



# MaxScale

## Configuration & Usage Scenarios

Mark Riddoch

Last Updated: 8<sup>th</sup> May 2014

# Contents

[Contents](#)

[Document History](#)

[Introduction](#)

[Terms](#)

[Configuration](#)

[Global Settings](#)

[Threads](#)

[Service](#)

[Router](#)

[Servers](#)

[User](#)

[Passwd](#)

[Server](#)

[Address](#)

[Port](#)

[Protocol](#)

[Monitoruser](#)

[MonitorPw](#)

[Listener](#)

[Service](#)

[Protocol](#)

[Address](#)

[Port](#)

[Socket](#)

[Monitor](#)

[Module](#)

[Servers](#)

[User](#)

[Passwd](#)

[Protocol Modules](#)

[MySQLClient](#)

[MySQLBackend](#)

[Telnetd](#)

[HTTPD](#)

[Router Modules](#)

[Connection Based Routing](#)

[Statement Based Routing](#)

[Available Routing Modules](#)

[Readconnroute](#)

[Master/Slave Replication Setup](#)

[Galera Cluster Configuration](#)

[Readwritesplit](#)

[Master/Slave Replication Setup](#)

[Debugcli](#)

[Debug CLI Configuration](#)

[Monitor Modules](#)

[Mysqmon](#)

[Galeramon](#)

[Encrypting Passwords](#)

[Creating Encrypted Passwords](#)

[Configuration Updates](#)

[Limitations](#)

[Authentication](#)

[Error Reporting](#)

## Document History

Date	Change	Who
21st July 2013	Initial version	Mark Riddoch
23rd July 2013	Addition of default user and password for a monitor and discussion of monitor user requirements New monitor documented for Galera clusters Addition of example Galera cluster configuration	Mark Riddoch
13th November 2013	state for Galera Monitor is “synced”	Massimiliano Pinto
2nd December 2013	Updated the description of the command line arguments to match the code updates. Improved descriptions and general documentation. Enhanced example configurations	Mark Riddoch
6th February 2014	Added “enable_root_user” as a service parameter	Massimiliano Pinto
7th February 2014	Addition of bind address information Clarification of user configuration required for monitoring users and the user needed to fetch the user data	Mark Riddoch
3rd March 2014	MySQL authentication with hostnames	Massimiliano Pinto
3rd March 2014	Addition of section that describes authentication requirements and the rules for creating user credentials	Mark Riddoch
28th March 2014	Unix socket support	Massimiliano Pinto
8th May 2014	Added “version_string” parameter in service	Massimiliano Pinto

# Introduction

The purpose of this document is to describe how to configure MaxScale and to discuss some possible usage scenarios for MaxScale. MaxScale is designed with flexibility in mind, and consists of an event processing core with various support functions and plugin modules that tailor the behaviour of the MaxScale itself.

## Terms

Term	Description
service	A service represents a set of databases with a specific access mechanism that is offered to clients of MaxScale. The access mechanism defines the algorithm that MaxScale will use to direct particular requests to the individual databases.
server	A server represents an individual database server to which a client can be connected via MaxScale.
router	A router is a module within MaxScale that will route client requests to the various database servers which MaxScale provides a service interface to.
connection routing	Connection routing is a method of handling requests in which MaxScale will accept connections from a client and route data on that connection to a single database using a single connection. Connection based routing will not examine individual requests on a connection and it will not move that connection once it is established.
statement routing	Statement routing is a method of handling requests in which each request within a connection will be handled individually. Requests may be sent to one or more servers and connections may be dynamically added or removed from the session.
protocol	A protocol is a module of software that is used to communicate with another software entity within the system. MaxScale supports the dynamic loading of protocol modules to allow for increased flexibility.
module	A module is a separate code entity that may be loaded dynamically into MaxScale to increase the available functionality. Modules are implemented as run-time loadable shared objects.
monitor	A monitor is a module that can be executed within MaxScale to monitor the state of a set of database. The use of an internal monitor is optional, monitoring may be performed externally to MaxScale.

listener	A listener is the network endpoint that is used to listen for connections to MaxScale from the client applications. A listener is associated to a single service, however a service may have many listeners.
connection failover	When a connection currently being used between MaxScale and the database server fails a replacement will be automatically created to another server by MaxScale without client intervention
backend database	A term used to refer to a database that sits behind MaxScale and is accessed by applications via MaxScale.

