WQLoader USER MANUAL

L.Emilitri, M.Signorini - Loway

WQLoader USER MANUAL

L.Emilitri, M.Signorini - Loway

Table of Contents

Chapter 1. Web Qloader - A short user guide

The web qloader (*wqloader* for short) is a daemon written in Perl that uploads `queue_log` events to a QueueMetrics instance over HTTP.

This has several advantages over the plain MySQL protocol used by the old *qloaderd* tool, especially for physically distant systems or hosted systems - the QueueMetrics HTTP port already has to be exposed, and you do not need to handle TCP connections to MySQL. Centralizing data loading into QueueMetrics also acts as a "choke point" to apply further security restrictions. Additionally, the wqloader offers automatic splitting of `queue_log` files in order to feed multiple separate QueueMetrics instances from the same Asterisk instance, so that you can host multiple independent clients on the same Asterisk system or cluster.

The *wqloader* was built for:

- Maximum data safety - your data is never lost, even in case of catastrophic failures on the net or on the receiving QueueMetrics instance
- High availability and continuous unattended operation
- Good performance even when having high or intermittent network latency
- Running in parallel with an existing qloaderd.

The *wqloader* works with all versions of QueueMetrics after 14.06.3.

Chapter 2. Installation

2.1. Prerequisite: a valid QueueMetrics user

In order to run *wqloader*, you need to have a valid user in QueueMetrics holding the key WQLOADER that must be used for remote authentication.

Recent versions of QueueMetrics already include a user named "webqloader" with password "qloader". It is disabled by default, so it has to be manually enabled from the GUI - select "Edit settings", then "Edit users", edit it and sent "Enabled" to "Yes". While you are at it, make sure you change the default password.

If your QueueMetrics instance does not have such a user defined, create a new user with minimum privileges and give it the key WQLOADER. Test it by logging on and off manually.

2.2. Automated yum installation

On systems that run CentOS and derived systems, including at the moment the all-popular AsteriskNOW, FreePBX and Elastix systems, the *wqloader* can be installed by issuing the following commands as root:

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

If the installation fails on older CentOS 5 systems, you may have to install a missing package manually (you could need to disable gpgcheck from yum's configuration), such as in:

```
wget http://pkgs.repoforge.org/perl-JSON/perl-JSON-2.50-1.el5.rf.noarch.rpm
yum localinstall perl-JSON-2.50-1.el5.rf.noarch.rpm
```

When the runner starts, you will have to edit its configuration located in `/etc/sysconfig/wqloaderd` to point it to your QueueMetrics instance and then restart it.

A valid configuration file will look like:

```
HTTPHOST=127.0.0.1
HTTPPORT=8080
HTTPCONTEXT=queuemetrics
QMUSER=webqloader
QMPASS=qloader

PARTITION=P001
QUEUELOG=/var/log/asterisk/queue_log

LOGFILE=/var/log/asterisk/wqloaderd.log
LOCKFILE=/var/lock/subsys/wqloaderd
PIDFILE=/var/run/wqloaderd.pid
```

You usually have to edit the HTTP parameters and the user name and password.

2.2.1. Starting and stopping

To start the wqloaderd you simply type:

```
/etc/init.d/wqloaderd start
```

And to stop it you type:

```
/etc/init.d/wqloaderd stop
```

You may also force a restart (e.g. after a change of configuration) by issuing:

```
/etc/init.d/wqloaderd restart
```

When you install it using *yum*, the *wqloader* is scripted to start automatically on boot.

2.2.2. Monitoring

To check that the system is running as expected, just enter:

```
tail -f /var/log/asterisk/wqloaderd.log
```

It will print out something like:

```
 |Thu Aug  7 09:30:56 2014|QueueMetrics Web loader - $Revision: 1.7 $
 |Thu Aug  7 09:30:56 2014|Evaluating wqloader_regexp.pl
 |Thu Aug  7 09:30:56 2014|Evaluating wqloader_cluster.pl
 |Thu Aug  7 09:30:56 2014|Found 1 cluster rule(s).
 |Thu Aug  7 09:30:56 2014|PID 1068 - Token: P001 - TZ offset: 0 s. - Heartbeat after 90
 |Thu Aug  7 09:30:56 2014|PID 1068 - Target host URI: demo.queuemetrics-live.com:80/mysy
 |Thu Aug  7 09:30:56 2014|Ignoring all timestamps below 0
```

It will also show any errors it should encounter while running. In general, when the loader encounters any error, it just waits a few seconds and retries.

2.3. Manual installation

The general procedure you should follow is:

- Download and unpack the latest version of *wqloader* from the downloads section of the QueueMetrics website.
- Make sure all Perl requisites are met
- Create a script to run the *wqloader* passing all required parameters (see the *Configuring* section)
- Make sure the script runs on boot.

