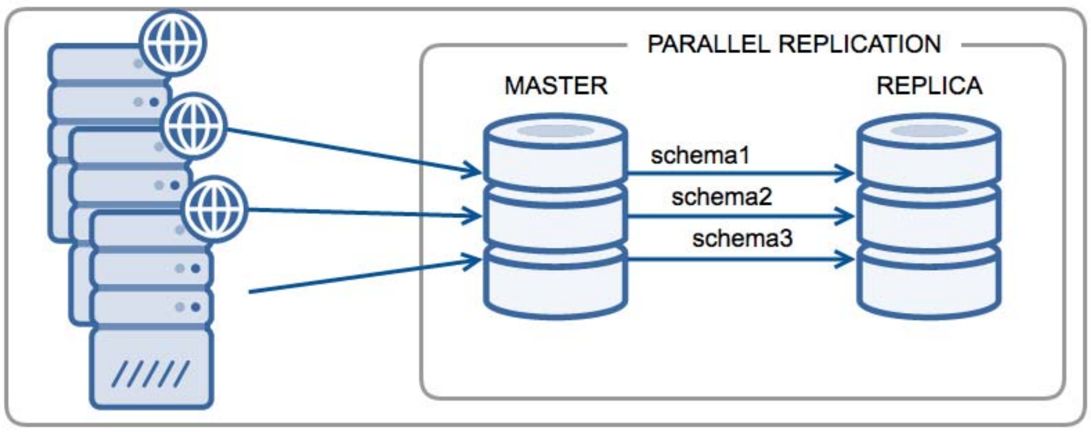




Caution is needed with active/active as both masters are writing at the same time and this could lead to conflicts if the same dataset is being written at the same time on both nodes. Generally this is handled on application level where the application is connecting to different schemas on the two hosts to prevent conflicts. Also as two nodes are writing data and replicating data at the same time they are limited in write capacity and the replication stream could become a bottleneck.

3.4. Parallel Replication

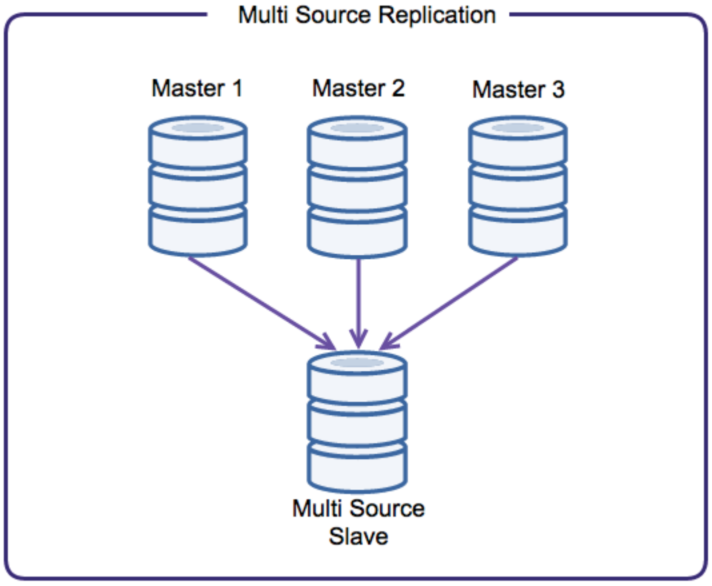
As MySQL replication is asynchronous, the single thread applying the replicated data can become the bottleneck, especially when DDL changes are sent via replication. DDL changes will block replication until the DDL change has been applied.On large tables, this could take a while.



In principle, the parallel replication, also known as multi-threaded slave, is similar to the Master/Slave or Multi-Master topology, but this tackles the problem of the single threaded applier. Parallel replication has been implemented in various way: replicate queries per schema in parallel (MySQL 5.6), tagging non conflicting writes (MariaDB 10.0), group committing large write sets (MariaDB 10.0 and MySQL 5.7). In theory this should allow you to increase the efficiency of replication.