# Configuration

The MaxScale configuration is read from a file which can be located in a number of places, MaxScale will search for the configuration file in a number of locations.

1. If the environment variable MAXSCALE\_HOME is set then MaxScale will look for a configuration file called MaxScale.cnf in the directory \$MAXSCALE\_HOME/etc
2. If MAXSCALE\_HOME is not set or the configuration file is not in the location above MaxScale will look for a file in /etc/MaxScale.cnf

Alternatively MaxScale can be started with the -c flag and the path of the MaxScale home directory tree.

An explicit path to a configuration file can be passed by using the -f option to MaxScale.

The configuration file itself is based on the “ini” file format and consists of various sections that are used to build the configuration, these sections define services, servers, listeners, monitors and global settings.

## Global Settings

The global settings, in a section named [MaxScale], allow various parameters that affect MaxScale as a whole to be tuned. Currently the only setting that is supported is the number of threads to use to handle the network traffic. MaxScale will also accept the section name of [gateway] for global settings. This is for backward compatibility with versions prior to the naming of MaxScale.

## Threads

To control the number of threads that poll for network traffic set the parameter threads to a number. It is recommended that you start with a single thread and add more as you find the performance is not satisfactory. MaxScale is implemented to be very thread efficient, so a small number of threads is usually adequate to support reasonably heavy workloads. Adding more threads may not improve performance and can consume resources needlessly.

```
# Valid options are:
#      threads=<number of epoll threads>
[MaxScale]
threads=1
```

It should be noted that additional threads will be created to execute other internal services within MaxScale, this setting is merely used to configure the number of threads that will be used to manage the user connections.

## Service

A service represents the database service that MaxScale offers to the clients. In general a service consists of a set of backend database servers and a routing algorithm that

determines how MaxScale decides to send statements or route connections to those backend servers.

A service may be considered as a virtual database server that MaxScale makes available to its clients.

Several different services may be defined using the same set of backend servers. For example a connection based routing service might be used by clients that already performed internal read/write splitting, whilst a different statement based router may be used by clients that are not written with this functionality in place. Both sets of applications could access the same data in the same databases.

A service is identified by a service name, which is the name of the configuration file section and a type parameter of service

```
[Test Service]
type=service
```

In order for MaxScale to forward any requests it must have at least one service defined within the configuration file. The definition of a service alone is not enough to allow MaxScale to forward requests however, the service is merely present to link together the other configuration elements.

## Router

The router parameter of a service defines the name of the router module that will be used to implement the routing algorithm between the client of MaxScale and the backend databases. Additionally routers may also be passed a comma separated list of options that are used to control the behaviour of the routing algorithm. The two parameters that control the routing choice are `router` and `router_options`. The router options are specific to a particular router and are used to modify the behaviour of the router. The read connection router can be passed options of master, slave or synced, an example of configuring a service to use this router and limiting the choice of servers to those in slave state would be as follows.

```
router=readconroute
router_options=slave
```

To change the router to connect on to servers in the master state as well as slave servers the router options can be modified to include the master state.

```
router=radconroute
router_options=master,slave
```

A more complete description of router options and what is available for a given router is included with the documentation of the router itself.

## Servers

The servers parameter in a service definition provides a comma separated list of the



backend servers that comprise the service. The server names are those used in the name section of a block with a type parameter of server (see below).

```
servers=server1,server2,server3
```

## User

The `user` parameter, along with the `passwd` parameter are used to define the credentials used to connect to the backend servers to extract the list of database users from the backend database that is used for the client authentication.

```
user=maxscale  
passwd=maxpassword
```

Authentication of incoming connections is performed by MaxScale itself rather than by the database server to which the client is connected. The client will authenticate itself with MaxScale, using the username, hostname and password information that MaxScale has extracted from the backend database servers. A detailed discussion of how this impacts the authentication process please see the “Authentication” section below.

The host matching criteria is restricted to IPv4, IPv6 will be added in a future release.

Existing user configuration in the backend databases must be checked and may be updated before successful MaxScale authentication:

In order for MaxScale to obtain all the data it must be given a username it can use to connect to the database and retrieve that data. This is the parameter that gives MaxScale the username to use for this purpose.

