Loway

QueueMetrics

call center suite

QueueMetrics - Advanced Configuration Manual

Loway SA

Version 26.10, 2026/10/12: covers QueueMetrics 26.10

Table of Contents

Acknowledgements . . . 1
ViciDial integration . . . 2
- Prerequisites . . . 2
- Changes to QueueMetrics database . . . 2
- Changes to ViciDial . . . 3
- Changes to QueueMetrics . . . 4

Serving QueueMetrics through Apache . . . 6
- Prerequisites . . . 6
- Installing mod_jk . . . 6
- Configuration of Apache and mod_jk . . . 7
- Virtual host configuration . . . 8
- Creating Virtualhosts in Tomcat . . . 9
- Troubleshooting . . . 10

Serving QueueMetrics over Apache/SSL . . . 11
- Prerequisites . . . 11
- Configure the Name-Based SSL Virtual Hosts . . . 11
- Install the Apache HTTP Server and its SSL/TLS Module . . . 11
- Configure the global Apache Settings . . . 12
- Configure the global SSL/TLS Settings . . . 12
- Create DNS records . . . 13
- Create the Application Directories . . . 13
- Install the CRT, CSR, and KEY files . . . 13
- Configure the Virtual Hosts . . . 14
- Verify the Configuration . . . 15
- Setup the service for automated startup . . . 16
- Troubleshooting . . . 16
- Summary of Log Files used by Apache . . . 16
- Notes . . . 17
- Sources . . . 17

Installing QM as a ROOT webapp . . . 18
- Prerequisites . . . 18
- Model 1: using a ROOT webapp . . . 18
- Model 2: defining a root context . . . 19
- Changes to QueueMetrics . . . 19

Enabling GZIP compression in Tomcat . . . 20
- Prerequisites . . . 20
- Changes to Tomcat . . . 20
- Changes to QueueMetrics . . . 20

For further reading . . . 20
Advanced QueueMetrics monitoring . . . 21
Prerequisites . . . 21
Assessing memory problems . . . 21
Remote monitoring with VisualVM . . . 22
Database connection pooling . . . 27
Prerequisites . . . 27
Changes to the MySQL server . . . 27
Changes to QueueMetrics . . . 27
Changes to Tomcat . . . 28
Moving QueueMetrics to a different server . . . 32
Prerequisites . . . 32
Required steps . . . 32
Moving the queue_log table to InnoDB . . . 33
Prerequisites . . . 33
Changes to MySQL . . . 33
Changes to QueueMetrics . . . 34
Using Master-master database replication for strong high-availability . . . 35
Prerequisites . . . 36
Changes to MySQL . . . 36
Changes to QueueMetrics . . . 39
Native MySQL logging of queue_log data . . . 40
Prerequisites . . . 40
Separating audio recordings in a daily folder . . . 41
Prerequisites . . . 41
Archiving script . . . 41
Changes to QueueMetrics . . . 42
Making older files accessible . . . 42
Show ringing phones in the realtime page . . . 43
Using pre-purchased keys (PPK) . . . 45
Terminology used . . . 45
Converting a PPK into a License key . . . 45
Upgrading / downgrading a PPK . . . 47
Enabling log rotation in Tomcat . . . 48
Prerequisites . . . 48
The rotation file . . . 48
Misc changes . . . 49
Changes to QueueMetrics . . . 49
For further reading . . . 50
Tuning QueueMetrics memory settings . . . 51
Prerequisites . . . 51

Usage scenario . . . 51
Monitoring basics: Java Visual VM . . . 51
Troubleshooting: taking thread and memory dumps . . . 54
Final Settings . . . 54
Quick JVM cheatsheet . . . 55
CRM Integration with QueueMetrics . . . 57
Prerequisites . . . 57
Integration with SugarCRM . . . 57
Integration with VTigerCRM . . . 59
Securing QueueMetrics (Tomcat) With A SSL Certificate . . . 62
Introduction . . . 62
Instructions . . . 62
Removing obsolete Diffie-Hellman ciphers . . . 64
Offlining a part of the queue_log table . . . 65
Prerequisites . . . 65
Moving data to a temporary table . . . 65
Backing up the temporary table . . . 66
Restoring data . . . 66
Removing duplicate rows from the queue_log table . . . 67
Prerequisites . . . 67
Reality check . . . 67
Spotting duplicate rows . . . 68
Loading unique rows . . . 68
Cleaning up . . . 69
Fixing broken indexes on the queue_log table . . . 70
Does this apply to you? . . . 70
Prerequisites . . . 71
How it works . . . 71
Procedure . . . 71
Cleaning up . . . 72
Printing all QueueMetrics users and agents in one go . . . 73
Bulk renaming audio files . . . 74
Removing the initial character in new files . . . 74
Renaming old files to remove the plus character . . . 74
Setting up QueueMetrics WebRTC Softphone . . . 75
0. Introduction . . . 75
1. QueueMetrics TLS Setup . . . 75
2. FreePBX Extension Setup . . . 75
Creating Self-Signed SSL Certificate . . . 77
3. Enable DTMF Feature code . . . 79
Configuring QueueMetrics . . . 80

Wallboard Softphone . . . 85
QueueMetrics running with UTF-8 charset . . . 90
Running post-call satisfaction IVRs and pushing their results to QM . . . 92
Prerequisites . . . 92
Implementation . . . 92
Keeping track of current client IP for VNC monitoring . . . 95
Prerequisites . . . 95
Required steps . . . 95
Serving QueueMetrics through a NGINX proxy . . . 96
Prerequisites . . . 96
NGINX configuration . . . 96
Self-signed ssl certificate . . . 99
Redirect the requests from Tomcat to Nginx . . . 100
Troubleshooting . . . 100
Further developments . . . 100
Self-host audio recordings with AudioVault and Nginx . . . 101
Prerequisites . . . 101
Setting up the PBX server . . . 101
Creating the AudioVault service . . . 101
Setting up the Nginx service . . . 104
Opening port 4006 in FreePBX . . . 106
Setting up the QueueMetrics-Live instance . . . 107
Troubleshooting . . . 107
Understanding MySQL storage and clustering . . . 108
Enabling MySQL storage . . . 109
Understanding MySQL storage . . . 109
Installing Qloaderd . . . 109
Setting up QueueMetrics . . . 113
Running QueueMetrics to monitor a cluster of Asterisk servers . . . 117
Installing Qloaderd on the Asterisk servers . . . 117
Setting up QueueMetrics . . . 117
Tips and tricks for Qloaderd . . . 120
Checking how much data is in the database on a daily basis . . . 120
Optimizing queue_log access time . . . 120
Getting help . . . 121

Acknowledgements

We would like to thank the following people for their precious contributions:

- Matt Florell from the ViciDial Group, USA
- Matthew J. Roth of InterMedia Marketing, USA
- Jens von Bulow of Xantech, South Africa
- Rahul Rajan of M.H.Alshaya Co
- Emile Coetzee of Clarotech Consulting, South Africa. www.clarotech.co.za
- deobfuscate, USA
- Félim Whiteley, IE

ViciDial integration

ViciDial is an enterprise class, open source call center suite in use by many large call centers around the world.

VICIdial has a full featured predictive dialer. It can also function as an ACD for inbound calls, or closer calls coming from VICIdial outbound frontiers. It is capable of inbound, outbound, and blended call handling.

It can also be easily integrated with QueueMetrics.

For more information, see http://www.vicidial.com

ViciDial is a registered trademark.

Prerequisites

- A working ViciDial instance, version 2.0.4 or later

It is very important that all servers involved (be they for QueueMetrics or ViciDial or general Asterisk usage) are on the same time zone and time, aligned with sub-second precision by an NTP daemon. If this is not so, the setting may lead to data corruption and inaccurate reports.

In order to translate ViciDial data to QueueMetrics, the following conventions are used:

- The campaign_id in ViciDial is seen as the queue in QueueMetrics
- The user ID in ViciDial is prepended by "agent/" and translated to the agent code in QueueMetrics (e.g. user 123 appears as agent/123)
- The UniqueID for the call appears as Asterisk's unique id prepended with server_id field (e.g. 1-1170345123.1234)

In this example, we imagine that:

- The QueueMetrics server has IP 1.2.3.4
- The QueueMetrics database server has IP 1.2.3.5 and the QM database is called "queuemetrics"
- The ViciDial server has IP 1.2.3.6

Changes to QueueMetrics database

ViciDial and QueueMetrics work together by sharing the database.

You must log on to the QueueMetrics database and create a user for ViciDial to connect to it. We use a different username from the one QM uses so it is easy to monitor who is doing what.

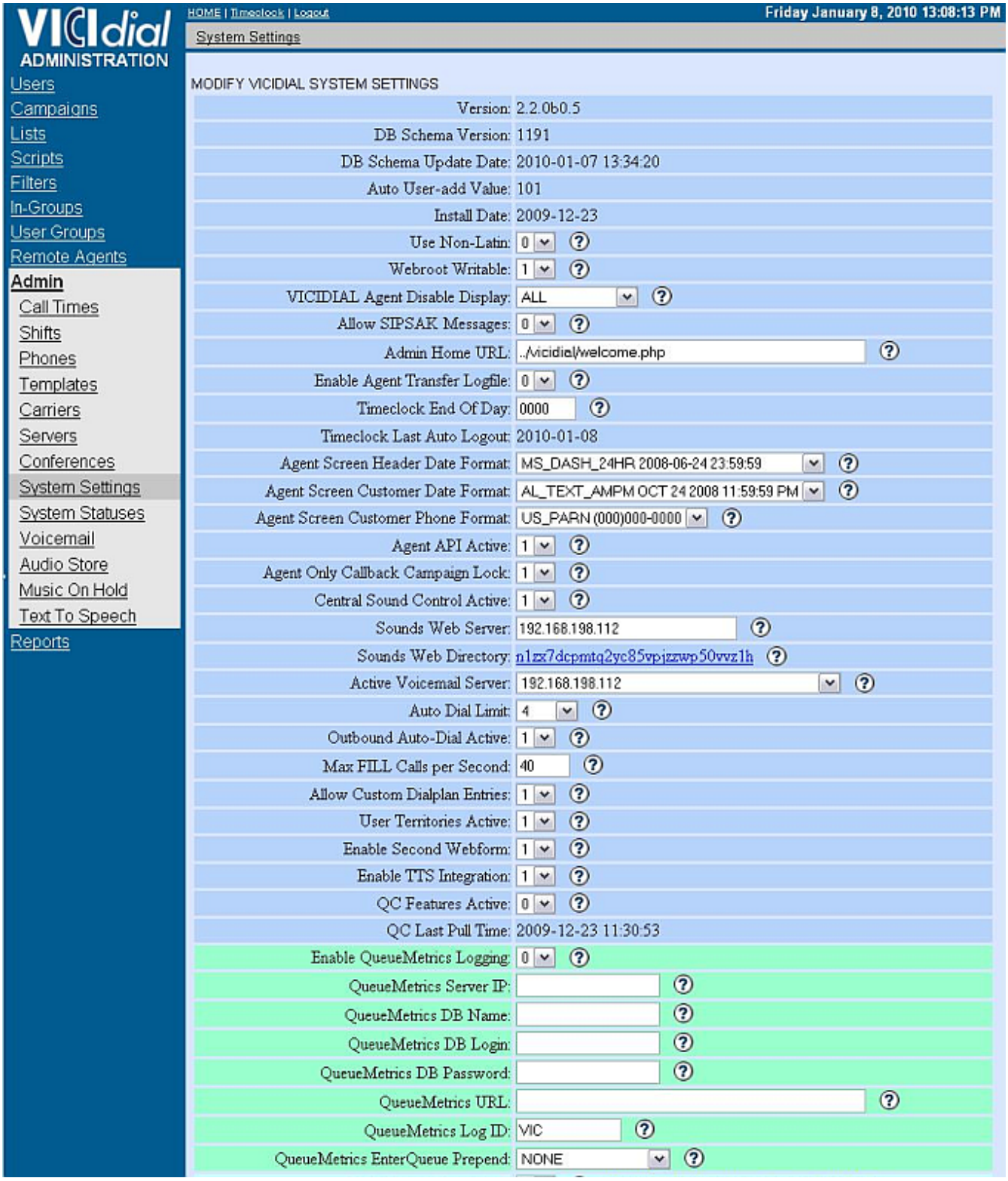
```
GRANT ALL PRIVILEGES ON queuemetrics.* TO vicidial@'1.2.3.6' IDENTIFIED BY 'qm';
```

ViciDial will also need special indexing on the *queue_log* table to work efficiently:

```
CREATE INDEX vici_time_id on queue_log(time_id);
CREATE INDEX vici_call_id on queue_log(call_id);
```

Changes to ViciDial

The system configuration can easily be set from the ViciDial Admin / System Settings page:



- Enable QueueMetrics logging: set to 1
- QueueMetrics server IP: this is the IP for the MySQL DB server, in our example "1.2.3.5"
- QueueMetrics DB name: the database name, in our example "queuemetrics"
- QueueMetrics DB login: the database login, in our example "vicidial"
- QueueMetrics DB password: the database password, in our example "qm"
- QueueMetrics URL: the login URL for QM, e.g. "http://1.2.3.4:8080/queuemetrics"
- QueueMetrics LogID: leave it to VIC (this in an ID for the server)
- QueueMetrics EnterQueue Prepend: This field is used to allow for prepending of one of the vicidial_list data fields in front of the phone number of the customer for customized QueueMetrics reports. Default is NONE to avoid populating anything.