2.3.1. Perl prerequisites

Some perl extra packages are needed but they usually are available through yum installer on CentOS distributions. Here is the list of needed extra packages (fon CentOS 5 / 6):

```
perl-MIME-tools
perl-JSON
```

On Debian/Ubuntu systems, dependencies are available from:

```
libwww-perl
libmime-tools-perl
libjson-perl
```

2.3.2. Example: installing on Ubuntu 14.04

The following sequence downloads and installs *wqloader* on an Ubuntu 14.04 server system.

```
apt-get install libwww-perl libmime-tools-perl libjson-perl

wget http://downloads.loway.ch/qm/wqloaderd-1.4.tar.gz
tar zxvf wqloaderd-1.4.tar.gz
cd wqloaderd-1.4/
chmod a+x wqloaderd.pl
./wqloaderd.pl -h cluster.queuemetrics-live.com -p 80 -c system13 \
          -u robot -w hello /var/log/asterisk/queue_log P001 -
```

See below for the meaning of configuration parameters.

2.4. Configuring the loader

Depending on the user needs, the loader runs in two different modes: basic and advanced. The basic mode is targeted at small single user systems where the advanced mode is targeted at large multi-tenant systems.

2.4.1. A bit of terminology

You define a QueueMetrics system to be a "target" for receiving data by defining:

- a *Host*: the name or IP address of the machine that runs QueueMetrics
- a *Port*: the TCP port QueueMetrics listens on
- a *Context*: the name of the "folder" QueueMetrics appears to be in

Then you would define:

- a *User*: a valid QueueMetrics user holding the key WQLOADER
- a *Password*: the access password for the user above
- a *Partition* (or *Token*): a logical "place" in the QueueMetrics database that your data will be written under. Each Asterisk instance MUST be uploaded to a separate partition, or subsequent data analysis will be incorrect.

2.4.2. Running in Basic mode

Running the loader in basic mode is quite simple. When running manually, all parameters should be specified from the command line. The valid parameters are:

```
Known flags:
  -h host       : host name (default: 127.0.0.1)
  -p port       : port number (default: 8080)
  -c context    : application context (default: queuemetrics)
  -u user       : user name (default: webqloader)
  -w password   : password  (default: qloader)

wqLoaderd.pl [flags] /my/queue_log/file partition_name /my/activity/log
```

Flags not explicitly defined are set to the default values specified above.

As example, pushing data into partition P001 to a *cluster.queuemetrics-live.com* host, running on port *80* with context *client4* with default (robot,robot) username and password, and logging to standard output would create a command line like:

```
./wqloaderd.pl -h cluster.queuemetrics-live.com -p 80 -c client3 \
                                /var/log/asterisk/queue_log P001 -
```

2.4.3. Running in Advanced mode

By running in Advanced mode, you can feed multiple QueueMetrics instances from one single Asterisk box. This way, each instance can be in a physically separate location and can be configured and possibly resold independently.

In order for this to work, the prerequisite is that you use a common naming schema for all agents and queues, be it "queue1-client1" and "Agent/100-client1" or "client1-queue1" and "Agent/client1-100". Then you tell *wqloaderd* which prefixes or suffixes apply to each QueueMetrics instance, and give it all the details to connect to it.

You do this by configuring a set of rules by editing the file `wqloader_cluster.pl` that resides in the same folder where the main script is stored.

Within the `wqloader_cluster.pl` file is a set of structures that are to be uncommented and/or edited according to your needs. For each defined rule, the loader starts a process that will be in charge of pushing data to the defined host for that rule. There is no limit on the number of rules that can be defined (except the CPU and RAM amount needed to handle all the processes that will be generated).

Each rule defines a set of filtering criteria. Two types of filters are availables: filters by queue and filters by agent. The filters are applied to each loader row as a *first match* (iptables-style) starting by queue information. Filters are to be individually enabled through configuration.

Rows without queue or agent information (i.e. containing NONE on these fields) are pushed to all servers. This allows to *split* a single multi-tenant `queue_log` into several single-tenant QueueMetrics instances.

A simple example of single rule is reported below:

```
{
    ruleName => "Queues starting with 'Customer-'",
    queueStarts => 1,
    queueEnds => 0,
    queueToken => 'Customer-',
    agentStarts => 0,
    agentEnds => 0,
    agentToken => '',
    targetHost => "10.10.5.110",
    targetPort => "8084",
```

```
    targetCtx => "customer",
    targetUser => "robot",
    targetPass => "robot",
    targetToken => "P08",
    timezoneOffset => "3600",
},
```

This example generates a process able to handle queues where name is starting with "Customer-". Events associated to that queues, as either generic unassociated events, are sent to the `10.10.5.110:8084/customer` URL. In this case, *queueStarts* should be set to 1 and the *queueToken* should be populated with "Customer-". Tokens are case-sensitive.

