MaxScale

Configuration & Usage Scenarios

Mark Riddoch

Last Updated: 8th September 2014

Contents

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
13rd 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
29th May 2014	Added troubleshooting section	Massimiliano Pinto
2nd June 2014	Correction of some typos, clarification of the meaning of session modification statements and the default user for the CLI. Addition of debugcli configuration option for developer and user modes.	Mark Riddoch

4th June 2014	Addition of “monitor_interval” for all monitors and “detect replication_lag” for MySQL monitor	Massimiliano Pinto
6th June 2014	Addition of filters sections	Mark Riddoch
27th June 2014	Addition of server weighting, the configuration for the maxadmin client	Mark Riddoch
2nd July 2014	Addition of new readwritesplit router options with description and examples.	Vilho Raatikka
31st July	Addition of NDB monitor for MySQL Cluster	Massimiliano Pinto
8th August 2014	Addition of routing hints	Vilho Raatikka
28th August 2014	Addition of “detect_stale_master” option for MySQL monitor	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 clien 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 quests 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.
filter	A module that can be placed between the client and the MaxScale router module. All client data passes through the filter module and may be examined or modified by the filter modules. Filters may be chained together to form processing pipelines.

Configuration

The MaxScale configuration is read from a file which can be located in a number of placing, 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=readconnroute
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=readconnroute
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.

Filters

The `filters` option allow a set of filters to be defined for a service; requests from the client are passed through these filters before being sent to the router for dispatch to the backend server. The filters parameter takes one or more filter names, as defined within the filter definition section of the configuration file. Multiple filters are separated using the | character.

```
filters=counter | QLA
```

The requests pass through the filters from left to right in the order defined in the configuration parameter.

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=Mhu87p2D
```

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. For 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
'maxscale'@'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
```

Values of “on” or “true” may also be given to enable the root user and “off” or “false” may be given to disable the use of the root user.

```
enable_root_user=true
```

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`

weightby

The weightby parameter is used in conjunction with server parameters in order to control the load balancing applied in the router in use by the service. This allows varying weights to be applied to each server to create a non-uniform distribution of the load amongst the servers.

An example of this might be to define a parameter for each server that represents the amount of resource available on the server, we could call this serversize. Every server should then have a serversize parameter set for the server.

```
serversize=10
```

The service would then have the parameter weightby set. If there are 4 servers defined in the service, serverA, serverB, serverC and serverD, with the serversize set as shown in the table below, the connections would balanced using the percentages in this table.

Server	serversize	% connections
serverA	10	18%
serverB	15	27%
serverC	10	18%
serverD	20	36%

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.

Filter

Filters provide a means to manipulate or process requests as they pass through MaxScale between the client side protocol and the query router. A filter should be defined in a section with a type of filter.

```
[QLA]
type=filter
module=qlafilter
options=/tmp/QueryLog
```

The section name may then be used in one or more services by using the `filters=` parameter in the service section. In order to use the above filter for a service called “QLA Service”, an entry of the following form would exist for that service.

```
[QLA Service]
type=service
router=readconnroute
router_options=slave
servers=server1,server2,server3,server4
user=massi
passwd=6628C50E07CCE1F0392EDEEB9D1203F3
```

```
filters=QLA
```

See the Services section for more details on how to configure the various options of a service.

Module

The module parameter defines the name of the loadable module that implements the filter.

Options

The options parameter is used to pass options to the filter to control the actions the filter will perform. The values that can be passed differ between filter implementation, the inclusion of an options parameter is optional.