A set of cron jobs is expected to run to keep the logs updated; check that they are present by issuing a *crontab -e*:

```
### fix the vicidial_agent_log once every hour and the full day run at night
33 * * * * /usr/share/astguiclient/AST_cleanup_agent_log.pl
50 0 * * * /usr/share/astguiclient/AST_cleanup_agent_log.pl --last-24hours
*/5 * * * * /usr/share/astguiclient/AST_cleanup_agent_log.pl --only-qm-live-call-check
1 1 * * * /usr/share/astguiclient/Vtiger_optimize_all_tables.pl --quiet
```

Also, you will need to install the PHP XML-RPC library in order to have audio data accessible from the QueueMetrics server:

```
pear install XML_RPC-1.5.1
```

Changes to QueueMetrics

Edit the *configuration.properties* file in order to set the following properties:

```
# This is the default queue log file.
default.queue_log_file=sql:P01
```

By default, ViciDial logs all data to partition "P01".

```
audio.server=it.loway.app.queuemetrics.callListen.listeners.ClassicXmlRpcRecordings

audio.liveserver=it.loway.app.queuemetrics.callListen.RTlisteners.ClassicXmlRpcListene
rRT

default.audioRpcServer=http://1.2.3.6/vicidial/xml_rpc_audio_server_vicidial.php
```

Change *1.2.3.6* to your ViciDial server address.

After this, you need to define each ViciDial campaign as a QueueMetrics queue, and set it properly as an inbound or outbound one. After that, you can freely create composite queues to report on all or some activity at once.

The live monitoring asks for an extension to send the call to, this is an extension dialed on the active voicemail server as defined in the system settings. If there is no active voicemail defined then the live monitor will place the call to the extension on the server that the agent is on.

Serving QueueMetrics through Apache

> We advise using a NGINX proxy instead - it's easier to configure and maintain. See Serving QueueMetrics through a NGINX proxy.

You may want to serve QueueMetrics through an Apache front-end instead of using Tomcat natively. This is useful if:

- You require better efficiency, so that static files are served natively without passing through Tomcat
- You need to integrate Qm on a virtual server that offers other services, e.g. applications written in PHP/Perl/CGI
- You need to serve QM on the public internet and want to use the security tools Apache offers.

Prerequisites

- A working QueueMetrics instance
- Apache 2.0 installed, with headers and compilation tools

Installing mod_jk

Download *mod_jk* from the Apache Tomcat website; it will be in a file named e.g *jakarta-tomcat-connectors-jk-1.2-src-current.tar.gz* .

Run the following commands:

```
tar zxvf jakarta-tomcat-connectors-jk-1.2-src-current.tar.gz
cd jk/native
```

Check where the *apxs* command is by running *locate apxs*. Default location is */usr/sbin/apxs*.

Check that $CATALINA_HOME and $JAVA_HOME are defined; default values are */usr/local/queuemetrics/tomcat* and */usr/local/queuemetrics/java* respectively.

Configure *mod_jk* by running

```
./configure \
    --with-apxs=/usr/sbin/apxs \
    --with-tomcat41=$CATALINA_HOME \
    --with-java-home=$JAVA_HOME \
    --with-jni

make
make install
```

This will build mod_jk and install it as an Apache module

Configuration of Apache and mod_jk

Add the following lines to */etc/http/conf/httpd.conf*. Check for paths to be correct.

```
#-------------------------------------------------------
#                      t o m c a t
#-------------------------------------------------------

# Load mod_jk module
LoadModule jk_module modules/mod_jk.so

# Where to find workers.properties
JkWorkersFile /etc/httpd/conf/workers.properties

# Where to put jk logs
JkLogFile /var/log/httpd/mod_jk.log
JkLogLevel info
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "

# JkOptions indicate to send SSL KEY SIZE,
JkOptions +ForwardKeySize +ForwardURICompat +ForwardDirectories

# JkRequestLogFormat set the request format
JkRequestLogFormat "%w %V %T"

# Send everything for context /examples to worker named worker1 (ajp13)
# JkMount /examples/* worker1
# JkMount /* worker1
```

We comment out JkMount lines because we will define them at the virtual host level.

Configure workers by creating the file */etc/httpd/conf/workers.properties*:

```
# Define 1 real worker using ajp13
 worker.list=worker1

# Set properties for worker1 (ajp13)
 worker.worker1.type=ajp13
 worker.worker1.host=localhost
 worker.worker1.port=8009
 worker.worker1.lbfactor=50
 worker.worker1.cachesize=10
 worker.worker1.cache_timeout=600
 worker.worker1.socket_keepalive=1
 worker.worker1.socket_timeout=300
```

Each worker is a Tomcat instance; you can define more than one if you run multiple webapps each in their own Virtual Machine, for maximum security.

Virtual host configuration

You basically have two possible approaches to mounting webapps in Apache so that their content is handled by Tomcat:

- Redirecting: you tell Tomcat that some URLs (e.g. the ones ending in **.jsp**) are to be handled by Tomcat, while all other files are server statically by Apache itself. This requires the webapp to be present on the same host as Tomcat
- Proxying: you tell Apache to forward all requests within a specified virtual host to Tomcat for serving. The server may be on a different host, therefore making it possible to serve contents from an external server.

You can use either model but - of course - you cannot use both for the same QueueMetrics instance.

Model 1: Redirecting some URLs to JK

Check the following lines in *httpd.conf*:

```
Listen 80
NameVirtualHost *
```

Add the following lines for each Virtual Host you want to support:

```
<VirtualHost *>
    ServerName queuemetrics.example.com
    ServerAlias queuemetrics_test.example.com
    ServerAdmin webmaster@example.com

    DocumentRoot /var/www/virtualhost/example.com/queuemetrics
    CustomLog /var/log/httpd/queuemetrics.example.com_access.log common
    ErrorLog /var/log/httpd/queuemetrics.example.com_error.log
    AddDefaultCharset UTF-8

    JkMount /*.jsp worker1
    JkMount /*.do worker1
    JkMount /tpf worker1
    JkMount /manager/* worker1
</VirtualHost>
```

You can include or exclude the */manager* path in order to access Tomcat's manager.

Model 2: Using Apache as a proxy to Tomcat

Using this model, you forward requests to a Tomcat server that may or may not be on the same

host. All requests for this domain are forwarded.

```
 <VirtualHost *:80>
    ServerName queuemetrics.example.com
    ServerAlias queuemetrics_test.example.com

    CustomLog /var/log/httpd/queuemetrics.example.com_access.log common
    ErrorLog /var/log/httpd/queuemetrics.example.com_error.log

    AddDefaultCharset UTF-8

    ProxyPreserveHost On
    ProxyPass / ajp://localhost:8009/
    ProxyPassReverse / ajp://localhost:8009/
</VirtualHost>
```

As the AJP request contains the full virtual host information, you still have to set up a virtual host in Tomcat for the hosts you need to proxy.

Of course, you could have Tomcat reside on a separate server on the same subnet, so you would offload all computation-intensive activities from your internet-facing server to a separate server that runs QueueMetrics.

Creating Virtualhosts in Tomcat

Turning off unnecessary connectors

Within Tomcat's *server.xml* file, within the section marked by *SERVICE NAME="Catalina"*, remove all connector entries but the one here:

```
<!-- Define a Coyote/JK2 AJP 1.3 Connector on port 8009 -->
    <Connector port="8009"
               enableLookups="false" redirectPort="8443" debug="0"
               protocol="AJP/1.3" />
```

This is the access point for Apache. This port should be unreachable outside this box.

Enabling the virtual host

By the end of *server.xml*, after the default virtual host (section *<Host>...</Host>*) add an entry like:

```
<Host name="queuemetrics.example.com"  debug="0"
    appBase="/var/www/virtualhost/example.com/queuemetrics"
    unpackWARs="true">
    <Alias>qm2.example.com</Alias>

    <Logger className="org.apache.catalina.logger.FileLogger"
```

```
        directory="/var/log/httpd"
        prefix="queuemetrics.example.com_tomcat-" suffix=".log"
           timestamp="false"/>

    <Context path="" docBase="" debug="0" reloadable="true"/>

   <Context path="/manager" debug="0" privileged="true"
       docBase="/usr/local/queuemetrics/tomcat/webapps/manager">
   </Context>
</Host>
```

If you want the manager webapp to be available, you need to include the context path as in the example above (check the path to be correct).

Restart everything.

```
/etc/init.d/httpd restart
/etc/init.d/queuemetrics restart
```

Check the logs when restarting. Go to http://queuemetrics.example.com/queuemetrics and check that QueueMetrics is working.

Troubleshooting

If you see lines like these appear on *catalina.out*:

```
org.apache.jk.common.HandlerRequest decodeRequest
WARNING: Error registering request
```

You need to locate the file *jk2.properties* and add/edit the following line:

```
request.registerRequests=false
```

CAUTION: This change must be made when Tomcat is stopped, or it will overwrite it when it terminates.

Serving QueueMetrics over Apache/SSL

> We advise using a NGINX proxy instead - it's easier to configure and maintain. See Serving QueueMetrics through a NGINX proxy.

Thanks to Matthew J. Roth.

Prerequisites

- A working QueueMetrics instance, served by an Apache 2 front-end
- Wildcard SSL certificates created by a recognized authority (e.g. GoDaddy.com)

The examples below are based on Apache 2 running on CentOS 5.3. Details may vary.

It is important to note that configuring Tomcat to take advantage of secure sockets is usually only necessary when running it as a stand-alone web server. When running Tomcat primarily as a Servlet/JSP container behind another web server, such as Apache or Microsoft IIS, it is usually necessary to configure the primary web server to handle the SSL connections from users. Typically, this server will negotiate all SSL-related functionality, then pass on any requests destined for the Tomcat container only after decrypting those requests. Likewise, Tomcat will return cleartext responses, that will be encrypted before being returned to the user's browser. In this environment, Tomcat knows that communications between the primary web server and the client are taking place over a secure connection (because your application needs to be able to ask about this), but it does not participate in the encryption or decryption itself.