3.5. Multi source Replication

Multi source replication is supported as of MariaDB 10.0 and MySQL 5.7 . Basically this means that a replica is allowed to replicate from multiple masters. To enable this, the replica should not have multiple masters writing to the same schema as this would lead to conflicts in the write set.



Multi source replication is currently not widely supported by replication tools. In general these tools use the output from SHOW SLAVE STATUS to determine the replication state of the replicas. With multi source replication, there are several replication channels and thereby multiple replication streams. Therefore it is not advised to use multi source replication in combination with a replication manager unless you have tested this thoroughly. The alternative is to make the slave an unmanaged replica.

Why would you need multi source replication? Multi source replication may be suitable for data warehousing needs, delayed slaves or data locality in some cases. Especially if multiple database clusters are not utilizing their full write capacity, it may save a few hosts by consolidating multiple slaves into one node.

3.6. Management & Monitoring

In addition to the initial replication setup, one might need a replication manager to both monitor the replication stream and react upon any issues if necessary. Why would you need to manage replication unless it breaks? There are two reasons for this: you may want to ensure slaves are not lagging behind the master. In cases of maintenance, you would need a tool that allows you to perform a master failover in a controlled manner. In cases of failure, you might want some automatic failover functionality.

A good replication manager should be able to monitor both the master and replicas. It should only failover when necessary and should have protection against flapping (failover consecutively). There are various replication managers that will suit your needs, but in our experience MySQL Master HA (MHA) is one of the most used replication managers which also supports the MySQL GTID.

3.7. Load balancers

As described earlier, working with configuration manager for your application requires additional logic to reconfigure your application whenever necessary. If you already have a configuration manager like Zookeeper or Consul in your environment, it would be good to use them Otherwise the extra work to install and set up these tools might less sense. Instead, you might want to consider using a load balancer.

For load balancing of MySQL, the most used solution is HAProxy. HAProxy is a layer 4 proxy, meaning it will only route on basis of TCP and UDP traffic. This makes it an extremely simple and stable proxy. MaxScale and ProxySQL both operate on layer 7, meaning they will understand the queries that pass by. MaxScale and ProxySQL are relatively new, but have the added benefit of allowing to split reads and writes separately from existing mixed connections. In all cases, you can use the load balancer to understand the replication topology and route the traffic to the master: HAProxy will need a few more snippets to implement than ProxySQL and MaxScale, which do that out of the box.

By introducing load balancers you will make MySQL more highly available as you can now transparently promote a slave to become the new master, or in case of a failure, act accordingly. However by introducing a load balancer you will introduce another single point of failure (SPOF), so you need to add a second load balancer to take over if the first one fails. This can be done in a much more simplistic fashion than with a MySQL failover: if the load balancer fails on TCP level, you need to failover to the second load balancer, or use Virtual IP address if you can use them in your environment. Keepalived can manage this type of failover and add high availability to the load balancing layer.

Monitoring

Monitoring is about keeping an eye on the database servers and alerting if something is not right, e.g., a database is offline or the number of connections crossed some defined threshold.

The components in the replication topology would do some kind of monitoring. This includes the replication manager, the load balancers and the slaves 'monitoring the master. But does this give us visibility into our entire setup? What are the requirements for monitoring?

4.1. Availability

To check the availability of MySQL, it is not enough to open a connection and perform a query like "SELECT 1". A host may allow new incoming connections and perform simple queries, but a SELECT 1 does not test anything important at all. You need to know the true status of the node, so for availability you would like to know if the essential schemas are available, if you can read and/or write to one of these schemas and if the node is still part of the replication topology.

Availability monitoring for MySQL is offered in MonYog, Percona Monitoring Plugins (Cacti & Nagios) and Zabbix.