Other Parameters

Any other parameters present in the filters section will be passed to the filter to be interpreted by the filter. An example of this is the regexfilter that requires the two parameters `match` and `replace`

```
[regex]
type=filter
module=regexfilter
match=form
replace=from
```

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 monitor 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
monitor_interval=8000
```

```
detect_replication_lag=0
detect_stale_master=0
```

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.

Monitor_interval

The monitor_interval parameter sets the sampling interval in milliseconds for each monitor, the default value is 10000 milliseconds.

Detect_replication_lag

This options if set to 1 will allow monitor to collect the replication lag among all configured slaves by checking the content of `maxscale_schema.replication_heartbeat` table.
The master server writes in and slaves fetch a UNIX timestamp from that there.
This timestamp is updated in each node server struct and it's used to calculate the replication lag.
That value is also used by the Read / Write split module via `max_slave_replication_lag` and `LEAST_BEHIND_MASTER` options.
This monitor option is not enabled by default.

Detect_stale_master

This options if set to 1 will allow monitor to select the previous selected Master for next

operations even if no slaves at all are found by the monitor polling.

This is such a case when the replication on all slave has been stopped via `STOP SLAVE` or the current configuration was removed by `RESET SLAVE ALL`.
As there are no slaves the replication topology cannot be computed and MaxScale can only check if the current monitored server was the master before: if that's the case mySQL monitor adds to the server status field the `SERVER_STALE_STATUS` bit and a log entry appears in the Message Log file.

If MaxScale or monitor is restarted and the Replication is still not configured or started there will not be any master server available even with this option enabled.

This option is not enabled by default and should be used at the administrator risk.

Protocol Modules

The protocols 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.

maxscaled

The protocol used used by the maxadmin client application in order to connect to MaxScale and access the command line interface.

HTTPD

This protocol module is currently still under development, it provides a means to create HTTP connections to MaxScale for use by web browsers 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 'synced' 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 "synced".

Master/Slave Replication Setup

To setup MaxScale to route connections 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 readconnroute in this situation. The readconnroute connection router can be used to balance the connections 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=readconnroute
router_options=synced
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
```

The specialized Galera monitor can also select one of the node in the cluster as master, the others will be marked as slave.
These roles are only assigned to synced nodes.

It then possible to have services/listeners with router_options=master or slave accessing a subset of all galera nodes.
The “synced” simply means: access all nodes.

Examples:

```
[Galera Master Service]
type=service
router=readconnroute
router_options=master

[Galera Slave Service]
type=service
router=readconnroute
router_options=slave
```

The Master and Slave roles are also available for the Read/Write Split router operation

MySQL Cluster Configuration

The readconnroute connection router can be used to balance the connections across a

MySQL cluster SQL nodes. A special monitor is available that detects if SQL nodes are connected to data nodes, with the addition of a router option to only route connections to nodes marked as NDB.
MaxScale can ensure that users are never connected to a node that is not a full cluster member.

Example:

```
[NDB Cluster Monitor]
type=monitor
module=ndbclustermon
servers=server1,server2
user=monitor
passwd=monitor

[MySQL Cluster Service]
type=service
router=readconnroute
router_options=ndb
servers=server1,server2

[Cluster Listener]
type=listener
service=MySQL Cluster Service
protocol=MySQLClient
port=4906
```

The “ndb” router option simply means: access all SQL nodes marked with NDB status, i.e. they are members of the cluster.

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[1]

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.

Failure Tolerance

Read/write split router implements transparent slave connection failover. Client connections are not affected if slave connection fails. Active transactions will fail but they can be re-executed without the need for re-connect.

Currently the readwritesplit router module is under development and has the following limitations:

- Master connection failover support has not yet been implemented. Client connections will fail if the master server fails over.

Master/Slave Replication Setup

To setup the readwritesplit connection router in a MySQL Replication requires a service definition with the router defined for the service and an associated listener.
The router_options parameter is not required but it can be used to specify how slave(s) are selected. Available option is `slave_selection_criteria` and possible value are `LEAST_BEHIND_MASTER` and `LEAST_CURRENT_OPERATIONS`.
Parameter `max_slave_connections` sets the maximum number of slaves a router session tries to connect to. `max_slave_replication_lag` sets the maximum allowed