Configure the Name-Based SSL Virtual Hosts

"Note" Apache will allow you to configure name-based SSL virtual hosts, but it will always use the configuration from the first-listed virtual host (on the selected IP address and port) to setup the encryption layer. In certain specific circumstances, it is acceptable to use a single SSL configuration for several virtual hosts. In particular, this will work if the SSL certificate applies to all of the virtual hosts. For example, this will work if:

1. All the virtual hosts are within the same domain, e.g. *one.example.com* and *two.example.com*.
2. You have a wildcard SSL certificate for that domain (one where the Common Name begins with an asterisk, e.g. **example.com*).

Remember that the SSL directives from all virtual hosts except the first-listed one will be ignored when setting up the initial SSL connection.

Install the Apache HTTP Server and its SSL/TLS Module

```
# yum install httpd
  * Installed: httpd.x86_64 0:2.2.3-31.el5.centos
```

```
  # yum install mod_ssl
    * Installed: mod_ssl.x86_64 1:2.2.3-31.el5.centos
    * Dependency Installed: distcache.x86_64 0:1.4.5-14.1
```

Configure the global Apache Settings

```
  # mkdir /etc/httpd/vhosts.d
  # cp /etc/httpd/conf/httpd.conf /etc/httpd/conf/httpd.conf.orig
  # vi /etc/httpd/conf/httpd.conf
    * Add the following lines to the end of 'Section 3: Virtual Hosts':
      #
      # Use name-based virtual hosting.
      #
      NameVirtualHost *:80

      #
      # Use name-based SSL virtual hosting.
      #
      NameVirtualHost *:443

      #
      # Load virtual hosts from the vhosts directory "/etc/httpd/vhosts.d".
      #
      Include vhosts.d/*.conf
```

Configure the global SSL/TLS Settings

```
  # cp /etc/httpd/conf.d/ssl.conf /etc/httpd/conf.d/ssl.conf.orig
  # vi /etc/httpd/conf.d/ssl.conf
    * Use the '<IfDefine>' directive to disable the default SSL virtual host as
follows:
      #
      # Disable this default SSL virtual host
      #
      <IfDefine 0>
      ##
      ## SSL Virtual Host Context
      ##

      <VirtualHost _default_:443>
      ...
      </VirtualHost>
      </IfDefine>
```

Change the following lines from:

```
SSLRandomSeed startup file:/dev/urandom  256
SSLSessionCacheTimeout  300
```

to:

```
SSLRandomSeed startup file:/dev/urandom 1024
SSLSessionCacheTimeout  600
```

Create DNS records

- https-test1.example.com
- https-test2.example.com

Create the Application Directories

```
# mkdir /var/www/https-test1.example.com
# mkdir /var/www/https-test1.example.com/{conf,html,logs,webapps}
# mkdir /var/www/https-test1.example.com/conf/ssl.{crl,crt,csr,key}
# mkdir /var/www/https-test2.example.com
# mkdir /var/www/https-test2.example.com/{conf,html,logs,webapps}
# mkdir /var/www/https-test2.example.com/conf/ssl.{crl,crt,csr,key}
```

Install the CRT, CSR, and KEY files

```
# install -m 400 -o root -g apache /tmp/wildcard.example.com.key \
                                   /var/www/https-test1.example.com/conf/ssl.key/
# install -m 400 -o root -g apache /tmp/wildcard.example.com.key.unsecure \
                                   /var/www/https-test1.example.com/conf/ssl.key/
# install -m 440 -o root -g apache /tmp/wildcard.example.com.crt \
                                   /var/www/https-test1.example.com/conf/ssl.crt/
# install -m 440 -o root -g apache /tmp/gd_bundle.crt \
                                   /var/www/https-test1.example.com/conf/ssl.crt/
# install -m 440 -o root -g apache /tmp/wildcard.example.com.csr \
                                   /var/www/https-test1.example.com/conf/ssl.csr/
# install -m 400 -o root -g apache /tmp/wildcard.example.com.key \
                                   /var/www/https-test2.example.com/conf/ssl.key/
# install -m 400 -o root -g apache /tmp/wildcard.example.com.key.unsecure \
                                   /var/www/https-test2.example.com/conf/ssl.key/
# install -m 440 -o root -g apache /tmp/wildcard.example.com.crt \
                                   /var/www/https-test2.example.com/conf/ssl.crt/
# install -m 440 -o root -g apache /tmp/gd_bundle.crt \
                                   /var/www/https-test2.example.com/conf/ssl.crt/
# install -m 440 -o root -g apache /tmp/wildcard.example.com.csr \
                                   /var/www/https-test2.example.com/conf/ssl.csr/
```

```
# rm -f /tmp/wildcard.example.com.* gd_bundle.crt
```

Configure the Virtual Hosts

```
# vi /etc/httpd/vhosts.d/000-https-test1.example.com.conf

  --- START 000-https-test1.example.com.conf CONTENTS -----------------------------
  # Define the https-test1.example.com name-based SSL virtual host

  # These are the default virtual hosts for port 80 and port 443

  <VirtualHost *:80>
      # This virual host exists solely to redirect all non-SSL traffic to the SSL
      # virtual host.  This is done in an SEO friendly manner by using the
      # 'RedirectPermanent' directive.  If the redirect is somehow circumvented,
      # the 'DocumentRoot' directive is set to serve content from a non-secure
      # directory.

      ServerAdmin admin@example.com
      ServerName https-test1.example.com
      ServerAlias https-test1
      DocumentRoot /var/www/html
      ErrorLog /var/www/https-test1.example.com/logs/error_log
      CustomLog /var/www/https-test1.example.com/logs/access_log common

      RedirectPermanent / https://https-test1.example.com/
  </VirtualHost>

  <VirtualHost *:443>
      # This is the SSL virtual host.  It is configured so that strong
      # cryptography (128 bit encryption or greater) is required to access any web
      # content.

      ServerAdmin admin@example.com
      ServerName https-test1.example.com
      ServerAlias https-test1
      DocumentRoot /var/www/https-test1.example.com/html
      ErrorLog /var/www/https-test1.example.com/logs/ssl_error_log
      CustomLog /var/www/https-test1.example.com/logs/ssl_access_log common

      # Enable SSL for this virtual host
      SSLEngine on

      # Deny all requests which are not using SSL...
      <Directory "/var/www/https-test1.example.com/html">
          SSLRequireSSL
      </Directory>

      # ...even under a 'Satisfy Any' situation
```

```
        SSLOptions +StrictRequire

        # List the SSL protocol flavors with which clients can connect
        SSLProtocol -all +TLSv1 +SSLv3

        # List the cipher suites that clients are permitted to negotiate
        SSLCipherSuite HIGH:MEDIUM:!aNULL:+SHA1:+MD5:+HIGH:+MEDIUM

        # Point to the PEM-encoded certificate, private key, and CA certificate
        # chain files for this virtual host
        SSLCertificateFile /var/www/https-
test1.example.com/conf/ssl.crt/wildcard.example.com.crt
        SSLCertificateKeyFile /var/www/https-
test1.example.com/conf/ssl.key/wildcard.example.com.key.unsecure
        SSLCertificateChainFile /var/www/https-
test1.example.com/conf/ssl.crt/gd_bundle.crt

        # Handle problems with broken clients, such as older versions of Internet
        # Explorer
        SetEnvIf User-Agent ".*MSIE.*" \
                 nokeepalive ssl-unclean-shutdown \
                 downgrade-1.0 force-response-1.0

        # Log information about the SSL parameters that are negotiated for requests
        CustomLog /var/www/https-test1.example.com/logs/ssl_request_log \
                  "%t %h %{HTTPS}x %{SSL_PROTOCOL}x %{SSL_CIPHER}x
%{SSL_CIPHER_USEKEYSIZE}x %{SSL_CLIENT_VERIFY}x \"%r\" %b"
    </VirtualHost>
    --- END 000-https-test1.example.com.conf CONTENTS -------------------------------
```

Do the same for the second virtual host, setting the names as appropriate.

```
  # vi /etc/httpd/vhosts.d/001-https-test2.example.com.conf
```

Verify the Configuration

```
  # httpd -S
  VirtualHost configuration:
  wildcard NameVirtualHosts and _default_ servers:
  *:443                  is a NameVirtualHost
           default server https-test1.example.com (/etc/httpd/vhosts.d/000-https-
test1.example.com.conf:22)
           port 443 namevhost https-test1.example.com (/etc/httpd/vhosts.d/000-https-
test1.example.com.conf:22)
           port 443 namevhost https-test2.example.com (/etc/httpd/vhosts.d/001-https-
test2.example.com.conf:20)
  *:80                   is a NameVirtualHost
           default server https-test1.example.com (/etc/httpd/vhosts.d/000-https-
```

```
test1.example.com.conf:5)
           port 80 namevhost https-test1.example.com (/etc/httpd/vhosts.d/000-https-
test1.example.com.conf:5)
           port 80 namevhost https-test2.example.com (/etc/httpd/vhosts.d/001-https-
test2.example.com.conf:3)
  Syntax OK
```

Setup the service for automated startup

```
  # chkconfig httpd on ; chkconfig --list httpd
  httpd           0:off   1:off   2:on    3:on    4:on    5:on    6:off
  # service httpd start
  Starting httpd:                                            [  OK  ]
```

Troubleshooting

```
  # openssl s_client -connect localhost:443
  ... SSL Connection Establishment ...
  ---
  GET / HTTP/1.0

  HTTP/1.1 200 OK
  ... HTTP Headers ...

  ... Web Page Contents ...
  closed
```

Summary of Log Files used by Apache

- /var/log/httpd/access_log
- /var/log/httpd/error_log
- /var/www/https-test1.example.com/logs/access_log
- /var/www/https-test1.example.com/logs/error_log
- /var/www/https-test1.example.com/logs/ssl_access_log
- /var/www/https-test1.example.com/logs/ssl_error_log
- /var/www/https-test1.example.com/logs/ssl_request_log
- /var/www/https-test2.example.com/logs/access_log
- /var/www/https-test2.example.com/logs/error_log
- /var/www/https-test2.example.com/logs/ssl_access_log
- /var/www/https-test2.example.com/logs/ssl_error_log

- /var/www/https-test2.example.com/logs/ssl_request_log

Notes

Warnings related to name-based SSL virtual hosts, such as the following, can be ignored:

```
[warn] Init: SSL server IP/port conflict: https-test1.example.com:443
(/etc/httpd/vhosts.d/000-https-test1.example.com.conf:22) vs. https-
test2.example.com:443 (/etc/httpd/vhosts.d/001-https-test2.example.com.conf:20)
[warn] Init: You should not use name-based virtual hosts in conjunction with SSL!!
[warn] RSA server certificate CommonName (CN) '*.example.com' does NOT match server
name!?
```

Sources

- Setting Up a Secure Apache 2 Server - http://www.informit.com/articles/article.aspx?p=30115
- Apache 2 with SSL/TLS: Step-by-Step - http://www.securityfocus.com/infocus/1818
- How to Create Self-Signed SSL Certificates with OpenSSL - http://www.xenocafe.com/tutorials/linux/centos/openssl/self_signed_certificates/index.php
- NameBasedSSLVHosts - Httpd Wiki - http://wiki.apache.org/httpd/NameBasedSSLVHosts
- What is a Wildcard SSL certificate? - http://help.godaddy.com/article/567
- Best SSL Wildcard Certificates - http://www.sslshopper.com/best-ssl-wildcard-certificate.html
- Apache SSL in htaccess examples - http://www.askapache.com/htaccess/apache-ssl-in-htaccess-examples.html
- Apache SSL/TLS Encryption - http://httpd.apache.org/docs/2.2/ssl/
- Apache Module mod_alias - http://httpd.apache.org/docs/2.2/mod/mod_alias.html

Installing QM as a ROOT webapp

QueueMetrics is usually deployed as a webapp whose path stems from the root of the webserver - for example, as http://queuemetrics.example.com:8080/queuemetrics

It is possible to deploy Queuemetrics as a ROOT webapp, so that the complete address ends up being simply http://queuemetrics.example.com

Prerequisites

- A working QueueMetrics instance

Model 1: using a ROOT webapp

Changes to Tomcat

Copy the current version of QueueMetrics to a webapp named *ROOT* (all capital letters).

```
cd /usr/local/queuemetrics/tomcat/webapps
cp -R /usr/local/queuemetrics/webapps/queuemetrics-1.6.0/ ROOT
```

Restart QueueMetrics

Set the access port

If you want QueueMetrics to be available on a port that is different from the default one (8080), edit the *server.xml* file and look for a line that looks like:

```
<!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8080 -->
<Connector port="8080"
   maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
   enableLookups="false" redirectPort="8443" acceptCount="100"
   debug="0" connectionTimeout="20000"
```

and change that to:

```
<Connector port="80"
   maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
   enableLookups="false" redirectPort="8443" acceptCount="100"
   debug="0" connectionTimeout="20000"
```

CAUTION: Make sure you have no Apache running on port 80; in case, turn it off.