4.2. Performance

Performance monitoring is essential for you to have insights in the health of your systems. Simply relying on availability monitoring is like sailing in the dark and hoping there will be no storm. Performance monitoring can give you insights in the capacity of your systems and how much of the resources are still available. It also helps predict when you need to scale up or out by plotting trends on for instance IO capacity/IOPS, CPU and memory usage.

With respect to replication, it is essential to monitor the slave lag as lagging slaves will be removed from the replication topology by most failover tools. Also if you end up with lagging slaves, the failover tool has to wait until the slave has caught up and applied all pending transactions and this slows down the failover process.

Trending is essential for performance monitoring so make sure your monitoring software keeps historical data. For some of the standard monitoring suites, the granularity/sampling of the data is really low so the insights you will get from that can be coarse. For instance an average of the connections made in the past 5 minutes will not give you any insights on why you are hitting max_connections every now and then. Therefore you can also use additional software that is more suitable for keeping fine grained data points like Grafana, Prometheus and Influx DB or a hosted solution like VividCortex.

4.3. Alerting

Obviously it will be a pain to constantly keep an eye on all the graphs and overviews, therefore alerting should be in place to draw attention to (potential) problems and outages. Most monitoring systems have a way to push messages to alerting services like PagerDuty, OpsGenie and VictorOps. Alternatively you could also have these services connect to (custom) APIs.

The management of a replication topology covers a wide range of tasks, and it is very unlikely that you will find one tool to handle everything. Therefore it is very common for ops teams to build their own custom solutions from readily available tools and utilities.

Management

5.1. Replication topology changes

Database setups are never static. You might need to add more nodes for scaling purposes. You might need to perform maintenance on your servers, and reconfigure the topology while this is ongoing. At some point, it might be necessary to migrate to new servers or a new data center. And in case the unexpected happens, e.g. a replication master fails, you will need to promote a new master and subsequently reconfigure slaves and load balancers.

5.2. Adding new slaves

Adding new slaves in your replication topology means you are scaling out your cluster. In the deployment chapter, we described that all you have to do is point a freshly installed replica to the first item in the master's binary logs. This should create an identical copy. But what if you already have a MySQL node with a large dataset running?

This can be done by priming the replica with a fresh copy of the master data. Creating a consistent dataset from the master can be done using Xtrabackup where it is also capable of registering the current log position of the master. This is extremely useful for setting up new replicas.

The difference between deployment and adding a slave comes when you need to add the new slave to the topology and make it available for use by your applications. Information about the topology change should be propagated throughout the topology (database instances, monitoring, failover tools and load balancers).

Scaling out replicas does have a side effect: every node replicating from the master will create a replication thread on the master. Creating many (50+) replication threads will create significant load on the master, as it will be busy with sending the right data from the binary logs to its replicas.

5.2.1. Why would you delay a slave?

We mentioned the Delayed Slave briefly as one of the reasons you might need multi source replication. The delayed slave is invaluable for those who need quick access to their data as it was one hour or maybe 24 hours before. The reason can vary between a "wrong code deployment that ate all your data" and "hackers wiped all our sales data". In MySQL 5.6, you can start delayed replication using the change master to syntax with the addition of the delay (1 hour in this example):

```
CHANGE MASTER TO …
MASTER DELAY = 3600;
```

For older versions and MariaDB you can use pt-slave-delay as an alternative.

Due to the delayed slave, you can quickly recover your data right up to the moment when it happened:

1. Stop the delayed slave from replicating

```
STOP SLAVE;
```

2. Find the binary log entry on the master that wipes the data

```
mysqlbinlog --start-datetime=”2016-03-09 08:15:00”
--stop-datetime=”2016-03-09 13:12:00” /var/lib/mysql/
bin_master.000036 | grep -i -B 25 “DROP TABLE orders”
```

3. Start the delayed slave and let it catch up with the master up till the wrong log entry

```
START SLAVE UNTIL MASTER_LOG_FILE = ‘bin_mas-
ter.000036’, MASTER_LOG_POS = 46626626778;
```

4. Promote slave to master

These four simple steps should get you back in business in no-time. This obviously still does not resolve the initial reason why your data got lost or corrupted.