[1] A session modification statement is any statement that is executed that may affect the behaviour of subsequent statements within the current connection. Examples of such statements are the USE SQL statement or a SET statement using the SESSION scope. PREPARE STMT clauses are session statements in MaxScale since they are executed in every backend server.

lag for slave in seconds. The criteria is checked when router chooses slaves and only slaves having smaller lag can be selected. The lag is continuously monitored by the monitor, and if it exceeds the accepted level the slave won't be used until it catches up with the master again. Note that replication lag is ignored if user has specified a routing hint.
`read_ses_variable_from_slaves` allows session variable data to be read from slave. It is recommended to `set write_ses_variables_to_all=Yes` to ensure that latest variable values can be found from every node involved in the session. Setting 'No' to both parameters routes all queries using any user- or system variables to master.

```
[RWSplit Service]
type=service
router=readwritesplit
router_options=slave_selection_criteria=LEAST_BEHIND_MASTER
max_slave_connections=50%
max_slave_replication_lag=30 # seconds
read_ses_variables_from_slaves=No
write_ses_variables_to_all=Yes
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.

Galera Cluster Configuration

Master and Slave roles that galera monitor assign to nodes make possible the Read Write split approach to Galera Cluster as well.

Simply configure a Split Service with galera nodes:

```
[Galera Split Service]
```

```
type=service
router=readwritesplit
servers=galera_node1,galera_node2,galera_node3
```

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, new users may be created using the `add user` command. As soon as any users are explicitly created the default username will no longer continue to work. The default username is admin with a password of skysql.

The debugcli supports two modes of operation, developer mode and user mode. The mode is set via the router_options parameter of the debugcli. The user mode is more suited to end-users and administrators, whilst the develop mode is explicitly targeted to software developing adding or maintaining the MaxScale code base. Details of the differences between the modes can be found in the debugging guide for MaxScale. The default mode for the debugcli is user mode. The following service definition would enable a developer version of the debugcli.

It should be noted that both a user and a developer version of the debugcli may be defined within the same instance of MaxScale, however they must be defined as two distinct services, each with a distinct listener.

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

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

[Admin Service]
type=service
router=debugcli

[Admin Listener]
type=listener
service=Debug Service
protocol=telnetd
port=4242
```

CLI

The command line interface as used by maxadmin. This is a variant of the debugcli that is built slightly differently so that it may be accessed by the client application maxadmin. It requires the used ot the maxscaled protocol to interact with it.

CLI Configuration

There are two components to the definition required in order to run the command line interface to use with MaxAdmin; a service and a listener.
The default entries required are shown below.

```
[CLI]
type=service
router=cli

[CLI Listener]
type=listener
service=CLI
protocol=maxscaled
address=localhost
port=6603
```

Note that this uses the default port of 6603 and confines the connections to localhost connections only. Remove the `address=` entry to allow connections from any machine on your network. Changing the port from 6603 will mean that you must allows pass a -p option to the MaxAdmin command.

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 a suitable destination for routing connections for particular query classifications. 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 assuming that MaxScale is running on the host maxscalehost.

```
create user 'maxscalemon'@'maxscalehost' identified by 'Ha79hjds';

grant REPLICATION SLAVE on *.* to 'maxscalemon'@'maxscalehost';

grant REPLICATION CLIENT on *.* to 'maxscalemon'@'maxscalehost';
```

MySQL monitor fetches the `@@server_id` variable and other informations from `SHOW SLAVE STATUS` in order to compute the replication topology tree that may include intermediate master servers, called relay servers.
The Master server used by router modules is the so called “root master”: a server that has the `SERVER_MASTER` status bit set and it’s at the lowest level of the replication depth.

MySQL monitor may optionally detect the replication lag among servers by using the `maxscale_schema.replication_heartbeat` table: the monitor user must have rights to create it and write into.

Another option may also allow to set a Stale Master when the replication has been stopped or the configuration doesn't allow to have both IO and SQL replication threads running on all slaves: the previous detected working Master will be selected for read and write operations.