Restart QueueMetrics.

Model 2: defining a root context

It is also possible to define a specific webapp as the root webapp for a virtual host by setting it as a root context. This can be achieved quite easily by editing the *'server.xml'* file in Tomcat:

```
<Host name="example.com"  debug="0"
        appBase="/www/example.com/www"
        unpackWARs="true">
        <Alias>www.example.com</Alias>

        <Logger className="org.apache.catalina.logger.FileLogger"
                directory="/var/log/httpd"
                prefix="queuemetrics-" suffix=".log"
           timestamp="false"/>

        <Context path="" docBase="queuemetrics" debug="0" reloadable="true"/>

</Host>
```

This example assumes that your QueueMetrics is installed in *'/www/example.com/www/queuemetrics'* and that you want to serve QM as a root webapp for domains *'example.com'* and *'www.example.com'*.

Changes to QueueMetrics

None required.

Enabling GZIP compression in Tomcat

> We advise using a NGINX proxy instead - it's easier to configure and maintain. See Serving QueueMetrics through a NGINX proxy

You can speed up the serving of QueueMetrics pages over a WAN by transparently compressing the page before being sent; it will be transparently decompressed by your browser. As QueueMetrics pages (especially large tables) are highly redundant, this technique can buy large improvements in the user experience at a cost of some CPU time on the server.

Prerequisites

- A working QueueMetrics instance

Changes to Tomcat

Edit the *server.xml* file under *tomcat/config*; locate the HTTP connector instance (the one that shows port 8080) and change it as follows:

```
<Connector port="8080" maxHttpHeaderSize="8192"
    maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
    enableLookups="false" redirectPort="8443" acceptCount="100"
    connectionTimeout="20000" disableUploadTimeout="true"

    compression="on"
    compressionMinSize="2048"
    noCompressionUserAgents="gozilla, traviata"
    compressableMimeType="text/html,text/xml"
/>
```

Restart QueueMetrics.

Changes to QueueMetrics

None required.

For further reading

- http://viralpatel.net/blogs/2008/11/enable-gzip-compression-in-tomcat.html

Advanced QueueMetrics monitoring

The Java VM offers very powerful APIs to monitor and diagnose live systems while they are running; they are meant to be run in production with negligible performance impact.

This can be useful to diagnose specific problems, e.g. Java heap exhaustion issues, or to monitor the activity of your QM servers.

Prerequisites

- A QueueMetrics instance running under JDK 6 or newer. The specific version of Java that is being run can easily be seen under the License page of QueueMetrics.

Recent versions of QueueMetrics installed using *yum* should already be running under JDK 8 or 11. If this is not your case, you should upgrade the *queuemetrics-java* package.

Assessing memory problems

If you feel you are experiencing memory issues, you should take multiple memory and thread dumps spaced a couple of hours in between and send them to Loway for inspection.

We will usually need:

- The current memory settings
- A memory dump
- A thread dump

They should be obtained as described below.

Finding the current QueueMetrics PID

In order to perform the procedures described below, you must know the PID of your currently running QueueMetrics instance. It can usually be found out by running:

```
[root@qm ~]# ps fax | grep catalina
32313 pts/0    S+     0:00          \_ grep catalina
12345 ?        Sl     0:14 /usr/java/jdk1.6.0_17/bin/java -Xms128M .....
```

Here in the example QM is running with a PID of 12345.

The PID is used to attach to the current JVM and query it. It is also possible to start the JVM so that it allows administrative access over a network; therefore all the procedures described below can be run on a remote JVM as well.

Taking a memory dump

A memory dump presents a (long) list of all the loaded Java classes, and how many instances of each are present in memory.

```
[root@qm ~]# /usr/java/jdk1.6.0_17/bin/jmap -histo:live 12345
```

You should also collect general memory area usage statistics by running:

```
[root@qm ~]# /usr/java/jdk1.6.0_17/bin/jmap 12345
```

Taking a thread dump

A thread dump prints out - thread by thread - what each one is doing at a given moment. This is useful to diagnose load-based issues where too many requests and open sessions "flood" the QM server.

```
[root@qm ~]# /usr/java/jdk1.6.0_17/bin/jstack -l 12345
```

This lets you know what a "frozen" server with high CPU usage is actually doing.

Remote monitoring with VisualVM

VisualVM is a graphical tool that comes with the Java SDK that lets you monitor a remote QueueMetrics instance while it's running (it can actually be used with any Java-based process).

It allows monitoring over a network, so it is common to run it on a workstation to monitor one or more remote servers.

You can find it at: https://visualvm.github.io/ and it is already included in all modern Java downloads (so it is likely you already have it on your PC) - if not, it can be easily downloaded.

Remote monitoring over SSH with RPM

To make your life easier, we made it possible to set up a fully working JMX connection just by uncommenting a single line, and creating an SSH tunnel that will let us access the data in a secure way. You will need a recent Tomcat RPM (8.5.64 or newer).

So, once it is installed, you edit `/etc/sysconfig/qm-tomcat6`, and where it says:

```
# Remove the comments below to activate JMX monitoring
# JAVA_RMI_ADDRESS=localhost
JAVA_JMX_PORT="30000"
```

You uncomment the line with `JAVA_RMI_ADDRESS`; save the file and restart. You just need to do this

once, and this can be done safely even on production systems. You can change the JMX port if you need to - does not matter which port you choose, as long as it is the same on your PC and on the server.

Now, you go to your workstation and start a new SSH session with the following incantation, to create a local tunnel for port 30000 and the following one too:

```
ssh root@my.queuemetrics.server \
    -L 30000:localhost:30000 \
    -L 30001:localhost:30001
```

You can now run VisualVM on your workstation - most likely it will already be installed if you have a Java SDK; if not, you can download it from: https://visualvm.github.io

Note that while the example above logs in over SSH as *root*, any user will do.

When you launch VisualVM, you should now click on the button *Add JMX connection*, and create a new connection to *localhost:30000*, with no password and no encryption (as we use SSH for security and encryption).

Remote monitoring over SSH

If you cannot use the simplified method above, as getting the configuration right for remote access can be tricky (see below for an example), it is usually better to use a SSH tunnel to connect to a running JVM instance.

The Java RMI protocol requires forwarding of two ports, and those ports must be the same on your box as they are on the remote server. As you can assign such ports freely, if you need to monitor multiple servers at once, make sure that the ports used are unique.

In order to allow it, you should add the following line to */etc/init.d/qm-tomcat6*:

```
export JAVA_OPTS="-Dcom.sun.management.jmxremote \
      -Dcom.sun.management.jmxremote.port=12345 \
      -Dcom.sun.management.jmxremote.rmi.port=12346 \
      -Dcom.sun.management.jmxremote.authenticate=false \
      -Dcom.sun.management.jmxremote.ssl=false \
      -Dcom.sun.management.jmxremote.local.only=false \
      -Djava.rmi.server.hostname=localhost \
      $JAVA_OPTS"
```

Restart the JVM after adding it. At this point the JVM allows connections, but only locally, on ports 12345 and 12346.

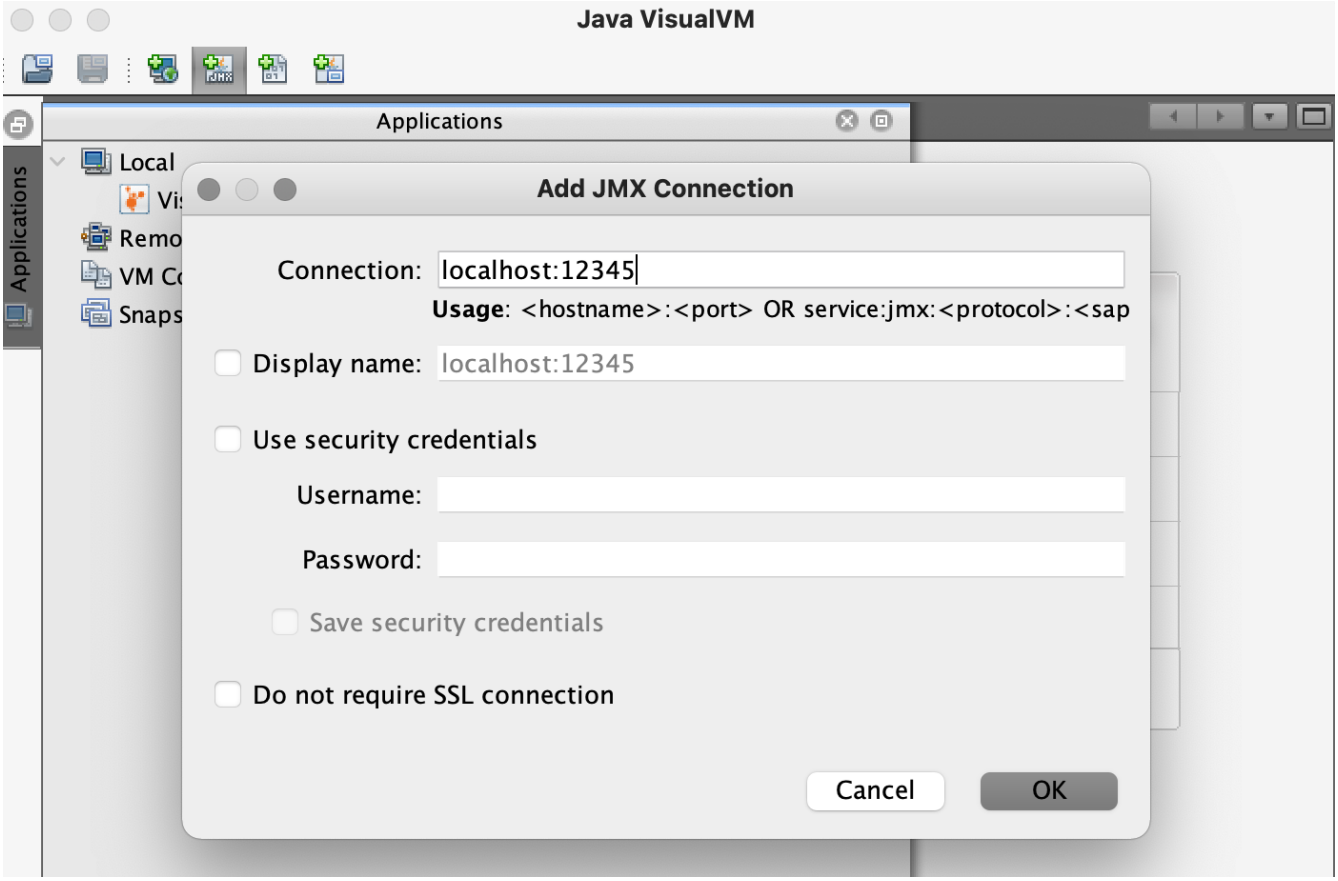
Now set up an encrypted tunnel to forward both ports to the same ports on your workstation:

```
ssh root@my.queuemetrics.server \
    -L 12345:localhost:12345 \
```

```
    -L 12346:localhost:12346
```

Note that while the example above logs in as *root*, any user will do.