5.3. Repairing a broken replication topology

MySQL Replication can be fragile: whenever it encounters a connectivity error, it will retry and if it is a serious error it will simply stop. Obviously in the latter case, you will need to repair the broken replication yourself.

Most common problems with replication are when replication stops due to either master failure or network problems. In those cases, promoting a slave to become a new master will resolve the problem. This failover scenario will be discussed below.

Also the replication can break due to data inconsistency. Data inconsistency can happen due to data drift, but since the arrival of row based replication (RBR) this happens less frequently. The best way to resolve this is by providing a fresh copy of the master’s data to the replica and redefining the replication stream.

5.3.1. Slave promotion

In case the master fails the whole topology becomes read-only and this means the write queries can’t be applied anymore. This is where normally you would promote one of the replicas to become the new master. To illustrate the difference in promotion between GTID and non-GTID cases we will go through the manual promotion below.

5.3.2. Most advanced slave without GTID

The first step in this promotion is to find the most advanced slave. As the master is no longer available, not all replicas may have copied and applied the same amount of transactions, so it is key to find the most advanced slave first.

We first iterate through all replicas to see which one is the furthest in the last binary log and elect this host to become the new master.

```
SHOW SLAVE STATUS\G
************************ 1. row ****************************
            Slave_IO_State: Waiting for master to send event
               Master_Host: 10.10.12.11
               Master_User: repluser
               Master_Port: 3306
             Connect_Retry: 60
           Master_Log_File: binlog.000003
       Read_Master_Log_Pos: 1447420
...
       Exec_Master_Log_Pos: 1447420
```

Then the next step is to advance the other replicas to the latest transactions on the candidate master. As the replicas are logging their slave updates in their own binary logs they have a different numbering for their own transactions and thus it is very difficult to match this data. An automated tool like MySQL HA Master (MHA) is capable of doing this, so when you are failing over by hand you generally are scanning through the binary logs or skipping these transactions.

Once we have done this we tell the replicas to start replicating from the designated points of the new master.

```
CHANGE MASTER TO
  MASTER_HOST = ‘new.master’,
  MASTER_PORT = 3306,
  MASTER_USER = ‘repl’,
  MASTER_PASSWORD = ‘repl’,
  MASTER_LOG_FILE = ‘binlog.000002’,
  MASTER_LOG_POS = 1446089;
```

5.3.3. Most advanced slave with GTID

By far the greatest benefit of using GTIDs within replication is that within the replication topology, all we have to do is to find the most advanced slave, promote it to master, and point the others to this new master.

So the most advanced slave is the same as without GTID:

```
************************ 1. row ****************************
            Slave_IO_State: Waiting for master to send event
               Master_Host: 10.10.12.14
               Master_User: repl
               Master_Port: 3306
             Connect_Retry: 60
           Master_Log_File: binlog.000003
       Read_Master_Log_Pos: 1590
...
       Exec_Master_Log_Pos: 1590
```

Finding out which part of the binary logs the other hosts are missing is not necessary, as the new master's binary logs already contain transactions with the GTIDs of the dead master and thus the slaves can realign with the new master automatically. This applies to both MariaDB and MySQL implementations of GTID.

```
CHANGE MASTER TO
  MASTER_HOST = new.master',
  MASTER_PORT = 3306,
  MASTER_USER = 'repl',
  MASTER_PASSWORD = 'repl',
  MASTER_AUTO_POSITION = 1;
```

As you can see this is a far more reliable way of promoting a slave to a master without the chance of loss of transactions. Therefore GTID failover is the preferred way in ClusterControl.

5.3.4. Automated slave promotion

You can use automated slave promotion software like MySQL MasterHA (MHA), MariaDB Replication Manager with MaxScale (MariaDB only), Percona Replication Manager (PRM) or Orchestrator.