Please note, those two options are not enabled by default.

Galeramon

The Galeramon monitor is a simple monitor 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 synced for those servers that reported the Galera SYNCED 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.

```
create user 'maxscalemon'@'maxscalehost' identified by 'Ha79hjds';

grant SELECT on INFORMATION_SCHEMA.GLOBAL_STATUS to
'maxscalemon'@'maxscalehost';
```

Assuming that MaxScale is running on the host maxscalehost.

The Galera monitor can also assign Master and Slave roles to the configured nodes:

among the set of synced servers, the one with the lowest value of 'wsrep_local_index' is selected as the current master while the others are slaves.

This way is possible to configure the node access based not only on 'synced' state but even on Master and Slave role enabling the use of Read Write split operation on a Galera cluster and avoiding any possible write conflict.

Example status for a Galera server node is:

```
Server 0x261fe50 (server2)
      Server:          192.168.1.101
      Status:          Master, Synced, Running
```

NDBclustermon

The NDBclustermon monitor is a simple router designed for use with MySQL Cluster. To execute the ndclustermon monitor an entry as shown below should be added to the MaxScale configuration file.

Example for monitor section:

```
[NDB Cluster Monitor]
type=monitor
module=ndbclustermon
servers=server1,server2
```

This will monitor the two SQL Node server1, server2 and will set the status of running or failed and NDB for those servers that reported the value of status variable **Ndb_number_of_ready_data_nodes** is greater than 0 - i.e. the monitored SQL node is able to contact one or more data nodes.

The user that is configured for use with the NDBcluster monitor must have sufficient privileges to select from the information_schema database and GLOBAL_STATUS table within that database..

Example of a monitored server:

```
Server 0x3873a40 (server2)
        Server:                 192.168.90.81
        Status:                 NDB, Running
        Protocol:               MySQLBackend
        Port:                   3306
        Server Version:         5.5.38-ndb-7.2.17-cluster-gpl
        Node Id:                13
```

The MySQL Cluster variables fetched by the monitor are:

```
mysql> SHOW STATUS LIKE 'Ndb_number_of_ready_data_nodes';
+---------------------------------+-------+
| Variable_name                   | Value |
+---------------------------------+-------+
| Ndb_number_of_ready_data_nodes | 2     |
+---------------------------------+-------+
1 row in set (0.00 sec)
```

The result is greater than 0 so the NBD status is added to status

```
mysql> SHOW STATUS LIKE 'Ndb_cluster_node_id';
+---------------------+-------+
| Variable_name       | Value |
+---------------------+-------+
| Ndb_cluster_node_id | 13    |
+---------------------+-------+
1 row in set (0.00 sec)
```

The value is stored in node_id server field.

Filter Modules

Currently four example filters are included in the MaxScale distribution

Module	Description
testfilter	Statement counting Filter - a simple filter that counts the number of SQL statements executed within a session. Results may be viewed via the debug interface.
qlafilter	Query Logging Filter - a simple query logging filter that write all statements for a session into a log file for that session.
regexfilter	Query Rewrite Filter - an example of how filters can alter the query contents. This filter allows a regular expression to be defined, along with replacement text that should be substituted for every match of that regular expression.
tee	A filter that duplicates SQL requests and sends the duplicates to another service within MaxScale.
hintfilter	Allows user for adding routing hints or modify MaxScale parameter values on-the-fly.

These filters are merely examples of what may be achieved with the filter API and are not sophisticated or consider as suitable for production use, they merely illustrate the functionality possible.

Hint filter

With hints users can specify the where an individual or group of queries should be routed and dynamically modify MaxScale configuration parameter values. It is possible to create several named hints which can be enabled, disabled and changed to another hint during the session they were created.

Routing hint determines where query will be routed unless following the hint would violate database consistency or cluster status. Routing target can be `master` or named backend server. The name used for pointing the routing target is the server section name from MaxScale configuration file.