When you launch VisualVM, you should now click on the button *Add JMX connection*, and create a new connection to *localhost:12345*, with no password and no encryption (as we use SSH for security and encryption).



When done, click on the newly created connection to start monitoring.

Allowing remote access

The standard JVM settings for QM *do not* allow remote access over a network, for obvious security reasons. In order to allow it, you should add the following line to */etc/init.d/qm-tomcat6*:

```
export JAVA_OPTS="-Dcom.sun.management.jmxremote \
      -Dcom.sun.management.jmxremote.port=12345 \
      -Dcom.sun.management.jmxremote.authenticate=false \
      -Dcom.sun.management.jmxremote.ssl=false \
      -Djava.rmi.server.hostname=10.10.5.106 \
      $JAVA_OPTS"
```

Restart the JVM after adding it. You can change the port (in this case we set it to 12345) for security purposes. Please note you must also set an host name or IP address for the machine - failure to do so will prevent all remote monitoring.

If you run this on a publicly-available server and/or you have a firewall, you should set up an encrypted connection and use SSL and password authentication - see http://docs.oracle.com/javase/1.5.0/docs/guide/management/agent.html

To make sure that the system restarted correctly and the JMX connection is actually used, you can run the following command:

```
[root@localhost ~]# lsof -i -P | grep 12345
java      14513     root   15u  IPv6 1133153       TCP *:12345 (LISTEN)
```

As you can see, Java process 14513 is listening on port 12345.

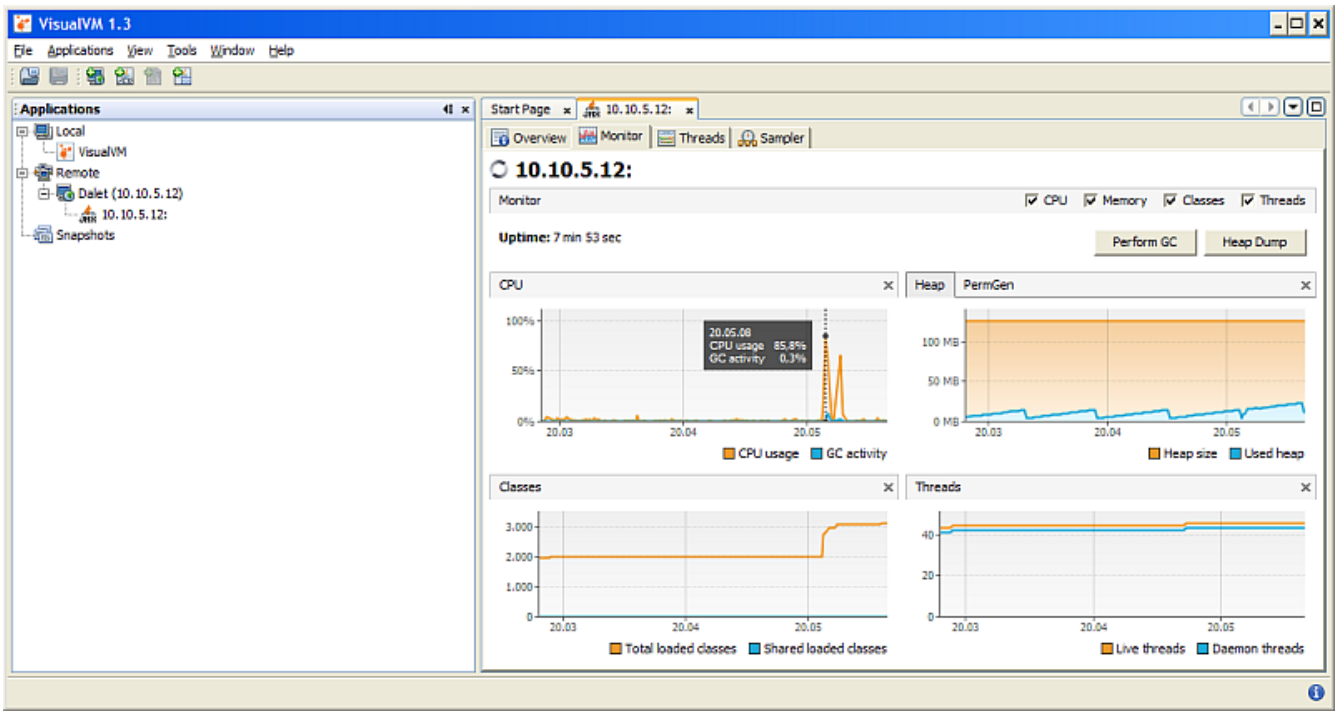
Starting VisualVM

To start VisualVM, you run *bin/visualvm.exe* in Windows, or just *jvisualvm* in Unix/Mac.

When started, click on "Remote" and enter the IP address of your QM server. Click on "Advanced settings" and set the port to the one you specified in the QM configuration (12345 in this example).

After that, you select your server and select "Add JMX connection" from the right-button menu. You enter the JMX connection as "IP:12345".

By clicking on it, you get a working connection, like in the picture below:



Things you can do in VisualVM

A number of interesting things can be done with VisualVM:

- *Know your JVM*: you can see the JVM settings from *Overview* / *JVM arguments*.
- *Memory monitoring*: you can see the current CPU, memory and thread usage from the *Monitor*

page. Note that with most settings, it is normal that all memory be used up before a garbage collection is performed; so you would expect to see spikes and falls in the graph. You can also force a garbage collection if you want to see the "true" memory usage, but this may be unwise on heavily loaded production servers.

- *Thread monitoring*: you can get a textual thread dump like the one discussed above by selecting *Threads* / *Thread dump*
- You can use the *Sampler* to acquire a breakdown of memory and CPU usage per class (you first need to install the plugin *VisualVM-Sampler* from the Plugins menu)
- You can keep a server open with multiple instances of VisualVM in order to monitor multiple QM servers

Tracing CPU hot-spots

In order to trace CPU hot-spots, getting to the right point requires a bit of black magic:

- attach to process via VisualVM
- go to "Sampler"
- click on "Sample: CPU"
- Wait a few minutes
- click on "Sample: Stop"
- Under "CPU Samples", click on "Collected data: Snapshot"
- Then in the section "View", click "Hot spots"
- At this point, click on the diskette and choose "Export hot spots"

You can also click on "Export profile", save an NPS file, and send it over. If you do, please send both files (NPS and Export Hot Spots), as sometimes NPS files are not readable with different version of Java.

The profile can then be loaded by opening VisualVM, choose "load" and select the NPS file.

Database connection pooling

Each QueueMetrics transaction requires a connection to the database in order to access call and configuration data. By using a connection pool in Tomcat, a given set of database connections is opened at startup and then recycled as needed, thus saving the cost of opening and closing a connection for every transaction. While this cost is negligible for general usage with a local database (generally in the order of 10 to 30 milliseconds), it can be a performance boost if the MySQL database is remote or your server is very busy.

The advantages of this technique can be summed up as:

- faster database access, mostly when the database is over a network or when there are many configuration parameters needed to fire up a connection
- easier monitoring of JDBC resource usage by third party tools
- maximum advantage when running AGAW clients

Prerequisites

- A working QueueMetrics instance, version 1.6.0 or newer

Before you start, find your JDBC URI in your *web.xml* file and copy it for future reference.

Changes to the MySQL server

Make sure that the total number of allowed connections is more than the maximum you configured for your pool. The number we use here is 50, as set by the *maxActive* parameter below, and should be OK for most MySQL servers. Keep in mind that if you need to access your MySQL server with other applications or a monitoring script, you will need more connections.

Changes to QueueMetrics

Modify the file *web.xml* in QueueMetrics as follows.

Change the parameter *JDBC_URL* (Where the JDBC URI was) to:

```
    <init-param>
        <param-name>JDBC_URL</param-name>
        <param-value>pool:jdbc/qm</param-value>
    </init-param>
```

By the end of the file, just before the web-app XML element closes, add:

```
  <resource-ref>
      <description>DB Connection</description>
      <res-ref-name>jdbc/qm</res-ref-name>
```

```
      <res-type>javax.sql.DataSource</res-type>
      <res-auth>Container</res-auth>
  </resource-ref>
```

This basically tells QueueMetrics that instead of connecting straight to the database, it must fetch a connection from a pool called *jdbc/qm* that is managed by Tomcat at the container level.

Changes to Tomcat

First, remove the username and password from the JDBC URI - they are passed separately here.

Modify the file *server.xml* that is usually held in */usr/local/queuemetrics/tomcat/config*, adding the following (long) section before the closing Host element:

```
<Context docBase="MYQMAPP"
         path="/MYQMAPP" reloadable="true">

  <Resource name="jdbc/qm"
              auth="Container"
              type="javax.sql.DataSource"/>

  <ResourceParams name="jdbc/qm">
    <parameter>
      <name>factory</name>
      <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
    </parameter>

    <!-- Maximum number of dB connections in pool. Make sure you
         configure your mysqld max_connections large enough to handle
         all of your db connections. Set to 0 for no limit.
         -->
    <parameter>
      <name>maxActive</name>
      <value>50</value>
    </parameter>
    <!-- You don't want too many idle connections hanging around
         if you can avoid it, only enough to soak up a spike in
         the load -->

    <parameter>
      <name>maxIdle</name>
      <value>5</value>
    </parameter>

    <!-- Don't use autoReconnect=true, it's going away eventually
         and it's a crutch for older connection pools that couldn't
         test connections. You need to decide whether your application
         is supposed to deal with SQLExceptions (hint, it should), and
         how much of a performance penalty you're willing to pay
```

```
        to ensure 'freshness' of the connection -->

    <parameter>
      <name>validationQuery</name>
      <value>SELECT 1</value>
    </parameter>

   <!-- The most conservative approach is to test connections
        before they're given to your application. For most applications
        this is okay, the query used above is very small and takes
        no real server resources to process, other than the time used
        to traverse the network.

        If you have a high-load application you'll need to rely on
        something else. -->

    <parameter>
      <name>testOnBorrow</name>
      <value>true</value>
    </parameter>

   <!-- Otherwise, or in addition to testOnBorrow, you can test
        while connections are sitting idle -->

    <parameter>
      <name>testWhileIdle</name>
      <value>true</value>
    </parameter>

    <!-- You have to set this value, otherwise even though
         you've asked connections to be tested while idle,
         the idle evicter thread will never run -->

    <parameter>
      <name>timeBetweenEvictionRunsMillis</name>
      <value>10000</value>
    </parameter>

    <!-- Don't allow connections to hang out idle too long,
         never longer than what wait_timeout is set to on the
         server...A few minutes or even fraction of a minute
         is sometimes okay here, it depends on your application
         and how much spikey load it will see -->

    <parameter>
      <name>minEvictableIdleTimeMillis</name>
      <value>60000</value>
    </parameter>

    <!-- Maximum time to wait for a dB connection to become available
         in ms, in this example 10 seconds. An Exception is thrown if
```

```
          this timeout is exceeded.  Set to -1 to wait indefinitely.
          -->
    <parameter>
      <name>maxWait</name>
      <value>30000</value>
    </parameter>

    <!-- MySQL dB username and password for dB connections  -->
    <parameter>
     <name>username</name>
     <value>queuemetrics</value>
    </parameter>

    <parameter>
     <name>password</name>
     <value>javadude</value>
    </parameter>

    <!-- Class name for MySQL JDBC driver -->
    <parameter>
       <name>driverClassName</name>
       <value>com.mysql.jdbc.Driver</value>
    </parameter>

    <!-- The JDBC connection url for connecting to your MySQL dB.
          The autoReconnect=true argument to the url makes sure that the
          mm.mysql JDBC Driver will automatically reconnect if mysqld closed the
          connection.  mysqld by default closes idle connections after 8 hours.
          -->
    <parameter>
      <name>url</name>

<value>jdbc:mysql://localhost/queuemetrics?zeroDateTimeBehavior=convertToNull&amp;jdbc
CompliantTruncation=false</value>
    </parameter>
  </ResourceParams>

</Context>
```

In the file above, you need to change the following elements:

```
<Context docBase="MYQMAPP"
         path="/MYQMAPP" reloadable="true">
```

Change *MYQMAPP* to the name of the webapp as deployed on your system (usually "queuemetrics").