A different rule with *queueStarts* set to 0 and *queueEnds* set to 1 will populate only queues with names ending in "Customer-". Filtering by agents requires to properly set the keys agentStarts/agentEnds/agentToken in the same way.

As soon as the configuration file is set up, the loader will be run by issuing the following command:

```
./wqloader.pl /var/log/asterisk/queue_log X -
```

The parameters are taken from the configuration file. In this example a dummy parameter (*X* in this case) is needed as a placeholder if you want to specify a custom output log.

When the default output file `/var/log/asterisk/qloader.log` is used, there is no need to specify any additional command line parameters:

```
./wqloader.pl /var/log/asterisk/queue_log
```

Chapter 3. The network protocol API

Conversations between *wqloader* and QueueMetrics used JSON as high level protocol. In this section is a list of JSON vectors involved in the process.

The Web Qloader process is responsible for starting all transactions. Each transaction is wrapped into a single HTTP POST with Basic Authentication targeted to an end-point named `jsonQLoaderApi.do` .

QueueMetrics does not support the standard Basic Authentication method but a slightly modified version. Where standard Basic Authentication requires to have a first unauthenticated HTTP transaction followed by a *HTTP 401 Not Authorized* server response code, and a second client HTTP request containing authorization headers, QueueMetrics requires to have a single client transaction with authorization headers already present.

3.1. JSON Vectors fields

3.1.1. Common fields

All JSON vectors contain some common fields used by QueueMetrics and/or the Web Qloader to persist information like, for example, the protocol version and answer results and statuses.

- **commandId**: a string containing the command identifier. Valid values are specified in the table below
- **version**: a string containing current protocol version. Valid values are: "1.0"
- **resultStatus**: a string containing the operation result status. Valid values are: "OK" or "KO: " followed by an error message description
- **result**: a string containing the result of the current operation. Valid values are specific for each operation type. Values associated to this field should be discarded for events marked as `KO` in *resultStatus*

commandId	See paragraph	Since protocol version
checkHWM	Check High Watermark	1.0
checkRow	Check Row	1.0
insertRow	Insert Row	1.0
checkHb	Check Heartbeat	1.0

3.1.2. Check High Watermark

This entry asks for the maximum timestamp already available on remote database for the specified token.

QueueMetrics expects to receive a JSON vector containing the following fields:

- **commandId**: a string populated with *checkHWM* token. See Section 3.1.1, “Common fields”
- **version**: see Section 3.1.1, “Common fields”
- **token**: a string containing the name of the token identifying the partition to be used in QueueMetrics

QueueMetrics answers with a JSON vector containing the following fields:

- **commandId**: a string populated with *checkHWM* token. See Section 3.1.1, “Common fields”
- **name**: a string containing a user readable name for this vector. In this case is *Check High WaterMark*
- **version**: see Section 3.1.1, “Common fields”
- **resultStatus**: see Section 3.1.1, “Common fields”
- **result**: a string containing the highest timestamp found in the database for the specified token or the string *null* if no data are present

3.1.3. Check Row

This entry asks for the presence of a `queue_log` data line in the database.

QueueMetrics expects to receive a JSON vector containing the following fields:

- **commandId**: a string populated with *checkRow* token. See Section 3.1.1, “Common fields”
- **version**: see Section 3.1.1, “Common fields”
- **token**: a string containing the name of the token identifying the dataset (actually: partition) set in QueueMetrics
- **timeId**: a string containing the row data timestamp as reported by asterisk

- **callId**: a string containing the asterisk unique id for the row data
- **queue**: a string containing the queue name field for the row data
- **agent**: a string containing the agent field present in the row data
- **verb**: a string containing the verb field present in the row data
- **parameters**: a string array containing other unnamed parameters present in the row data. The expected array size is 5 meaning that parameters not available should be always present as empty strings.

QueueMetrics answers with a JSON vector containing the following fields:

- **commandId**: a string populated with *checkRow* token. See Section 3.1.1, “Common fields”
- **name**: a string containing a user readable name for this vector. In this case is *Check Exsisting Row*
- **version**: see Section 3.1.1, “Common fields”
- **resultStatus**: see Section 3.1.1, “Common fields”
- **result**: a string containing the number of matching data lines found in the database

3.1.4. Insert Row

This entry asks for the insertion of a `queue_log` data line in the database.