5.4. Backups

Making backups is useful for disaster recovery or providing copies of your production data for development and/or testing. Backups for MySQL can be made through creating either a logical backup or a physical backup. The difference between the two is that a logical backup is a dump of all records in the database while the physical backup is done by copying the files from the MySQL data directory. Logical backups can be made using mysqlbackup and physical backups using Xtrabackup or filesystem snapshots.

Always create the backup on the node with the least impact, so if possible always make a backup on a replica and preferably not on a master node. Even though snapshots and Xtrabackup have relatively low overhead, it may very well be that another (analytics) job gets scheduled at the same time and the combined stress can be the tipping point.

5.4.1. Logical or physical backups?

The logical backups generally take more time to make as while scanning through the data the queries also have to be constructed. Also for MyISAM tables the logical backup will put a lock on the tables while dumping the data to ensure a consistent copy is made.

Filesystem snapshots may seem to be favorable due to their low impact on the system and them being a snapshot of the filesystem. However they take more time to recover if you only need to recover single tables or single rows.

5.4.2. Do you need full or incremental backups?

Mysqldump generally only allows you to create a full copy of your data while Xtrabackup allows you to make incremental backups since last backup. You do need to have an initial full backup available before you can actually make incremental backups. Xtrabackup will then only copy the altered data since the last (full) backup. If you have large backups to make, this can reduce the size of your backups.

Keep in mind that incremental backups will not decrease the time it takes to make the backups: Xtrabackup still needs to scan through the InnoDB files of all schemas and tables to find the altered data. Also restoring an incremental backup can take more time than restoring a full backup so if restore time is your biggest concern: stick to full backups.

5.4.3. Scheduling

Schedule your backups always during off-peak to ensure the least number of users will be impacted. Scheduling tools/backup suites are available that can handle the scheduling for you and also report if anything went wrong. Good choices are MySQL Enterprise Backup, Bacula, Zmanda Recovery Manager (ZRM) , Holland Backup and ClusterControl.

5.4.4. Testing your backups

You can make all the backups you like, but they are worthless if they have not been tested. It is essential to regularly test/check your backups. These tests can range from simply decompressing the archive files, starting MySQL using the data from the backup to rebuilding a slave and adding it back to the replication topology.

5.5. Updating to a newer version

Updating to newer major versions can be tricky. New features like the introduction of GTID after updating from 5.5 to 5.6 requires a full restart of the replication topology, meaning all nodes in the topology should be down at the same time. Therefore always read and test the upgrade procedure up front.

Updating to minor versions should be less of an issue as they mostly introduce bugfixes. Updating does not require you to bring down the whole toplogy at the same time: you can perform a rolling restart. In a rolling restart you will restart host by host until you have restarted the whole topology. It is advised to stay up to date with the latest releases as they also include fixes for security issues like Heartbleed and DROWN.

5.6. Schema changes

Applying schema changes in a MySQL replication setup can be tricky: DDL changes will be propagated via the replication to the slaves. With a bit of bad luck, all slaves will be busy applying the same change to that 800GB table at the same time. As this operation will take a while, the replication starts to lag behind and the failover software and load balancers remove all read slaves one by one. The scenario described is not an unlikely scenario.

A solution is available through the Percona Online Schema Change (pt-osc) that creates a new table with the new structure next to the existing table, copies the data into the new structure and creates triggers on the existing table to backfill the data to the new table. Everything is replicated to the slaves and as the data is copied from one table to another, it will be sent through row updates and thus the slaves will most likely not lag behind in replication. The Percona Online Schema Change is not fully compatible with every DDL change, so always test your schema changes prior to applying them on your production environment!

