QLOADERD USER MANUAL

Loway

Revision History

Revision $Revision: 1.11 $ - covers version 1.29

$Date: 2012/12/21 11:07:15 $

L

Table of Contents

Loading the queue_log file into MySQL with Qloaderd

Note

this is not a tutorial on how to use MySQL storage in QueueMetrics. Such a topic is covered in detail in QueueMetrics' User Manual

Warning

qloaderd 1.21 and newer requires QueueMetrics 1.7.0 or newer.

What is Qloaderd?

Qloaderd is a small script that uploads queue_log data into a MySQL database for further analysis by http://queuemetrics.com . It is designed to be used instead of the older queueLoader.pl script, and offers a number of advantages over the older version:

- qloaderd is designed to be a one-stop solution to upload data. No need for tail or custom startup scripts.
- qloaderd is able to upload only data that is missing from the given partition, by checking the timestamp and the row's contents in order to prevent duplicate rows. This way every time it is started, it will update the MySQL table with all data that is missing from it and then will start tailing the queue_log file for variations.
- In the case of a database problem or a disconnection, qloaderd will keep trying to connect to the database until it succeeds. No data is lost if the database goes offline for a while.
- qloaderd will automatically detect invalid queue_log lines and will skip them immediately
- qloaderd keeps a detailed log of all its activity
- qloaderd can be installed as a service using a simple startup script
- In order to avoid service disruptions due to a lost database connection and to confirm to QueueMetrics that the connection is alive, qloader will periodically insert heartbeat information in the queue_log database.
- qloader will automatically detect when the queue_log file is rotated

Installing qloaderd

To install qloaderd, copy the file qloader.pl to a location of your choice (we suggest /usr/local/qloader). Run the following commands to make sure the file is executable:

```
dos2unix qloader.pl
chmod a+x qloader.pl
```

If you run a yum-based Linux distro, you can also have qloaderd installed automatically by issuing:

```
wget -P /etc/yum.repos.d http://yum.loway.ch/loway.repo
yum install qloaderd
```

You will still have to update it manually in order to configure it correctly.

You now have to edit the qloader.pl file in order to set up your database connection, by changing the following parameters:

```
my $mysql_host = "myserver.mylan";
my $mysql_db   = "queuemetrics";
my $mysql_user = "username";
my $mysql_pass = "password";
```

The qloader script expects three parameters to run:

- The first parameter is the full path to the queue_log file it has to upload
- The second parameter is the partition into which it will upload data
- The third parameter is the log file it will create when running

To test that the qloader script is working, try the following command:

```
./qloader.pl /var/log/asterisk/queue_log P02 /var/log/asterisk/qloader.log
```

This command will try to upload the contents of the file */var/log/asterisk/queue_log* to partition *P02* of the chosen database, writing its own log on */var/log/asterisk/qloader.log*. When the program is running no output is written on stdout; if you want to know how the loading is going you have to consult its own log file.

Optionally, you can pass along one or more of the following command line parameters before the mandatory parameters:

- -h *hostname*: the MySQL hostname to connect to
- -d *database*: the MySQL database to connect to
- -u *user*: the MySQL username to use
- -p *pass*: the MySQL password to use

The parameters above override the settings edited in qloader.pl. Please note that it's not advisable to pass the MySQL password on the command line.

Starting qloaderd init-style on RH-based systems

We provide a script (qloaderd) that can be put into /etc/init.d in order to start and stop qloader as a service and have it start automatically when the machine boots. You will find it in the *Redhat-style-initscripts* directory. The file is called qloaderd and must be put into /etc/init.d on the Asterisk server. It was made for RHEL/CentOS/AAH/Trixbox, but it will likely work on most similar versions of Linux.

Run the following commands to make sure the file is executable:

```
dos2unix /etc/init.d/qloaderd
chmod +x /etc/init.d/qloaderd
```

You then need to create a configuration file under /etc/sysconfig/qloaderd that will lok like the following one:

```
PARTITION=P001
QUEUELOG=/var/log/asterisk/queue_log
LOGFILE=/var/log/asterisk/qloaderd.log

LOCKFILE=/var/lock/subsys/qloaderd
PIDFILE=/var/run/qloaderd.pid

MYSQLHOST=localhost
```

```
MYSQLDB=queuemetrics
MYSQLUSER=queuemetrics
MYSQLPASS=javadude
```

You usually only need to modify the PARTITION and maybe QUEUELOG entries.