Parameter hint allows the user to set value for supported parameters. The scope of the setting is same as with routing hints. Today, August 2014, the only supported parameter is `max_slave_replication_lag`.

Configuring Hints

Hints are configured much like the other filters. What is needed is a listener for incoming client requests, section describing the filter and a service providing hints. As of today (August 2014) the router used by the service is readwritesplit router.

```
[Hint listener]
type=listener
service=RWSplit Hint Router
protocol=MySQLClient
port=4010

[hints]
type=filter
module=hintfilter
```

The service section only needs the the filter name as follows.

```
filters=hints
```

See the Examples section below for full service description.

Hint Syntax

Hint specification is added to the comment of an SQL query. Use either `'-- '` or `'#'` after the semicolon or `'/* .. */'` before the semicolon. Notice white characters. All hints must start with the `'maxscale'` tag :

```
<query> ; -- maxscale <hint>
```

Where routing `<hint>` has the following syntax:

```
route to [master | slave | server <server name>]
```

Parameter `<hint>` syntax:

```
<param>=<value>
```

Note that both syntaxes above only describe one-off hint syntax. Stored - named and unnamed - hint syntax is listed below.

Named hint syntax:

```
<hint name> prepare <hint>
```

Once named hint is prepared it can be activated with the following syntax:

```
<hint name> begin
```

If there are multiple hints defined one after another, deactivating a hint makes previous hint automatically active. Deactivating hint is done simply with the following hint:

```
end
```

A hint can be defined and activated in a single command using the syntax:

```
<hint name> begin <hint>
```

Examples:

```
[RWSplit Hint Router]
type=service
router=readwritesplit
servers=server1,server2,server3,server4,server5,server6,server7
max_slave_connections=100%
router_options=slave_selection_criteria=LEAST_CURRENT_OPERATIONS
user=maxscale
passwd=mypasswd
filter=hints
```

- Defines Read/Write Router with hints filter

```
SELECT @@server_id ; # maxscale routemaster prepare route to master
```
- creates hint named ‘routemaster’, saves it but keeps it inactive

```
SELECT @@server_id ; # maxscale routesrv2 begin route to server srv2
```
- creates hint ‘routesrv2’, saves and makes it immediately active

```
SELECT @@server_id ; # maxscale routemaster begin
```
- makes previously created ‘routemaster’ active by replacing ‘routesrv2’

```
SELECT @@server_id ; #maxscale end
```
- deactivates currently active hint (‘routemaster’), ‘routesrv2’ becomes active again

```
SELECT @@server_id ; # maxscale end
```
- deactivates currently active hint (‘routesrv2’)

```
SELECT @@server_id; # maxscale route to server srv8
```
- routes the current query to server ‘srv8’

```
SELECT @@server_id; # maxscale relaxed prepare
max_slave_replication_lag=180
```
- creates hint ‘relaxed’ and saves it for later use

```
SELECT @@server_id; # maxscale strict prepare max_slave_replication_lag=45
```
- creates hint ‘strict’ and saves it for later use

```
SELECT @@server_id; # maxscale relaxed begin
```
- makes ‘strict’ active

Examples with alternative syntax for hints, such as:

```
SELECT @@server_id; -- maxscale route to server srv8
SELECT @@server_id /* maxscale route to server srv8 */ ;
SELECT /* maxscale route to server server4 */ @@server_id;
```

Please note mysql client requires the -c option sending comments to the server:

[user1@server-45-01 bin]# mysql -c -h 127.0.0.1 -P 4606 -uuser -ppasswd

```
MariaDB [(none)]> SELECT @@server_id /* maxscale route to server server2 */
;
+-------------+
| @@server_id |
+-------------+
|           2 |
+-------------+
1 row in set (0.00 sec)
```

Statement Counting Filter

The statement counting filter is implemented in the module names `testfilter` and merely keeps a count of the number of SQL statements executed. The filter requires no options to be passed and takes no parameters. The statement count can be viewed via the diagnostic and debug interface of MaxScale.