5.7. Configuration changes

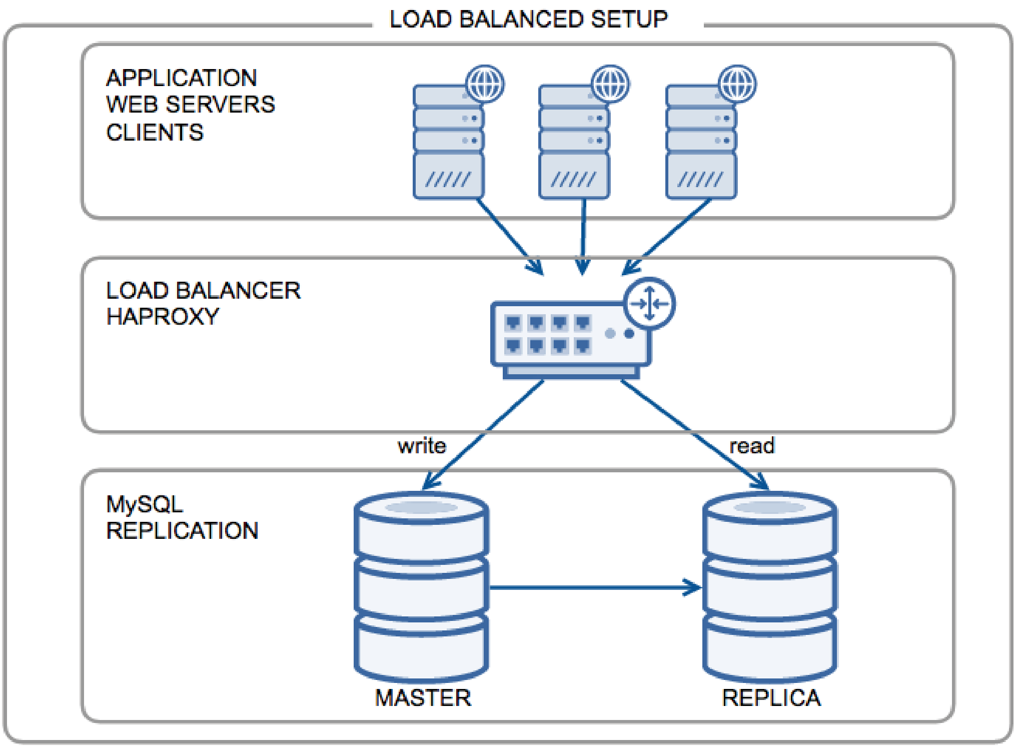
Configuration management ties into your deployment mechanism. If you are using for instance Puppet or Chef, you need to apply your changes in these systems as well. If you're not using any configuration management system, it is advised to keep a copy of your configuration files inside a (git) repository. Should you make a configuration error, you can easily revert back to the previous version.

Also if you are applying configuration changes, keep in mind that some parameters can be changed at runtime. You have to make sure that both the configuration and runtime parameters are in sync. If not, there might be an unhappy surprise waiting after the next restart of MySQL. Some monitoring systems, like MonYog, are able to detect these anomalies and bring them to your attention.

Load Balancing

To ensure any component in our topology can fail, it has to be a highly available setup with no single point of failure. The most important component in the MySQL Replication topology is the master node, as this is a single component where all write operations end up. As we showed earlier in this paper, it is possible to promote a slave to master. So, losing a master is not disastrous. However, slave promotion will not automatically point your application to the correct host to write to.

There is a wide range of methods available to perform an automated failover from an application perspective. Implementations range from service discovery tools like Zookeeper and Consul to reconfigure your applications, to Virtual IP addresses that will perform the change on network level. Service discovery tools have the downside that they rely on other tools that understand the topology and make the configuration change. On the other hand, the use of Virtual IP addresses isn't always available or allowed in (cloud) hosting environments.