Then change the *url* parameter and the *username* and *password* elements, setting your JDBC URL.

You may also fine-tune the maximum number of allowed connections in the pool and the eviction policies (but this goes beyond the scope of this tutorial).

Now copy the connector driver, such as *mysql-xxxx.jar*, to the */common/lib* directory of your Tomcat installation and remove it from *WEB-INF/lib/* of your QueueMetrics instance.

Restart QueueMetrics.

Moving QueueMetrics to a different server

This chapter explains the required steps in order to move a working QueueMetrics to a different server.

Prerequisites

- Stop qloader on the Asterisk server
- Make a backup of the current QM database
- Make sure the clocks of all servers involved are aligned to a sub-second difference (use NTP)

Required steps

The following steps must be followed:

1. Install a barebone QM - same version- on the new server
2. Move the database from the old server to the new one (overwrite the existing database)
3. Move the files *web.xml*, *tpf.properties* and *configuration.properties* to the new server (make sure you use the correct path on each server - you can find this on the License page)
4. Edit the file */etc/sysconfig/qloaderd* so that you can upload data to the new server. You will likely be adding grants on the new server's MySQL so that a user from the Asterisk box can connect
5. Restart qloaderd and make sure it is uploading data to the new server. You can see if this works from the DBTEST → Live DB inspector. When there is a new line on the queue_log, you should also see it in the database, after 1-2 seconds
6. Edit the *configuration.properties* file to make sure that the AMI connection strings (those properties that contain strings like with "tcp:...") point to your Asterisk server. You will probably have to edit Asterisk (*/etc/asterisk/manager.conf*) to allow AMI connections from a new server. You can test if this works from DBTEST → AMI Tester
7. If you use audio recordings, make sure that you can access them from the new server (you may have to use a network mount on the Asterisk server, and edit the path in *configuration.properties*)

Moving the queue_log table to InnoDB

Thanks to Jens von Bulow

Most QueueMetrics tables use a storage engine called myISAM - this was the default storage engine, and works well on tables that are written infrequently and read often. If you run a clustered call-center with multiple partitions and many rows being inserted per second, you may see database response times degrade. In this case, moving the storage engine to InnoDB, a storage engine that is way better for contended tables, may make a difference.

Prerequisites

- A working QueueMetrics instance
- MySQL server version 5

Before you consider doing this, make sure that you have read the current MySQL documentation on "Converting tables to MySQL" - http://dev.mysql.com/doc/refman/5.6/en/converting-tables-to-innodb.html - your server settings may require some tweaking to get good performance out of the new table.

Changes to MySQL

In order to convert the table, we will:

- Make sure you have enough space on disk (at least 2x the size of your current queue_log table). Make sure you have a complete backup of the old DB (in case something goes wrong) and you have a suitable maintenance window so you can work comfortably.
- Create a new table called "queue_log_i" where we will load old data. We will create it with the following definition - make sure that the set of fields exactly matches your current fields:

```
CREATE TABLE `queue_log_i` (
  `partition` varchar(20) NOT NULL DEFAULT '',
  `time_id` int(11) unsigned NOT NULL DEFAULT '0',
  `call_id` varchar(200) NOT NULL,
  `queue` varchar(50) NOT NULL,
  `agent` varchar(30) NOT NULL DEFAULT '',
  `verb` varchar(30) NOT NULL DEFAULT '',
  `data1` varchar(200) NOT NULL DEFAULT '',
  `data2` varchar(200) NOT NULL DEFAULT '',
  `data3` varchar(200) NOT NULL DEFAULT '',
  `data4` varchar(200) NOT NULL DEFAULT '',
  `data5` varchar(200) NOT NULL DEFAULT '',
  `serverid` varchar(10) NOT NULL DEFAULT '',
  `unique_row_count` int(10) unsigned NOT NULL ,
  KEY `idx_sel` (`partition`,`time_id`,`queue`(2)),
  KEY `partizione_b` (`partition`,`time_id`,`unique_row_count`),
  KEY `by_hotdesk` (`verb`,`time_id`)
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

- Copy data from the old table to the new one. We will insert it in the correct order to make reading quicker.

```
INSERT INTO queue_log_i
 SELECT *
   FROM queue_log
  ORDER BY `partition` ASC, `time_id` ASC, `unique_row_count` ASC
```

- Stop qloaderd
- Rename the new table to "queue_log" and the previous one to "queue_log_old". This can be done atomically as in:

```
RENAME TABLE queue_log TO queue_log_old,
             queue_log_i TO queue_log
```

- Restart QueueMetrics
- Restart qloaderd
- Optionally - delete the old queue_log table.

Bonus: checking your InnoDB execution plan

If you are unsure whether your new table is working correctly, you can:

- Log slow queries from QueueMetrics
- Run the following commands:

```
SET optimizer_trace="enabled=on";

SELECT ..... (your slow query);

SELECT * FROM INFORMATION_SCHEMA.OPTIMIZER_TRACE;
```

At this point, you can find the detailed execution plan in a field called TRACE. This is especially useful to diagnose why an index may not be used.

Changes to QueueMetrics

None required.

Using Master-master database replication for strong high-availability

It is possible to obtain strong, resilient high availability over a wide area network with automated replication using master-master replication.

Imagine this scenario: you have 4 Asterisk servers that are physically in two locations connected over a WAN; you want the cluster monitoring to be available from any location, and you want to have a usable (local) system even in a case where the WAN should become unavailable.

```
        SITE A                    SITE B
     IPs 10.10.1.x             IPs 10.10.2.x

      +---------+               +---------+
      |  QM A   |               |  QM  B  |
      +---------+               +---------+
           |                         |
           |                         |
      +---------+  == A->B ==>  +---------+
      | MySQL A |               | MySQL B |
      +---------+  <== B->A ==  +---------+
        |     |                   |     |
        |     |                   |     |
+---------+ +---------+   +---------+ +---------+
| PBX A1  | | PBX A2  |   | PBX B1  | |  PBX B2 |
+---------+ +---------+   +---------+ +---------+
```

What we do is this: we connect the two MySQL servers in a master-master fashion, so that data from one is automatically replicated to the other within a minimum delay.

In case the WAN goes down, data from the local PBXs is still available in real-time; when the WAN comes back online, the two databases sync automatically and make all data available to all users again.

How it works:

- Each PBX has a local qloaderd that uploads data to the local MySQL database; each PBX uploads data to a partition that has its own name in it (e.g. PBX A1 uploads data to partition PA1, A2 to PA2, B1 to PB1, B2 to PB2)
- Each MySQL server is configured to insert rows with a unique id that is always odd for server A and always even for server B; this way the same table can be shared on insert with no issues
- Each MySQL server holds all the data; some coming from local Qloaderds and some from the replica of the other database
- Each QM server is able to monitor all four PBXs at once; using a cluster of all partitions - so it does not need to know what is where

Prerequisites

- Two clustered QueueMetrics licenses.
- All server clocks aligned to a sub-second difference via NTP
- MySQL server version 5 or later

When doing this tutorial, we assume that we have a working QueueMetrics database on MySQL "A" while we have nothing on server "B". During the replication, we will clone the database on server A to the new server B.

	Server A	**Server B**
QM IP Address	10.10.1.10	10.10.2.10
MySQL IP address	10.10.1.11	10.10.2.11
Asterisk 1	10.10.1.12	10.10.2.12
Asterisk 2	10.10.1.13	10.10.2.13
QM database	queuemetrics	queuemetrics

Changes to MySQL

Changes to the insert order

On server A, you add the following lines to your /etc/my.cnf configuration file, under the [mysqld] section:

```
auto_increment_increment= 2
auto_increment_offset   = 1
```

This way all inserted lines will be odd.

You do the same on server B.

```
auto_increment_increment= 2
auto_increment_offset   = 2
```

In this case all inserted lines will be even.

On both servers, make sure that the MySQL server is available on a public IP by editing */etc/mysql/my.cnf* - this is not so by default:

```
bind-address = 0.0.0.0
```

Restart both MySQL servers.

Configure replica from A (master) to B (slave)

First, upload the QM default database on MySQL server A.

On server A, we create a slave for replica to server B:

```
GRANT REPLICATION SLAVE ON queuemetrics.*
   TO 'slave_b'@'%'
   IDENTIFIED BY 'slave_b_pass';
FLUSH PRIVILEGES;
```

then we edit server B's my.cnf file and set:

```
[mysqld]
server-id = 1
replicate-same-server-id = 0
auto_increment_increment= 2
auto_increment_offset   = 2

master-host = 10.10.1.11
master-user = slave_a
master-password = slave_a_password
master-connect-retry = 60
replicate-do-db = queuemetrics

log-bin = /var/log/mysql/mysql-bin.log
binlog-do-db = queuemetrics
log-slave-updates

relay-log = /var/lib/mysql/slave-relay.log
relay-log-index = /var/lib/mysql/slave-relay-log.index

expire_logs_days        = 10
max_binlog_size         = 500M
```

After this, we create a dump of the database on A and SHOW MASTER STATUS, as in:

```
mysql> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000010 |     1067 | queuemetrics |                  |
+------------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

Then unlock the tables:

```
mysql> UNLOCK TABLES;
```

and run the following command to make server3 a slave of server2 (it is important that you replace the values in the following command with the values you got from the SHOW MASTER STATUS command that we ran on server2!):

```
mysql> CHANGE MASTER TO
                MASTER_HOST='192.168.0.101',
                MASTER_USER='slaveuser_for_s3',
                MASTER_PASSWORD='slave_user_for_server3_password',
                MASTER_LOG_FILE='mysql-bin.000010',
                MASTER_LOG_POS=1067;
```

You see that the values for *MASTER_LOG_FILE* and *MASTER_LOG_POS* come from the MASTER STATUS query.

Finally start the slave:

```
START SLAVE;
```

Then check the slave status: It is important that both *Slave_IO_Running* and *Slave_SQL_Running* have the value Yes in the output (otherwise something went wrong, and you should check your setup again and take a look at */var/log/syslog* or the MySQL logs to find out about any errors):

```
mysql> SHOW SLAVE STATUS \G
*************************** 1. row ***************************
             Slave_IO_State: Waiting for master to send event
                Master_Host: 192.168.0.101
                Master_User: slaveuser_for_s3
                Master_Port: 3306
              Connect_Retry: 60
            Master_Log_File: mysql-bin.000010
        Read_Master_Log_Pos: 1067
             Relay_Log_File: slave-relay.000002
              Relay_Log_Pos: 235
      Relay_Master_Log_File: mysql-bin.000010
           Slave_IO_Running: Yes
          Slave_SQL_Running: Yes
            Replicate_Do_DB: exampledb
        Replicate_Ignore_DB:
         Replicate_Do_Table:
     Replicate_Ignore_Table:
    Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
                 Last_Errno: 0
                 Last_Error:
```

```
               Skip_Counter: 0
        Exec_Master_Log_Pos: 1067
            Relay_Log_Space: 235
            Until_Condition: None
             Until_Log_File:
              Until_Log_Pos: 0
         Master_SSL_Allowed: No
         Master_SSL_CA_File:
         Master_SSL_CA_Path:
            Master_SSL_Cert:
          Master_SSL_Cipher:
             Master_SSL_Key:
      Seconds_Behind_Master: 0
1 row in set (0.00 sec)
```

Configure replica from B (master) to A (slave)

```
GRANT REPLICATION SLAVE ON queuemetrics.*
   TO 'slave_a'@'%'
   IDENTIFIED BY 'slave_a_pass';
FLUSH PRIVILEGES;
```

Changes to QueueMetrics

Just point each QM instance to its copy of the database as you would for two distinct instances. Make sure that the changes you make on one server are immediately available on the other one. Also, make sure that each qloaderd instance uses a distinct partition.