The account used must be able to select from the `mysql.user` table, the following is an example showing how to create this user.

```
MariaDB [mysql]> create user 'maxscale'@'maxscalehost'  
identified by 'Mhu87p2D';  
Query OK, 0 rows affected (0.01 sec)  
  
MariaDB [mysql]> grant SELECT on mysql.user to  
'maxscalemon'@'maxscalehost';  
Query OK, 0 rows affected (0.00 sec)
```

## Passwd

The `auth` parameter provides the password information for the above user and may be either a plain text password or it may be an encrypted password. See the section on encrypting passwords for use in the `MaxScale.cnf` file. This user must be capable of connecting to the backend database and executing the SQL statement “SELECT user, host, password FROM `mysql.user`”.

### **enable\_root\_user**

This parameter controls the ability of the root user to connect to MaxScale and hence onwards to the backend servers via MaxScale.

The default value is 0, disabling the ability of the root user to connect to MaxScale.

Example for enabling root user:

```
enable_root_user=1
```

### **version\_string**

This parameter sets a custom version string that is sent in the MySQL Handshake from MaxScale to clients.

Example:

```
version_string=5.5.37-MariaDB-RWsplit
```

If not set the default value is the server version of the embedded MySQL/MariaDB library.

Example: 5.5.35-MariaDB

## **Server**

Server sections are used to define the backend database servers that can be formed into a service. A server may be a member of one or more services within MaxScale. Servers are identified by a server name which is the section name in the configuration file. Servers have a type parameter of server, plus address port and protocol parameters.

```
[server1]
type=server
address=127.0.0.1
port=3000
protocol=MySQLBackend
```

### **Address**

The IP address or hostname of the machine running the database server that is being defined. MaxScale will use this address to connect to the backend database server.

### **Port**

The port on which the database listens for incoming connections. MaxScale will use this port to connect to the database server.

### **Protocol**

The name for the protocol module to use to connect MaxScale to the database. Currently only one backend protocol is supported, the MySQLBackend module.

### **Monitoruser**

The monitor has a username and password that is used to connect to all servers for

monitoring purposes, this may be overridden by supplying a monitoruser statement for each individual server

```
monitoruser=mymonitoruser
```

### **MonitorPw**

The monitor has a username and password that is used to connect to all servers for monitoring purposes, this may be overridden by supplying a monpasswd statement for the individual servers

```
monitorpw=mymonitorpasswd
```

The monpasswd parameter may be either a plain text password or it may be an encrypted password. See the section on encrypting passwords for use in the MaxScale.cnf file.

### **Listener**

The listener defines a port and protocol pair that is used to listen for connections to a service. A service may have multiple listeners associated with it, either to support multiple protocols or multiple ports. As with other elements of the configuration the section name is the listener name and a type parameter is used to identify the section as a listener definition.

```
[Test Listener]
type=listener
service=Test Service
protocol=MySQLClient
address=localhost
port=4008
socket=/tmp/testlistener.sock
```

### **Service**

The service to which the listener is associated. This is the name of a service that is defined elsewhere in the configuration file.

### **Protocol**

The name of the protocol module that is used for the communication between the client and MaxScale itself.

### **Address**

The address option sets the address that will be used to bind the listening socket. The address may be specified as an IP address in 'dot notation' or as a hostname. If the address option is not included in the listener definition the listener will bind to all network interfaces.

### **Port**

The port to use to listen for incoming connections to MaxScale from the clients. If the port is omitted from the configuration a default port for the protocol will be used.

## Socket

The socket option may be included in a listener definition, this configures the listener to use Unix domain sockets to listen for incoming connections. The parameter value given is the name of the socket to use.

If a socket option and an address option is given then the listener will listen on both the specific IP address and the Unix socket.

## Monitor

In order for the various router modules to function correctly they require information about the state of the servers that are part of the service they provide. MaxScale has the ability to internally monitored the state of the back-end database servers or that state may be feed into MaxScale from external monitoring systems. If automated monitoring and failover of services is required this is achieved by running a monitor module that is designed for the particular database architecture that is in use.