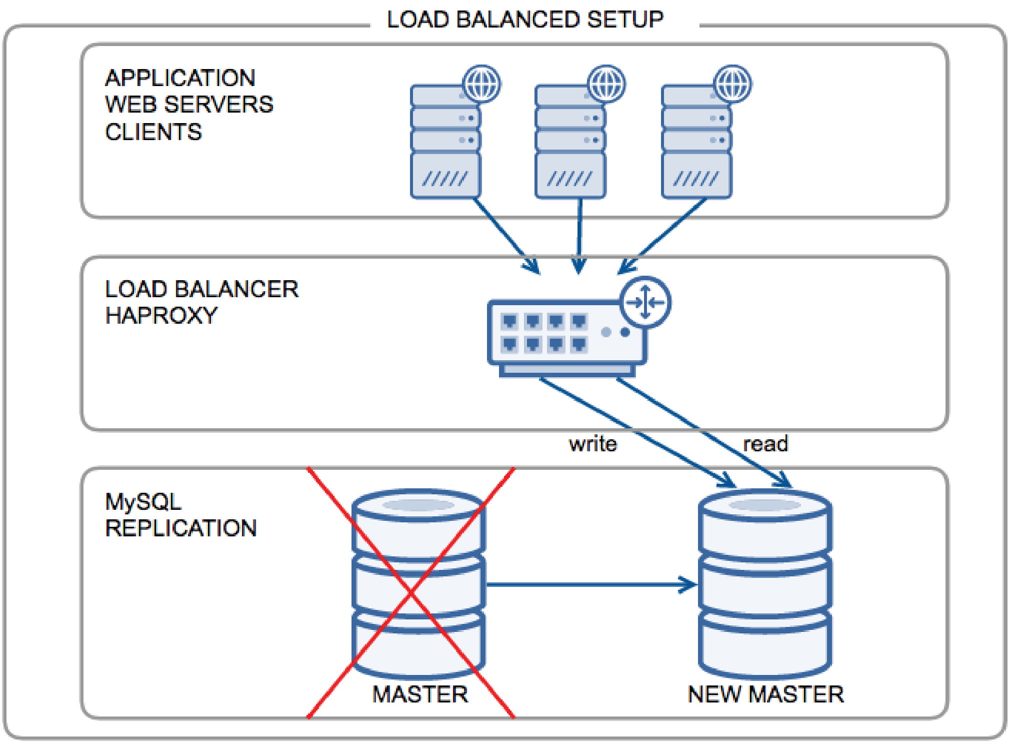


At this moment the most frequent deployed solutions are proxies. Popular proxy solutions are HAProxy, MySQL Proxy, MaxScale and ProxySQL.

6.1. What are the benefits of proxies?

A proxy will sit in between the application and the database nodes and route the traffic transparently to the correct node in the topology. As described in the previous paragraph, it allows you to split your database traffic into a read and write stream, and ensures they only arrive on the master or the replicas.

The proxy needs to be aware of the replication topology. This is defined in the configuration of the proxy. In case of change of master, a manual reconfiguration is necessary for HAProxy. MaxScale and ProxySQL can automatically reconfigure themselves.



6.2. Read/Write splitting

In general if you wish to scale out MySQL Replication topologies, you'll want to send all the write operations to the master and all read operations to the slaves. This means you will have to somehow handle connections accordingly: ensure you only connect to the master if you intend to write data.

The easiest solution for this is by adding the read/write split to your application logic. Only when your application is going to write data, it will create a (second) connection to the master and write the data. If your application is going to read data, it will connect to a (random) slave in the topology.

To make this solution easier you could add two ports on a proxy that handle either read or write traffic. The read traffic will be automatically load balanced over all available read slaves while the write traffic will be assigned to the current master. In case of a master failover or a slave promotion, the load balancer will be updated accordingly. Some of the proxies, ProxySQL and MaxScale, are more intelligent and can connect to the read slave by default, detect write operations within a transaction and switch over to the master connection to write this data.

6.3. Which proxy to choose?

At the moment of writing, ProxySQL is a relatively new player on the market and is not yet supported in ClusterControl. MaxScale offers the ability to do read/write splitting, but this comes at the price of overhead and memory consumption at high loads. This makes it suitable for smaller scale topologies and act as a drop in replacement for a single master topology being scaled out with replicas. HAProxy is a solid and proven technology and we would recommend it if you are planning for large scale operations.