At this point, you can start the qloader simply by typing */etc/init.d/qloaderd start* and stop it by typing */etc/init.d/qloaderd stop*. As qloader is able to upload only missing data, no data is lost if you stop it.

In order to have the service started on boot, use the command:

```
chkconfig --add qloaderd
```

As a default, it starts automatically in runlevels 2,3,4 and 5. You can easily change that with chkconfig.

Starting qloaderd init-style on other systems

We also provide a plain-vanilla startup script that can be installed in your local startup directory in order to start Qloader automatically. It may require a bit of tweaking, but it will run on all Linux distributions. You will find it in the *Other-initscripts* directory. Follow the same instructions as for a RedHat based system to make the script executable and configure it.

Advanced topics

The following features of Qloaderd can be enabled or disabled, as required. Most people will not need them.

Heartbeat

Qloaderd can insert a *heartbeat* log entry in order to make sure that:

- The MySQL channel stays open all the time, even when there is no activity on the Asterisk side. This is useful because MySQL disconnects the DB connection after a given channel inactivity (usually 8 hours) and sometimes reconnecting a dead channel will not work.
- On the QueueMetrics side, it is possible to know immediately if one or more members of a cluster have been disconnected, as the heartbeat information will be present even if Asterisk is idle

This option is turned on by default, and is controlled by the configuration option:

```
my $heartbeat_delay  = 15 * 60;
```

This means the heartbeat is sent every 15 minutes after the queue_log is idle. If you would like to turn this off, set it to 0.

Subqueues

If you work in an environment where a single queue is used to service a number of different agents, it might be of interest to you to enable subqueues; i.e. to have the queue name rewritten appending a client code at the end, so that you can query them from QueueMetrics as if each client was serviced by a dedicated queue.

In order to activate this, you must:

- Set the line:

  ```
  my $use_subqueue = 1;
  ```

- If you optionally add a comment to the client code after the "/" symbol, you should set

  ```
  my $split_subq_name = 0;
  ```

  to have the comment clipped off. This is useful if, for debugging reasons, you would like the client code to be "1234/Red", where "1234" is the client code and "Red" is a human-readable comment.

- Create a table called *qlog_opencalls* in the QueueMetrics database, using the following schema:

  ```
  CREATE TABLE  `qlog_opencalls` (
    `rowId` int(10) unsigned NOT NULL auto_increment,
    `partition` varchar(20) NOT NULL default '',
    `callId` varchar(45) NOT NULL default '',
    `queue` varchar(45) NOT NULL default '',
    `subqueue` varchar(45) NOT NULL default '',
    `lastVerb` varchar(45) NOT NULL default '',
    `lastTst` int(10) unsigned NOT NULL default '0',
    `lastUpd` datetime NOT NULL default '0000-00-00 00:00:00',
  ```

```
  PRIMARY KEY  (`rowId`),
  KEY `accIdx1` (`callId`,`partition`)
) TYPE=MyISAM;
```

If you use a recent version of QueueMetrics, the table will already be present in the database.

- Whenever you pipe a call to the Queue() command in Asterisk, set the client code in the URL parameter of the command, like for example in:

  ```
  exten => s,8,Queue(queue-service|t|1234||180)
  ```

  Here the queue *queue-service* is called with a client code of *1234*; the resulting queue activity will be seen in QueueMetrics as if it was done on queue *queue-service.1234*.

Please note that you should mantain the *qlog_opencalls* table yourselves, periodically deleting rows with a *lastUpd* timestamp over one day old and optimizing it as needed (this will be done automatically in future revisions of Qloaderd).

To achieve this result, just run the following queries through a cron job in a moment when the system is idle or is experiencing a low load:

```
DELETE FROM qlog_opencalls WHERE lastUpd &lt;  DATE_SUB( NOW(), INTERVAL 48 HOUR);
OPTIMIZE TABLE qlog_opencalls;
```

As we delete a potentially large number of rows at once, running the OPTIMIZE saves space and improves index performance as well. The downside is that it could lock the table for a few seconds.

When qloader imports a call and does rewriting, the URL field in the queue_log will always be set to blank.