Monitors are defined in much the same way as other elements in the configuration file, with the section name being the name of the monitor instance and the type being set to monitor.

```
[MySQL Monitor]
type=monitor
module=mysqlmon
servers=server1,server2,server3
user=dbmonitoruser
passwd=dbmonitorpwd
```

## Module

The module parameter defines the name of the loadable module that implements the monitor. This module is loaded and executed on a separate thread within MaxScale.

## Servers

The servers parameter is a comma separated list of server names to monitor, these are the names defined elsewhere in the configuration file. The set of servers monitored by a single monitor need not be the same as the set of servers used within any particular server, a single monitor instance may monitor servers in multiple servers.

## User

The user parameter defines the username that the monitor will use to connect to the monitored databases. Depending on the monitoring module used this user will require specific privileges in order to determine the state of the nodes, details of those privileges can be found in the sections on each of the monitor modules.

Individual servers may define override values for the user and password the monitor uses by setting the monuser and monpasswd parameters in the server section.

**Passwd**

The password parameter may be either a plain text password or it may be an encrypted password. See the section on encrypting passwords for use in the MaxScale.cnf file.

## Protocol Modules

The protocol supported by MaxScale are implemented as external modules that are loaded dynamically into the MaxScale core. These modules reside in the directory `$MAXSCALE_HOME/module`, if the environment variable `$MAXSCALE_HOME` is not set it defaults to `/usr/local/skysql/MaxScale`. It may also be set by passing the `-c` option on the MaxScale command line.

### MySQLClient

This is the implementation of the MySQL protocol that is used by clients of MaxScale to connect to MaxScale.

### MySQLBackend

The MySQLBackend protocol module is the implementation of the protocol that MaxScale uses to connect to the backend MySQL, MariaDB and Percona Server databases. This implementation is tailored for the MaxScale to MySQL Database traffic and is not a general purpose implementation of the MySQL protocol.

### Telnetd

The telnetd protocol module is used for connections to MaxScale itself for the purposes of creating interactive user sessions with the MaxScale instance itself. Currently this is used in conjunction with a special router implementation, the debugcli.

### HTTPD

This protocol module is currently still under development, it provides a means to create HTTP connections to MaxScale for use by web browser or RESTful API clients.

## Router Modules

The main task of MaxScale is to accept database connections from client applications and route the connections or the statements sent over those connections to the various services supported by MaxScale.

There are two flavours of routing that MaxScale can perform, connection based routing and statement based routine. These each have their own characteristics and costs associated with them.

### Connection Based Routing

Connection based routing is a mechanism by which MaxScale will, for each incoming connection decide on an appropriate outbound server and will forward all statements to that server without examining the internals of the statement. Once an inbound connection is associated to a particular backend database it will remain connected to that server until the connection is closed or the server fails.

### Statement Based Routing

Statement based routing is somewhat different, the routing modules examine every statement the client sends and determines, on a per statement basis, which of the set of backend servers in the service is best to execute the statement. This gives better dynamic balancing of the load within the cluster but comes at a cost. The query router must understand the statement that is being routing and will typically need to parse the statement in order to achieve this. This parsing within the router adds a significant overhead to the cost of routing and makes this type of router only really suitable for loads in which the gains outweigh this added cost.

### Available Routing Modules

Currently a small number of query routers are available, these are in different stages of completion and offer different facilities.

#### **Readconnroute**

This is a statement based query router that was originally targeted at environments in which the clients already performed splitting of read and write queries into separate connections.

Whenever a new connection is received the router will examine the state of all the servers that form part of the service and route the connection to the server with least connections currently that matches the filter constraints given in the router options. This results in a balancing of the active connections, however different connections may have different lifetimes and the connections may become unbalanced when later viewed.

The readconnroute router can be configured to balance the connections from the clients across all the backend servers that are running, just those backend servers that are currently replication slaves or those that are replication masters when routing to a master slave

replication environment. When a Galera cluster environment is in use the servers can be filtered to just the set that are part of the cluster and in the 'syncd' state. These options are configurable via the `router_options` that can be set within a service. The `router_option` strings supported are "master", "slave" and "syncd".