In order to add this filter to an existing service create a filter section to name the filter as follows

```
[counter]
type=filter
module=testfilter
```

Then add the filter to your service by including the filters= parameter in the service section.

```
filters=counter
```

Query Log All Filter

The QLA filter simply writes all SQL statements to a log file along with a timestamp for the statement. An example of the file produced by the QLA filter is shown below

```
00:36:04.922 5/06/2014, select @@version_comment limit 1
00:36:12.663 5/06/2014, SELECT DATABASE()
00:36:12.664 5/06/2014, show databases
00:36:12.665 5/06/2014, show tables
```

A new file is created for each client connection, the name of the logfile can be controlled by the use of the router options. No parameters are used by the QLA filter. The filter is implemented by the loadable module `qlafilter`.

To add the QLA filter to a service you must create a filter section to name the filter, associated the loadable module and define the filename option.

```
[QLA]
type=filter
module=qlafilter
options=/tmp/QueryLog
```

Then add the filters= parameter into the service that you wish to log by adding this parameter

to the service section

```
filters=QLA
```

A log file will be created for each client connection, the name of that log file will be /tmp/QueryLog.<number>

Regular Expression Filter

The regular expression filter is a simple text based query rewriting filter. It allows a regular expression to be used to match text in a SQL query and then a string replacement to be made against that match. The filter is implemented by the `regexfilter` loadable module and is passed two parameters, a match string and a replacement string.

To add the filter to your service you must first create a filter section to name the filter and give the match and replacement strings. Here we define a filter that will convert to MariaDB 10 command show all slaves status to the older form of show slave status for MariaDB 5.5.

```
[slavestatus]
type=filter
module=regexfilter
match=show *all *slaves
replace=show slave
```

You must then add this filter to your service by adding the filters= option

```
filters=slavestatus
```

Another example would be a filter to convert from the MySQL 5.1 `create table` syntax that used the `TYPE` keyword to the newer `ENGINE` keyword.

```
[EnginerFilter]
type=filter
module=regexfilter
match=TYPE
replace=ENGINE
```

This would then change the SQL sent by a client application written to work with MySQL 5.1 into SQL that was compliant with MySQL 5.5. The statement

```
create table supplier(id integer, name varchar(80))
type=innodb
```

would be replaced with

```
create table supplier(id integer, name varchar(80))
ENGINE=innodb
```

before being sent to the server. Note that the text in the match string is case independent.

Tee Filter

The tee filter is a filter module for MaxScale is a “plumbing” fitting in the MaxScale filter toolkit. It can be used in a filter pipeline of a service to make a copy of requests from the client and dispatch a copy of the request to another service within MaxScale.

The configuration block for the TEE filter requires the minimal filter parameters in it’s section within the MaxScale.cnf file that defines the filter to load and the service to send the duplicates to.

```
[ArchieveFilter]
type=filter
module=tee
service=Archieve
```

In addition parameters may be added to define patterns to match against to either include or exclude particular SQL statements to be duplicated. You may also define that the filter is only active for connections from a particular source or when a particular user is connected.

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 updating 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 users 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 form 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 address matching in connections rules. This is also in line with the current protocol modules that do not support IPV6.

Partial address matching, such as 10.% is also not supported in the current version of MaxScale.

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.

Troubleshooting

MaxScale binds on TCP ports and UNIX sockets as well.

If there is a local firewall in the server where MaxScale is installed, the IP and port must be configured in order to receive connections from outside.

If the firewall is a network facility among all the involved servers, a configuration update is required as well.

Example:

```
[Galera Listener]
type=listener
address=192.1681.3.33
port=4408
socket=/servers/maxscale/galera.sock
```

TCP/IP Traffic must be permitted to `192.1681.3.33 port 4408`

For Unix socket, the socket file path (example: `/servers/maxscale/galera.sock`) must be writable by the Unix user MaxScale runs as.