6.4. Query Caching

ProxySQL is able to perform query caching at the proxy level. In the past 10 years, query caching in MySQL is considered to be bad, but query caching in itself isn't. MySQL query caching is a shared memory space that all connections make use of, this means any query performing an operation on the query cache will put a lock on it to ensure consistency. For read operations, this takes maybe a few microseconds but for write operations it may take a bit longer. Even worse: if a write operation happens the query cache is invalidated for all entries that contain the same table. Cache evictions do take quite a lot of time and in high concurrency environment, this could be your biggest bottleneck.

Still the fastest query on MySQL is the query that never got executed. So if you can prevent a query from being executed , e.g. using a proxy, it would improve performance dramatically. Also since the roundtrip time is a lot shorter, as the result is instantly returned from the proxy. ProxySQL is only caching queries that are defined via a regular expression for a certain amount of milliseconds, so it works similar as you would cache results via Memcache or Redis. It also doesn't have a locking mechanism for the cache and thus it can never be a bottleneck.

6.5. Query rewrites

ProxySQL and MaxScale both offer query rewriting through regular expressions. This is a very powerful feature as it allows you to change queries while they are being sent through the proxy. This could help you to survive through the weekend by "correcting" a faulty query and then have it properly fixed on Monday. Or you like to add a "force index" to a query once the optimizer of MySQL has decided it really needs to use a different index. Another example would be rewriting a query where a hacker performed an SQL injection and negate the attack this way. You could also shard your data over multiple clusters and use the query rewriter to send the queries to the correct shard cluster.

About Severalnines

Severalnines provides automation and management software for database clusters. We help companies deploy their databases in any environment, and manage all operational aspects to achieve high-scale availability.

Severalnines' products are used by developers and administrators of all skills levels to provide the full 'deploy, manage, monitor, scale' database cycle, thus freeing them from the complexity and learning curves that are typically associated with highly available database clusters. The company has enabled over 8,000 deployments to date via its popular ClusterControl solution. Currently counting BT, Orange, Cisco, CNRS, Technicolour, AVG, Ping Identity and Paytrail as customers. Severalnines is a private company headquartered in Stockholm, Sweden with offices in Singapore and Tokyo, Japan. To see who is using Severalnines today visit, http://severalnines.com/customers.

Deploy Manage Monitor Scale

Related Resources from Severalnines

Whitepapers

Tutorial

MySQL Replication for High Availability

severalnines

MySQL Replication for High Availability

This tutorial covers information about MySQL Replication, with information about the latest features introduced in 5.6 and 5.7. There is also a more hands-on, practical section on how to quickly deploy and manage a replication setup using ClusterControl.

Download here

severalnines

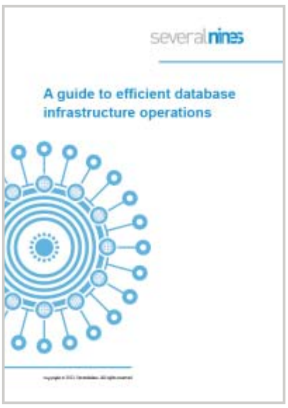
Management and Automation of Open Source Databases

A Severalnines White Paper

Management and Automation of Open Source Databases

Proprietary databases have been around for decades with a rich third party ecosystem of management tools. But what about open source databases? This whitepaper discusses the various aspects of open source database automation and management as well as the tools available to efficiently run them.

Download here


severalnines

A guide to efficient database infrastructure operations

A Guide to Efficient Database Infrastructure Operations

Taking control of their data is every company's number one job.

Database operations encompass a number of functions, including the initial deployment of a solution, configuration management, performance monitoring, SLA management, backups, patches, version upgrades and scaling.

Download here

severalnines

Deploy

Manage

Monitor

Scale