### Master/Slave Replication Setup

To setup MaxScale to route connections to evenly between all the current slave servers in a replication cluster a service entry of the form shown below is required.

```
[Read Service]
type=service
router=readconnroute
router_options=slave
servers=server1,server2,server3,server4
user=maxscale
auth=thepasswd
```

With the addition of a listener for this service, which defines the port and protocol that MaxScale uses

```
[Read Listener]
type=listener
service=Read Service
protocol=MySQLClient
port=4006
```

the client can now connect to port 4006 on the host which is running MaxScale. Statements sent using this connection will then be routed to one of the slaves in the server set defined in the Read Service. Exactly which is selected will be determined by balancing the number of connections to each of those whose current state is "slave".

Altering the router options to be `slave`, `master` would result in the connections being balanced between all the servers within the cluster.

It is assumed that the client will have a separate connection to the master server, however this can be routed via MaxScale, allowing MaxScale to manage the determination of which server is master. To do this you would add a second service and listener definition for the master server.

```
[Write Service]
type=service
router=readconnroute
router_options=master
servers=server1,server2,server3,server4
user=maxscale
auth=thepasswd
```



```
[Write Listener]
type=listener
service=Write Service
protocol=MySQLClient
port=4007
```

This allows the clients to direct write requests to port 4007 and read requests to port 4006 of the MaxScale host without the clients needing to understand the configuration of the Master/Slave replication cluster.

Connections to port 4007 would automatically be directed to the server that is the master for replication at the time connection is opened. Whilst this is a simple mapping to a single server it does give the advantage that the clients have no requirement to track which server is currently the master, devolving responsibility for managing the failover to MaxScale.

In order for MaxScale to be able to determine the state of these servers the mysqlmon monitor module should be run against the set of servers that comprise the service.

#### Galera Cluster Configuration

Although not primarily designed for a multi-master replication setup it is possible to use the readconroute in this situation. The readconroute connection router can be used to balance the connection across a Galera cluster. A special monitor is available that detects if nodes are joined to a Galera Cluster, with the addition of a router option to only route connections to nodes marked as synced. MaxScale can ensure that users are never connected to a node that is not a full cluster member.

```
[Galera Service]
type=service
router=readconroute
router_options=syncd
servers=server1,server2,server3,server4
user=maxscale
auth=thepasswd
```

```
[Galera Listener]
type=listener
service=Galera Service
protocol=MySQLClient
port=3336
```

```
[Galera Monitor]
type=monitor
module=galeramon
servers=server1,server2,server3,server4
user=galeramon
```

```
passwd=galeramon
```

## Readwritesplit

The readwritesplit is a statement based router that has been designed for use within Master/Slave replication environments. It examines every statement, parsing it to determine if the statement falls into one of three categories;

- read only statement
- possible write statement
- session modification statement

Each of these three categories has a different action associated with it. Read only statements are sent to a slave server in the replication cluster. Possible write statements, which may include read statements that have an undeterminable side effect, are sent to the current replication master. Statements that modify the session are sent to all the servers, with the result that is generated by the master server being returned to the user.

Session modification statements must be replicated as they affect the future results of read and write operations, so they must be executed on all servers that could execute statements on behalf of this client.

Currently the readwritesplit router module is under development and is limited:

- Only a single slave connection is managed currently, therefore statements can not be balanced across multiple slaves.
- Connection failover support has not yet be implemented. Client connections will fail if the master server fails over.

### Master/Slave Replication Setup

To setup the readwritesplit connection router in a master/slave failover environment is extremely simple, a service definition is required with the router defined for the service and an associated listener.

The router\_options parameter is not required.

```
[Split Service]
type=service
router=readwritesplit
servers=server1,server2,server3,server4
user=maxscale
auth=thepasswd
```

```
[Split Listener]
type=listener
service=Split Service
protocol=MySQLClient
port=3336
```

The client would merely connect to port 3336 on the MaxScale host and statements would be

directed to the master or slave as appropriate. Determination of the master or slave status may be done via a monitor module within MaxScale or externally. In this latter case the server flags would need to be set via the MaxScale debug interface, in future versions an API will be available for this purpose.

## **Debugcli**

The debugcli is a special case of a statement based router. Rather than direct the statements at an external data source they are handled internally. These statements are simple text commands and the results are the output of debug commands within MaxScale. The service and listener definitions for a debug cli service only differ from other services in that they require no backend server definitions.