Native MySQL logging of queue_log data

Recent versions of Asterisk have a way of writing the *queue_log* file right to the database, without needing the qloaderd process. We **do not** advise using native logging instead of qloaderd, because:

- It's way more likely that a different networked process will be unavailable and so data will be lost; unless the disk is full, a file should always be writable. The qloaderd is - on the other side - extremely reliable and lightweight.
- The qloaderd process allows for clustering, while you can only have one Asterisk server when using native logging.

Still, this option could be useful in some scenarios (e.g. read-only disks for embedded systems running Asterisk), so we support it.

Prerequisites

- Asterisk 1.6 or 1.8, compiled with MySQL support.

Separating audio recordings in a daily folder

If you have full access to the configuration files of your Asterisk system, it is easy to save audio recordings in separate folders each day; this helps QueueMetrics in finding them quickly and makes them way more manageable (e.g. for archiving).

You can obtain this result by the following dialplan code:

```
. . . .
exten => 999,n,Set(MONITOR_FILENAME=/recordings/${STRFTIME(${EPOCH},,%Y-%m/%d)}/audio-
${UNIQUEID}.wav)
exten => 999,n,Queue(778,t,,)
. . . .
```

Asterisk should automatically create missing directories, as needed.

A problem arises if you do not have control of where your recordings will be stored, possibly because you use a GUI that does not help you with this.

It is still possible to obtain the same result by making the folder where Asterisk writes its own recordings a symbolic link that actually points to the folder that recordings are stored under. This makes implementing various NAS solutions simple to do.

The following steps will archive voice recordings by month and day as subfolders of the folder */recordings*, as in the following example:

```
/recordings/2010-11/15/audio-123456.789.wav
```

This way audio is archived by month and day.

> If you use such a solution with an external NAS, make sure you do not overload the I/O and network capacities of your server. If e.g. you record all traffic, you are going to double the Asterisk-related network bandwidth used, so beware.

Prerequisites

- A working Asterisk, sending recordings to */var/spool/asterisk/monitor*
- An external NAS or disk volume, mounted on */recordings* where you will store all audio data.
- A working QueueMetrics system

Archiving script

The following script will create the destination directory and a symbolic link to it.

Before running it, copy the contents of */var/spool/asterisk/monitor* to a different location, or they

will be lost.

```
#!/bin/bash

VAR1=`date +%Y-%m`
VAR2=`date +%d`

mkdir /recordings/$VAR1/
mkdir /recordings/$VAR1/$VAR2
chown -R asterisk.asterisk /recordings/

cd /var/spool/asterisk/
rm -rf monitor
ln -s /recordings/$VAR1/$VAR2 monitor
chown -R asterisk.asterisk monitor
```

Make the script executable.

All data you should have in */var/spool/asterisk/monitor* will be deleted on the first run of this script; so make a copy first!

The script should run every day at midnight by using a cron job.

Changes to QueueMetrics

Edit the *configuration.properties* file as follows:

```
# This is the name of the class that finds recordings. See documentation.
audio.server=it.loway.app.queuemetrics.callListen.listeners.LocalFilesByDay

#The top level directory where monitored calls are held.
#Do NOT forget to add the ending slash.
default.monitored_calls=/recordings/%YY-%MM/%DD/
```

Log off and on again. You should see the file search being much quicker now.

Making older files accessible

If you have older files, they will not be accessible unless you separate and save them by day as newer files. This can be more or less easy based on the format of your file names.

If you need it, Loway offers a file separation service that can be performed remotely in order to obtain the desired result.

Show ringing phones in the realtime page

With Asterisk 1.4 and QueueMetrics 1.7.1 is possible to have ringing phones information on the realtime page. This information should be provided to QueueMetrics through minor modifications in Asterisk configuration files. Since Asterisk 1.4, a new RINGNOANSWER event is available in the queue_log. This event shows the last agent that did not pick up the phone when ringing but, obviously, this information is available only when the phone stops ringing. With the hereby listed modifications it is possible to have ringing phone information as soon as a call enters in a particular queue. This option is available only for systems where the hotdesking is not enabled; for systems where hotdesking is required, the realtime ringing information can not be written in the queue_log but the standard RINGNOANSWER information is available directly from asterisk.

Realtime ringing information is fed to QueueMetrics by means of AGENTATTEMPT events inserted in the queue_log. This information should be generated by the Asterisk dialplan. For this reason it's mandatory to specify Local Channel extensions as members in the queues definitions: this will enable the app_queue to pass from the dialplan to start ringing phones.

We take as example the queue *queue_dps* defined in the queues.conf, as reported below:

```
[queue-dps]
announce-frequency=0
announce-holdtime=no
eventmemberstatus=no
eventwhencalled=no
joinempty=yes
leavewhenempty=no
maxlen=0
periodic-announce-frequency=0
queue-callswaiting=silence/1
queue-thereare=silence/1
queue-youarenext=silence/1
retry=5
strategy=ringall
timeout=15
wrapuptime=0
member=Local/100@from-internal-custom
member=Local/101@from-internal-custom
member=Local/102@from-internal-custom
```

Let's assume we have three members tied to the extensions 100, 101 and 102 defined in the *from-internal-custom* context. We assume that the extension 200@from-internal-custom is the entry point for the queue *queue-dps*.

The *extensions.conf* should be defined as:

```
[from-internal-custom]

exten => 100,1,System( echo "${EPOCH}|${PCHANNEL}|queue-dps|SIP/${EXTEN}|AGENTATTEMPT"
```

```
>> /var/log/asterisk/queue_log )
exten => 100,n,Dial(SIP/100)
exten => 100,n,Hangup()

exten => 101,1,System( echo "${EPOCH}|${PCHANNEL}|queue-dps|SIP/${EXTEN}|AGENTATTEMPT"
>> /var/log/asterisk/queue_log )
exten => 101,n,Dial(SIP/101)
exten => 101,n,Hangup()

exten => 102,1,System( echo "${EPOCH}|${PCHANNEL}|queue-dps|SIP/${EXTEN}|AGENTATTEMPT"
>> /var/log/asterisk/queue_log )
exten => 102,n,Dial(SIP/102)
exten => 102,n,Hangup()

exten => 200,1,NoOp("Here is a call for the queue")
exten => 200,n,Set(__PCHANNEL=${UNIQUEID})
exten => 200,n,Queue(queue-dps,,subq)
exten => 200,n,Hangup()
```

In the example above, the PCHANNEL variable is set in the extension 200 to the UNIQUEID for each incoming call in the queue. The variable is used by the extensions 100, 101 and 102 to write a signature in the queue_log file. The same should be replicated for each agent, for each queue and for each internal extension in the system.

To have real time ringing information, the last step to be performed is to modify the configuration key *default.ignoreRingNoAnswer* present in the configuration.properties file in the QueueMetrics installation folder. This key should be set to "true". This switches the QueueMetrics analyzer to the proper working modality, where RINGNOANSWER verbs are discarded, because ringing information is now provided by the AGENTATTEMPT events.

Using pre-purchased keys (PPK)

Since QueueMetrics 1.7, pre-purchased license keys can be used instead of normal license keys. This is of interest for resellers and installers who want to pre-purchase a set of "blank" activation keys and then convert them into regular activation keys immediately, as and when required.

The advantages of this model are that:

- You can generate a key as soon as your clients ask for it (no delays due to bank payments, different time zones, etc.)
- If your business model is hosting remote call-centers with QueueMetrics instances, you can purchase QM licenses that last as long as your client has paid you for; if they pay you monthly, you do not have to commit to a four-year license in advance. Also, when each key expires, you can install a different one - you invoice your client for actual usage.

Terminology used

A **pre-purchased key (PPK)** is an activation key that you purchase from Loway and specifies an activation period for a given license of QueueMetrics - for example, a PPK that looks like this:

```
XQI-LOWAY-74KKMREX6TOH1:QM-50/1/90
```

Will eventually convert into a 90-day QueueMetrics key for 50-agents and 1 Asterisk server. You can purchase PPK for most common time lengths (one month, three months, one year, four years).

A **license key** is the actual key that has to be installed in QueueMetrics in order to activate it. It can be installed manually within the *web.xml* file or can be installed over QM itself on the License page. It looks like the following example:

```
5231317C-5232476E-..... ...-307C
```

It is important to note that it must written all on one line.

Converting a PPK into a License key

In order to convert a PPK into a License key, you call a web service on a **key server** that you will be given when you purchase it.

For the license to be converted from a blank PPK to a full license, you need to pass a few parameters:

- The **end-user name** will be used to create the key name that appears on the License page of the QM instance. It should match the end-user firm name.
- The **email address** that you specify in the web service will receive a note containing all the information of the newly created license; this is for your reference only.

In order to activate the PPK, you issue an HTTP GET call to the activation URL you will be given, as in the example below (note: it has to be written all on the same line):

```
wget -O- "http://my.server/llm/licence_wsvc_gen.do?
   K_enduser=ABC&
   K_email=me@gmail.com&
   K_id=XQI-LOWAY-74KKMREX6TOH1:QM-50/1/90"
```

The output will look like the following:

```
S:OK
D:2011-08-26 15:51:58
K:5231317C-5232476E-.....
I:2011-08-25 18:46:56
X:2011-09-24
M:R
```

A successful call will start with the **S:OK** status preamble; any other status is invalid. The **D** parameter shows the current time on the activation server. The **K** parameter holds the key itself; the key can be 50-200 characters long. The **I** parameter shows the **Issue date**; subsequent calls made with the same activation key will always return the same key, with a different **D** parameter. The **X** parameter shows the expiration date for this license; note that QM may stop working at any time after the expiration date is through. The **M** parameter should be ignored.

In case there is an error, the output will look like:

```
S:KO - Key not found U:1234
D:2011-08-26 15:53:31
K:n/a
I:
X:
M:ERR
```

And will start with **S:KO**.

Automatic key installation

Although the key can be manually generated by issuing the *wget* command, the process was built so that it can easily be automated via a script or a client-management program.

We suggest installing the newly-generated key on the destination instance by calling the method **QM.setActivationKey** on the QM instance and passing the new key.

This way the whole provisioning process can be easily automated.

> For more information, see QueueMetrics' **XML-RPC manual**.

Upgrading / downgrading a PPK

You cannot upgrade or downgrade a PPK or a license generated from a PPK; if you need a larger license, you simply activate a new one. This is not an issue as PPK keys are meant to be short-lived for most environments.

Enabling log rotation in Tomcat

This document details the required steps for setting up log rotation in Tomcat running on CentOS 5, when it is installed from the Loway repository.