Agent rewriting

Note

It would be advisable to use *Regexp-based rewriting* instead, as it is more flexible.

It is possible to perform advanced channel rewriting to the *Agent/XXX* format. This is done on data loading and does not require QueueMetrics to do it again and again every time it loads data.

To enable this feature, set:

```
my $rewriteToAgent   = 1;   # 0 no; 1 yes
my @channelsToAgent  = ( 'Local', 'SIP' );
```

The configuration above will mean that channels like SIP/1234 will be rewritten to Agent/1234, while Local/2345@agents will be Agent/2345.

Database-driven agent rewriting

It is possible to use a database table to store channel rewriting rules for maximum flexibility. This is often used e.g. when using Asterisk Realtime and still wanting Agent/xxx and not SIP/yyy, where you keep track of agent xxx being dynalmically logged on to SIP/yyy.

To enable this feature, set:

```
my $dbAgentRewrite   = 1;
```

You should also create a table in the QueueMetrics database with the following definition:

```
CREATE TABLE `qlog_rewrite` (
  `ag_from` varchar(50) NOT NULL,
  `ag_rewritten` varchar(50) NOT NULL,
  `last_upd` datetime NOT NULL,
  PRIMARY KEY  (`ag_from`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

Enter the rewrite rules as tuples *(ag_from, ag_rewritten)*. The *last_upd* key is not used and only given as an hint of very old entries that might need deletion. Before enabling this mode, please consider that a query will be performed for *nearly every line* of the queue_log that is being processed. Results are not cached, as the ability to change the underlying table while qloaderd is running is considered a feature.

Regexp-based rewriting

Starting from qloaderd 1.23, it is possible to create a set of regexp-based substitution rules that make qloaderd very flexible in applying arbitrary text substitutions. This allows to perform nearly any possible text substitution.

This model is based on the power of Perl's regular expression to match and replace text. As such expressions may become quite complex, they are meant to be stored in a separate file called *qloader_regexp.pl* that is in the same directory as qloaderd.

This is an example that shows one of such rules in action:

```
{
        rule => "PBX SIP channels",
        from => '^(\d+?\|.+?\|.+?)\|SIP\/(.+?)-(.+?)\|(.+)$',
        replace => '\1|SIP/\2-\3-rm|\4',
        final => 0
},
```

This rule, that has an arbitraty name *PBX SIP channels*, tries to rewrite the channel name to *SIP/123-45-rm* if it is *SIP/123-45*.

Notes:

- Rules are applied in order, from the first to the last, unless *final* is set to 1; in which case, the current result is returned withuot other rules being applied
- All regexp matching is case-insensitive
- It is a good idea to use line-start (^) to line-end ($) qualifiers to avoid spurious matches.
- In order to replace captured sequences in the *replace* expression, you must use the tokens *\1*, *\2* … *\9* to replace the contents of the first, second…. ninth capture.
- The file *qloader_regexp.pl* is a common Perl file; its common syntax rules apply.
- In case the regexp rules contain any errors, the whole file is simply ignored.

As regular expressions can be tricky to program and debug, we offer a script called *regexp_shell.pl* that prints the set of current regexps and applies it to the input received on STDIN.

```
./regexp_shell.pl < myfile.in
```

This will print out the results of translating "myfile.in" with the given set of rules. If the rules file is invalid, the shell will refuse to load.

Debugging qloaderd

If you suspect problems with qloaderd:

- First of all, have a look at its own log file.
- Try running a queue and check that lines get loaded into MySQL in real-time. The following query will often be helpful to count all elements in your queue_log table:

```
select partition, queue, count(*) as n_records
from queue_log
group by partition, queue
order by partition, queue
```

Note

This query runs a full table scan, so it may take a long time to complete if you have million of lines loaded.

- Run the real-time page in QueueMetrics

qloader and queue_log rotation (Obsolete)

Note

since qloaderd 1.21, qloaderd automatically detects file rotations and picks them up immediately, no manual notification is necessary.

As qloader keeps reading the queue_log file, it must be notified when the queue_log file is being rotated. Sending it a */etc/init.d/qloaderd restart* after the rotation has taken place will ensure that qloader starts loading data from the new queue_log file. It would be a nice idea to rotate its own log file too.

You could modify the */etc/logrotate.d/asterisk* file as follows:

```
/var/log/asterisk/*log {
```

```
    missingok
    rotate 5
    weekly
    create 0640 asterisk asterisk
    postrotate
        /usr/sbin/asterisk -rx 'logger reload' > /dev/null 2> /dev/null
        /etc/init.d/qloaderd restart > /dev/null 2> /dev/null
    endscript
}
```

Getting help

If you need further help in setting up qloaderd….

- First of all, have a look at the QueueMetrics FAQ [http://queuemetrics.com/faq.jsp]
- Then try the QueueMetrics forum [http://forum.queuemetrics.com/faq.jsp]
- Feel free to contact us [http://queuemetrics.com/contact.jsp].

Automatically load all queue_log files

As the queuePartialUpdater script is not very easy to use, we created a wrapper that is able to upload a series of broken up queue_log files that are stored under /var/log/asterisk as queue_log.* as it is usual for rotated queue_log files.

Configuration

You should edit your configuration in /etc/sysconfig/qloaderd - it is the same file that is used by the main qloaderd application, so if it works, this loader will work as well with no configuration changes.

How to use it

First make sure that it is installed:

```
yum list | grep qloaderd
```

You have to make sure that you have at least version 1.19 of the qloaderd.

If not, you can easily upgrade with:

```
yum update qloaderd
```

Then run the following commands:

```
cd /usr/local/qloader/

/etc/init.d/qloaderd stop

./loadAllQueueLogFiles.sh

/etc/init.d/qloaderd start
```

It should print a very detailed log on the stdout. As it tries to optimize the database table, this command should NOT be run when QueueMetrics is being actively used, also, new data will not be loaded when the command is running.

After this command terminates, you should make sure that you turn the queue_log file rotation (see above).

Data Queue Partial Update

The script is able to update the queues data activity already stored in the QueueMetrics database with a new data coming from a different queue_log. It's useful for maintaining corrupted (or restoring) databases.

This script works in pair with the qloader.pl script **with revision number equal or greater than 1.12**.

Configuration

In order to use it, the script has to be customized with information related to the database. The relevant configuration keys are:

```
my $mysql_host = "127.0.0.1";
```

```
my $mysql_db   = "queuemetrics";
my $mysql_user = "queuemetrics";
my $mysql_pass = "javadude";

my $qloaderbin = "./qloader.pl";

my $logfile    = "queuepartialupdater.log";
```

More in detail:

- $mysql_host: is the server where the QM database is located
- $mysql_db: is the QM database name
- $mysql_user: is a QM database user with read/write permissions
- $mysql_pass: is the password related to the below-specified user
- $qloaderbin: is the full path where the qloader.pl script could be reached
- $logfile: is the file where the log will be written

You can pass the following parameters to avoid changing the file itself:

- -h *hostname*: the MySQL hostname to connect to
- -d *database*: the MySQL database to connect to
- -u *user*: the MySQL username to use
- -p *pass*: the MySQL password to use

How to use it

The script can be called via the command line by specifying:

- The queue data file to be read
- The partition to be updated
- The queue log data file (the one the qloader script will use).

The script will be responsible for removing all data overlapping with the period specified in the input queue data file for the defined partition. It will then be responsible for calling the qloader script. That script will be able to fill the database with the new information provided by the same queue data log. At the end of the processing, the log specified by the comnmand line will report the result of the new data filling. The log specified by the configuration, instead, will list the actions performed by the queuePartialUpdater script in order to correct the database.

An example of usage is:

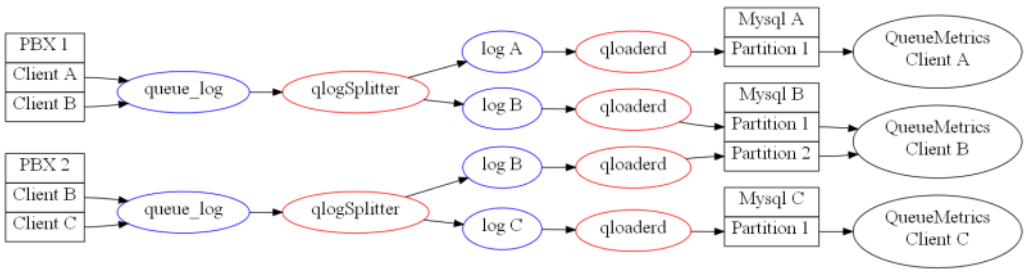
```
./queuePartialUpdater.pl /var/log/asterisk/queue_log P001 /var/log/qlog.log
```

Feeding multiple QM instances from a single queue_log file

Is is a common scenario for hosted, cloud-based and multi-tenant systems that you have a set of PBX's where you run queues for multiple clients. You would like to deploy a separate QueueMetrics instance for each client, maybe using the Loway Keyring as a pay-as-you go licensing model.

The **qlogSplit** is a simple tool that does the preprocessing of the queue_log file produced by the shared Asterisk instance and creates separate queue_log files, each of which will be handled by their separate qloader instance.

A typical usage scenario looks like the following picture:



In this case we have a virtual PBX where the queues of client A are processed on PBX 1, the queues of client B are processed on both PBX 1 and 2, and client C is processed only on PBX 2.

What we do is to add a single qlogSplit instance per each Asterisk system that will create a separate queue_log file for each client and will launch a qloader instance tailored to uploading data in a separate pertition of a database. Each database is used by a separate QueueMetrics instance.

Each instance of a client on a PBX is a separate **system**.

Prerequisites

The prerequisites of running qlogSplit are that you create agents and queues in a way that the client can be recognized, ideally postfixing them with the client's ID. So you would have e.g SIP channels in the format "SIP/123-client1" and queues in the format "sales-client1". This is the way mosty multi-tenancy environments for Asterisk handle multi-tenat systems.

The queueSplitter is NOT compatible with running a single qloader instance in "classic" mode, that is manually run. All qloader instances must be launched and terminated by itself.

How does queueSplitter work?

In order to run queueSplitter, you have to configure it by editing the QlogSplitter.pm file (see below).

After that, you simply launch it as in:

```
nohup nice perl qlogSplit.pl &
```

When running, it will:

- Stop all existing qloaders running for known systems
- Truncate their queue_log files
- Recreate their queue_log files (this may take a few seconds, based on the size of the existing queue_log file)
- Restart tthe qloaders
- Feed data to the correct system's queue_log file as it is appended

As the splitter has to uderstand to which system each line of the queue_log belongs (as per your configuration), in case it should not understand some data it will log it to a queue_log file marked *UNKNOWN* for your inspection.

In order to terminate the splitter, just kill the process.

Files created (all paths can be edited):

- /var/log/asterisk/qlogSplitter.log is the activity log of the splitter
- /var/log/asterisk/split-qlog-#.txt - the queue_log rebuilt for system #. If you have e.g. systems A and B, you might find split-qlog-A.txt, split-qlog-B.txt and possibly split-qlog-UNKNOWN.txt
- /var/log/asterisk/logs/qloader-#.log - the activity log of the qloader running for system #

Configuration

You must edit the file QlogSplitter.pm. It has an initial preamble specifying some general information of the running systems:

```
$LOGFILE     = "/var/log/asterisk/qlogSplitter.log";
$SRC         = "/var/log/asterisk/queue_log";
$QLOG        = "/var/log/asterisk/split-qlog-#.txt";

$QLOADER_BIN = "/usr/local/qloader/qloader.pl";
$QLOADER_LOG = "/var/log/asterisk/qloader-#.log";
```

Note that the QLOADER_BIN file must be callable (marked for execution and with a working Perl interpreter. You may need to run dos2unix to make it runnable).

Then you add a stanza in the $SYS section for each system, like in the following example:

```
'MAC' => {

  qloader => {
    host => '1.2.3.4',
    db => 'qm_mac',
```

```
    user => 'queuemetrics',
    pass => 'javadude',
    partition => 'P001'
  },
  'match_q' => qr/^(.+?)-mac$/i,
  'match_a' => qr/^(.+?)-mac$/i
},
```

This entry specifies a system called MAC (just short ASCII names, no spaces or symbols), which qloader uploads data to partition P001 the QueueMetrics database called "qm_mac" on the MySQL server located at 1.2.3.4, and sets a matching rule for queues and agents so that they must end in "-mac".

Note that each regular expression has a capture group - that is the name the channel of the queue that will be rewritten to. So on the QM system for MAC, a queue named "q1-mac" will appear as "q1", and an agent called "Agent/101-mac" will appear as "Agent/101".