### Debug CLI Configuration

The definition of the debug cli service is illustrated below

```
[Debug Service]
type=service
router=debugcli

[Debug Listener]
type=listener
service=Debug Service
protocol=telnetd
port=4442
```

Connections using the telnet protocol to port 4442 of the MaxScale host will result in a new debug CLI session. A default username and password are used for this module which may be overridden by creating new account for user. The default username is admin with a password of skysql.

## Monitor Modules

Monitor modules are used by MaxScale to internally monitor the state of the backend databases in order to set the server flags for each of those servers. The router modules then use these flags to determine if the particular server is suitable for routing connections to particular query classifications to. The monitors are run within separate threads of MaxScale and do not affect the MaxScale performance.

The use of monitors is optional, it is possible to run MaxScale with external monitoring, in which case arrangements must be made for an external entity to set the status of each of the servers that MaxScale can route to.

### Mysqlmon

The MySQLMon monitor is a simple monitor designed for use with MySQL Master/Slave replication cluster. To execute the mysqlmon monitor an entry as shown below should be added to the MaxScale configuration file.

```
[MySQL Monitor]
type=monitor
module=mysqlmon
servers=server1,server2,server3,server4
```

This will monitor the 4 servers; server1, server2, server3 and server4. It will set the status of running or failed and master or slave for each of the servers.

The monitor uses the username given in the monitor section or the server specific user that is given in the server section to connect to the server. This user must have sufficient permissions on the database to determine the state of replication. The roles that must be granted to this user are REPLICATION SLAVE and REPLICATION CLIENT.

To create a user that can be used to monitor the state of the cluster, the following commands could be used.

```
MariaDB [mysql]> create user 'maxscalemon'@'maxscalehost'
identified by 'Ha79hjds';
Query OK, 0 rows affected (0.01 sec)
```

```
MariaDB [mysql]> grant REPLICATION SLAVE on *.* to
'maxscalemon'@'maxscalehost';
Query OK, 0 rows affected (0.00 sec)
```

```
MariaDB [mysql]> grant REPLICATION CLIENT on *.* to
'maxscalemon'@'maxscalehost';
Query OK, 0 rows affected (0.00 sec)
```

```
MariaDB [mysql]>
```

Assuming that MaxScale is running on the host maxscalehost.

## GaleraMon

The GaleraMon monitor is a simple router designed for use with MySQL Galera cluster. To execute the galeraMon monitor an entry as shown below should be added to the MaxScale configuration file.

```
[Galera Monitor]
type=monitor
module=galeraMon
servers=server1,server2,server3,server4
```

This will monitor the 4 servers; server1, server2, server3 and server4. It will set the status of running or failed and joined for those servers that reported the Galera JOINED status.

The user that is configured for use with the Galera monitor must have sufficient privileges to select from the information\_schema database and GLOBAL\_STATUS table within that database.

To create a user that can be used to monitor the state of the cluster, the following commands could be used.

```
MariaDB [mysql]> create user 'maxscalemon'@'maxscalehost'
identified by 'Ha79hjds';
Query OK, 0 rows affected (0.01 sec)
```

```
MariaDB [mysql]> grant SELECT on
INFORMATION_SCHEMA.GLOBAL_STATUS to
'maxscalemon'@'maxscalehost';
Query OK, 0 rows affected (0.00 sec)
```

```
MariaDB [mysql]>
```

Assuming that MaxScale is running on the host maxscalehost.

## Encrypting Passwords

Passwords stored in the MaxScale.cnf file may optionally be encrypted for added security. This is done by creation of an encryption key on installation of MaxScale. Encryption keys may be created manually by executing the `maxkeys` utility with the argument of the filename to store the key.

```
maxkeys $MAXSCALE_HOME/etc/.secrets
```

Changing the encryption key for MaxScale will invalidate any currently encrypted keys stored in the MaxScale.cnf file.

## Creating Encrypted Passwords

Encrypted passwords are created by executing the `maxpasswd` command with the password you require to encrypt as an argument. The environment variable `MAXSCALE_HOME` must be set, or MaxScale must be installed in the default location before `maxpasswd` can be executed.

```
maxpasswd MaxScalePw001  
61DD955512C39A4A8BC4BB1E5F116705
```

The output of the `maxpasswd` command is a hexadecimal string, this should be inserted into the MaxScale.cnf file in place of the ordinary, plain text, password. MaxScale will determine this as an encrypted password and automatically decrypt it before sending it the database server.

```
[Split Service]  
type=service  
router=readwritesplit  
servers=server1,server2,server3,server4  
user=maxscale  
password=61DD955512C39A4A8BC4BB1E5F116705
```

## Configuration Updates

The current MaxScale configuration may be updated by editing the configuration file and then forcing MaxScale to reread the configuration file. To force MaxScale to reread the configuration file a SIGTERM signal is sent to the MaxScale process.