QueueMetrics expects to receive a JSON vector containing the following fields:

- **commandId**: a string populated with *insertRow* token. See Section 3.1.1, “Common fields”
- **version**: see Section 3.1.1, “Common fields”
- **token**: a string containing the name of the token identifying the dataset (actually: partition) set in QueueMetrics
- **timeId**: a string containing the row data timestamp as reported by asterisk
- **callId**: a string containing the asterisk unique id for the row data
- **queue**: a string containing the queue name field for the row data
- **agent**: a string containing the agent field present in the row data
- **verb**: a string containing the verb field present in the row data
- **parameters**: string array containing other unnamed parameters present in the row data. The expected array size is 5 meaning that parameters not available should be always present as empty strings.

QueueMetrics answers with a JSON vector containing the following fields:

- **commandId**: a string populated with *insertRow* token. See Section 3.1.1, “Common fields”
- **name**: a string containing a user readable name for this vector. In this case is *Insert Row*
- **version**: see Section 3.1.1, “Common fields”
- **resultStatus**: see Section 3.1.1, “Common fields”
- **result**: a string containing a not null internal technical id associated to this event

3.1.5. Check Heartbeat

This entry asks for an insetion of an heartbeat data line in the database. Heartbeat events should be sent periodically to QueueMetrics to allow remote process survey. The interval between different Heartbeat events is not defined but suggested as 900 seconds.

QueueMetrics expects to receive a JSON vector containing the following fields:

- **commandId**: a string populated with *checkHb* token. See Section 3.1.1, “Common fields”
- **version**: see Section 3.1.1, “Common fields”
- **token**: a string containing the name of the token identifying the dataset (actually: partition) set in QueueMetrics
- **timeId**: a string containing the timestamp of this query (in seconds starting from 01/01/1970)

QueueMetrics answers with a JSON vector containing the following fields:

- **commandId**: a string populated with *checkHb* token. See Section 3.1.1, “Common fields”
- **name**: a string containing a user readable name for this vector. In this case is *Check Heartbeat*

- **version**: see Section 3.1.1, “Common fields”
- **resultStatus**: see Section 3.1.1, “Common fields”
- **result**: a string containing a not null internal technical id associated to this event

3.2. HTTP Example

As example, below is shown a simple transaction dump.

Firts, the Web Qloader sends a POST query to QueueMetrics.

In this example we can see the authorization field containing the base64 encoding of the `username:password` pair and the COMMANDSTRING parameter populated with the HTTP encoded JSON object:

```
{ "commandId":"checkHWM",
  "version":"1.0",
  "token":"P08" }
```

The reply is:

```
POST /queuemetrics/jsonQLoaderApi.do HTTP/1.1
TE: deflate,gzip;q=0.3
Connection: TE, close
Authorization: Basic ZGVtb2FkbWluOmRlbW8=
Host: 10.10.5.110:8084
User-Agent: libwww-perl/5.833
Content-Length: 106
Content-Type: application/x-www-form-urlencoded

COMMANDSTRING=%7B%22commandId%22%3A%22checkHWM%22%2C%22version%22%3A%221.0%22%2C\
                        %22token%22%3A%22P08%22%7D
```

QueueMetrics answers with the following JSON vector:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=94C0990E61F67440D95; Path=/queuemetrics/; HttpOnly
Content-Type: application/json; encoding="UTF-8";charset=ISO-8859-1
Content-Length: 119
Date: Thu, 07 Aug 2014 09:36:37 GMT
Connection: close


{  "commandId":"checkHWM",
   "version":"1.0",
   "resultStatus":"OK",
   "result":null,
   "token":null,
   "name":"Check High WaterMark" }
```

In this example the field **result** is populated with *null* meaning that there is no data for the required token.

If data are already present for the required token, the answer resembles to the following:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=F517168E427059693B73; Path=/queuemetrics/; HttpOnly
Content-Type: application/json; encoding="UTF-8";charset=ISO-8859-1
Content-Length: 127
Date: Thu, 07 Aug 2014 09:50:27 GMT
Connection: close


{ "commandId":"checkHWM",
```

```
 "version":"1.0",
 "resultStatus":"OK",
 "result":"1379435567",
 "token":null,
 "name":"Check High WaterMark" }
```

where the field **result** is populated with the timestamp value *1379435567* (that is Tue, 17 Sep 2013 16:32:47 GMT).

Chapter 4. Frequently Asked Questions (FAQs)

4.1. How do you handle multi-client clusters?

You may have two possible scenarios:

- You have multiple Asterisk servers, and each client is handled on one Asterisk server.
- You have multiple Asterisk servers, and each client may be handled by any machine on the cluster.

In the first case, you simply set up a one-to-many cluster as explained in the "Advanced mode" section of this manual.

In the second case, you need to set up a partition for each Asterisk server on the client's machine. As this data is processed by different Asterisk boxes, it is important that no data from two distinct boxes is sent to the same partition.