Some changes in configuration can not be dynamically changed and require a complete restart of MaxScale, whilst others will take some time to be applied.

## Limitations

Services that are removed via the configuration update mechanism can not be physically removed from MaxScale until there are no longer any connections using the service.

When the number of threads is decreased the threads will not actually be terminated until such time as they complete the current operation of that thread.

Monitors can not be completely removed from the running MaxScale.

## Authentication

MySQL uses username, passwords and the client host in order to authenticate a user, so a typical user would be defined as user X at host Y and would be given a password to connect. MaxScale uses exactly the same rules as MySQL when user connect to the MaxScale instance, i.e. it will check the address from which the client is connecting and treat this in exactly the same way that MySQL would. MaxScale will pull the authentication data from one of the backend servers and use this to match the incoming connections, the assumption being that all the backend servers for a particular service will share the same set of user credentials.

It is important to understand however that when MaxScale itself makes connections to the backend servers the backend server will see all connections as originating from the host that runs MaxScale and not the original host from which the client connected to MaxScale. Therefore the backend servers should be configured to allow connections from the MaxScale host for every user that can connect from any host. Since there is only a single password within the database server for a given host, this limits the configuration such that a given user name must have the same password for every host from which they can connect.

To clarify, if a user X is defined as using password *pass1* from host a and *pass2* from host b then there must be an entry in the user table for user X from the MaxScale host, say *pass1*.

This would result in rows in the user table as follows

Username	Password	Client Host
X	pass1	a
X	pass2	b
X	pass1	MaxScale

In this case the user X would be able to connect to MaxScale from host a giving the password of *pass1*. In addition MaxScale would be able to create connections for this user to the backend servers using the username X and password *pass1*, since the MaxScale host is also defined to have password *pass1*. User X would not however be able to connect from host b since they would need to provide the password *pass2* in order to connect to MaxScale, but then MaxScale would not be able to connect to the backends as it would also use the password *pass2* for these connections.

## Wildcard Hosts



Hostname mapping in MaxScale works in exactly the same way as for MySQL, if the wildcard is used for the host then any host other than the localhost (127.0.0.1) will match. It is important to consider that the localhost check will be performed at the MaxScale level and at the MySQL server level.

If MaxScale and the databases are on separate hosts there are two important changes in behaviour to consider:

1. Clients running on the same machine as the backend database now may access the database using the wildcard entry. The localhost check between the client and MaxScale will allow the use of the wildcard, since the client is not running on the MaxScale host. Also the wildcard entry can be used on the database host as MaxScale is making that connection and it is not running on the same host as the database.
2. Clients running on the same host as MaxScale can not access the database via MaxScale using the wildcard entry since the connection to MaxScale will be from the localhost. These clients are able to access the database directly, as they will use the wildcard entry.

If MaxScale is running on the same host as one or more of the database nodes to which it is acting as a proxy then the wildcard host entries can be used to connect to MaxScale but not to connect onwards to the database running on the same node.

In all these cases the issue may be solved by adding an explicit entry for the localhost address that has the same password as the wildcard entry. This may be done using a statement as below for each of the databases that are required:

```
MariaDB [mysql]> GRANT SELECT, INSERT, UPDATE, DELETE,  
CREATE, DROP ON employee.* 'user1'@'localhost' IDENTIFIED BY  
'xxx';  
Query OK, 0 rows affected (0.00 sec)
```

## Limitations

At the time of writing the authentication mechanism within MaxScale does not support IPV6 matching if connections rules. This is also in line with the current protocol modules that do not support IPV6.

## Error Reporting

MaxScale is designed to be executed as a service, therefore all error reports, including configuration errors, are written to the MaxScale error log file. MaxScale will log to a set of files in the directory `$MAXSCALE_HOME/log`, the only exception to this is if the log directory is not writable, in which case a message is sent to the standard error descriptor.