Prerequisites

- A working QueueMetrics instance, installed via yum

The rotation file

If you would like to setup Tomcat logrotation you can do the following:

```
vi /etc/logrotate.d/qm-tomcat6
```

This will create the logrotate file, that you will have to set up as follows:

```
/usr/local/queuemetrics/tomcat/logs/*.log {
  notifempty
  copytruncate
  daily
  rotate 10
  compress
  missingok
}

/usr/local/queuemetrics/tomcat/logs/catalina.out {
  notifempty
  copytruncate
  dateext
  daily
  rotate 10
  compress
  missingok
}
```

If you want to, you can make it take care of cleaning out temp files as well (or just do it from cron as you would normally)

```
/usr/local/queuemetrics/tomcat/logs/catalina.out {
  notifempty
  copytruncate
  dateext
  daily
  rotate 10
  compress
```

```
  missingok
  postrotate
    /bin/nice /usr/bin/find /usr/local/queuemetrics/tomcat/temp -type f -mtime +10
-exec /bin/rm {} \; > /dev/null
  endscript
}
```

Misc changes

You can also look into the CentOS logrotation settings and implement some of the following changes, depending on your needs. Even if you set the rotation to daily in qm-tomcat6, logrotate.conf overrides it to weekly.

```
vi /etc/logrotate.conf
```

Set weekly to daily

```
# rotate log files daily
daily
```

Change the backlogs from 4 week to 14 days

```
# keep 14 days worth of backlogs
rotate 14
```

Compressed logs

```
# uncomment this if you want your log files compressed
compress
```

By default logrotation runs at 4am. You can change this by editing /etc/crontab

Set it to 02:02

```
02 2 * * * root run-parts /etc/cron.daily
```

Changes to QueueMetrics

None required.

For further reading

- http://articles.slicehost.com/2010/6/30/understanding-logrotate-on-centos-part-1
- http://articles.slicehost.com/2010/6/30/understanding-logrotate-on-centos-part-2

Tuning QueueMetrics memory settings

This article was contributed by Emile Coetzee of Clarotech Consulting (South Africa), who spent a significant time working with Loway to tune the JVM memory and garbage collection policies. This article is a summary of his findings and explains how to tune the JVM in your own environment.

'I've spent a good few months working with the Loway team trying to track down a performance problem in QueueMetrics and it looks like we have finally made a breakthrough. I'm currently testing a "beta" version which is looking to be very promising. I thought I would post some of the history and some of the useful information I've gathered over time. Even though I believe the improvements Loway have made mostly contribute to the overall solution, your Java performance settings play a key role as well.'

Prerequisites

In order to understand this article fully, it is advisable to have a basic knowledge of JVM monitoring and tuning; it would be advisable to read Advanced JVM monitoring.

This article applies to recent version of QueueMetrics running on Java 6 or 7 JVMs. QueueMetrics 12.09 includes significant performance increases so it is highly recommended that you upgrade to this version in addition to following the steps detailed below.

Usage scenario

For simplicity I will be referring to QM as the application. Obviously it is served by Tomcat which uses Java. Between Tomcat and Java is where most of the troubleshooting and setting changes need to happen, but the action of running QM is what causes Tomcat and Java to become unstable.

Typical symptoms I was experiencing were either all or a combination of the following:

- QueueMetrics GUI becomes terribly slow or inaccessible
- High CPU usage caused by Java
- Out of memory errors in catalina.out
- High run time values recorded in catalina.out
- XML-RPC queries time out

For a number of clients simply setting up a cron job to restart Tomcat once a day was generally enough to prevent slowdowns from occurring (might still happen once or twice a month). This unfortunately did not work for the larger sites with 400+ agents, where I'd often have to restart Tomcat multiple times during office hours.

Monitoring basics: Java Visual VM

So where does one start? The first thing you want to do is get your Java Visual VM monitoring working. This is detailed earlier in this manual: Advanced JVM monitoring.

The 3 things you want to look at on the Monitor page are:

- CPU
- (Memory) Heap
- (Memory) PermGen

Memory Settings - Heap

After discussion with Loway they require 5/6Mb of RAM in the Heap per agent accessing the GUI. On top of that you need to allow overhead for Java as well as your reporting. At one client site I had about 400 agents. So 400 x 6 = 2400. I'm not sure how much to allocate for reports so I played it safe and rounded up to 4096 as they do pull large reports. You then use this value to set your Xms and Xmx values.

You can read how to set them in the QM Manual: https://docs.loway.ch/QueueMetrics/010_InstallingQM.html#_understanding_queuemetrics_memory_requirements .

Loway suggested that I set the Xms and Xmx values the same. Thus I used: -Xms4096M -Xmx4096M. You also want to make sure to add -server as this changes the compiler in Java. Read more here: http://stackoverflow.com/questions/198577/real-differences-between-java-server-and-java-client

Note: Be sure that your memory settings are within the limits of your physical RAM (bearing in mind that your OS and other applications like MySQL also need resources). I have 12GB of RAM in my 400 Agent server of which 8GB is in use (mostly Tomcat and MySQL).

Memory Settings - PermGen

Next thing to look at is PermGen. Often the OutOfMemory events are in fact not from Heap, but PermGen. You might see this in the catalina.out log:

```
Exception in thread "RMI TCP Connection(idle)" java.lang.OutOfMemoryError: PermGen
space.
```

I hadn't realised that just like the Heap you can also set the PermGen size. By default this seems to be about 80Mb. I experimented with 256Mb and eventually settled on 512Mb. So add these settings to your config for tomcat:

```
-XX:PermSize=512M -XX:MaxPermSize=512M
```

This change made a significant difference to the stability of QM. You can read more about PermGen here: https://blogs.oracle.com/jonthecollector/entry/presenting_the_permanent_generation

Garbage Collection

Next up is Garbage Collection. When you start reading about Garbage Collection there is a lot of information and a lot of it differs between Java versions, so make sure your reading matches your Java version. The default collector in Java 6 is selected based on your hardware and OS, but you can

force which collector to use by adjusting your tomcat settings. For single CPU setups use a serial collector: -XX:+UseSerialGC for multi CPU servers use a parallel (aka throughput collector): -XX:+UseParallelGC. Before I discovered my PermGen size problem I also tried a concurrent collector: -XX:+UseConcMarkSweepGC. This seems to perform better where PermGen size is limited. Once I increased my PermGen size I went back to UseParallelGC as Loway recommended this. My server has 2 x quad core CPUs with HT, so it makes sense to use it.

While we are talking about GC let's also look at some additional logging you can turn on for GC. You can add the following to your tomcat settings:

```
-verbose:gc -XX:+PrintGCTimeStamps -XX:+PrintGCDetails
```

This adds additional logging to your catalina.out file. Often when QM was in a hung state I would only see GC log events in catalina.out this generally coincided with Heap being maxed out. Later when I paid more attention to PermGen and CPU I would see the same effects when they were maxed. You can also add settings to alert you when Java runs out of memory.

Add the following to tomcat settings:

```
-XX:OnError=/bin/javaerrormailgen.sh
-XX:OnOutOfMemoryError=/bin/javaerrormailmem.sh
```

The scripts can contain anything you like (you could for instance trigger a restart of Tomcat). In my case I just used them to send me email.

This is '*javaerrormailgen.sh*':

```
#!/bin/sh
#

echo `date` | mail -s "SITENAME Java Error: General" me@domain.com
```

And this is '*javaerrormailmem.sh*':

```
#!/bin/sh
#

echo `date` | mail -s "SITENAME Java Error: OutOfMemory" me@domain.com
```

You can read more about GC here: http://java.sun.com/performance/reference/whitepapers/6_performance.html and here http://techfeast-hiranya.blogspot.com/2010/11/taming-java-garbage-collector.html

Troubleshooting: taking thread and memory dumps

Once you have these things in place you can now start monitoring JavaVM and Tomcat logs and capture details for feedback to Loway. Capturing jstack and jmap is detailed in the same document and the JVM setup, but I will list some changes to these commands which I found worked better.

```
jstack -F -l 21472
```

Parameters are:

- -F Forces the thread dump. I often found that in a hung state I was unable to get a thread dump without this.
- -l Prints a long listing with more info
- 21472 Is the Java (Tomcat) PID

```
jmap -F -dump:live,format=b,file=heap.bin 21472
```

Parameters are:

- -F Forces the thread dump.
- -dump Dumps into a binary format file called heap.bin. Make sure you have disk space available as this file can get very large. It does compress reasonably well using bz2 if you need to upload it somewhere for Loway.
- 21472 Is the Java (Tomcat) PID

> I have found that both these commands will pause Tomcat while the information is extracted, so running this on a working system will cause it to stop while it executes. Obviously if the system is already hung, it doesn't matter

Once I had a larger PermGen set I did see an improvement in the sense that no longer would QM simply hang, but it would still slow down. This was evident in the JVM where you could see as PermGen usage climbed so did the CPU. In the past when PermGen was maxed out it would eventually cause QM to become completely unresponsive. Once you have more overhead in PermGen it can actually recover.

QueueMetrics release 12.09 and greater requires less PermGen space for string handling, but may still require a sizeable quantity that exceeds the JVM defaults.

Final Settings

For a quick copy and paste here are my final settings for a 400+ Agent server with 2 x Quad CPU and 12GB RAM running Tomcat, MySQL & Apache. These settings must be set in the JAVA_OPTS property in */etc/init.d/qm-tomcat6*.

Bare essentials:

```
-Xms4096M -Xmx4096M -server -XX:+UseParallelGC -XX:PermSize=512M -XX:MaxPermSize=512M
```

With extra logging, JVM and Java alerts:

```
-Xms4096M -Xmx4096M -server
-Dcom.sun.management.jmxremote.port=9003
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
-verbose:gc -XX:+PrintGCTimeStamps -XX:+PrintGCDetails
-XX:+UseParallelGC -XX:PermSize=512M -XX:MaxPermSize=512M
-XX:OnError=/bin/javaerrormailgen.sh
-XX:OnOutOfMemoryError=/bin/javaerrormailmem.sh
```

Though the finer details of tuning your own JVM depend on the total system memory, whether you have a multi-core machine or not and whether you run a 32 or 64 bit server, the process described in this article will offer you data you can work with and wil be a reasonable start for large sites looking for real-life QueueMetrics implementations.

Quick JVM cheatsheet

Memory size

It is better to set the default and maximum memory setting to the same amount, so that memory can be efficiently allocated from the starts.

- *-Xmx=1000M -Xmx=1000M* set the total heap to 1000 Megabytes - this does not include Permgen
- *-XX:PermSize=512M -XX:MaxPermSize=512M* set the total PermGen size to 512M - this does not include the heap

Garbage collection models

You must choose only one of these options, based on your hardware and throughput specifications:

- *-XX:+UseSerialGC* - this is the default model but may cause large application pauses
- *-XX:+UseParallelGC* - this tries collecting memory in parallel, ideal for large heaps and multi-CPU systems. uses a parallel version of the young generation collector
- *-XX:+UseConcMarkSweepGC* - the Concurrent Low Pause Collector may offer better throughput at the price of some additional heap usage.
- *-XX:+UseParNewGC* - runs parallel GC on the New generation, avoididing promotion of useless objects to the old generation
- *-XX:+CMSParallelRemarkEnabled* - lowers remarking pauses when running with Concurrent Mark Sweep

64-bit servers

- *-XX:+UseCompressedOops* will use less heap on 64-bit systems that have less than 32G installed.

May speed things significantly up.

Debugging

The following options may be helpful in understanding what is going on:

- *-verbose:gc* - logs garbage collections
- *-XX:+PrintGCTimeStamps* - prints the timestamp of a garbage collection
- *-XX:+PrintGCDetails* - print details of a garbage collection
- *-XX:OnError=...* - runs a script on errors
- *-XX:OnOutOfMemoryError=...*' - runs a script on memory errors

Misc

- *-server* mode should always be turned on for QueueMetrics systems.

CRM Integration with QueueMetrics

SugarCRM and VTigerCRM are two widely used CRM software packages used to track contacts and opportunities that are widely used by call-centres worldwide. QueueMetrics can be easily integrated with them thanks to its ability to automatically open a specific URL from the agent's Live agent page. When integrated, any new call answered by an agent opens the associated CRM contact page, if present, or pre-fills a new form with the incoming caller ID.

Prerequisites

- A working QueueMetrics instance
- One of the supported CRM packages, already installed and working
- A working PHP+Apache instance (usually the same where the CRM is running)
- A copy of the OpenQueueMetricsAddOns package available at https://github.com/Loway/OpenQueueMetricsAddOns

Integration with SugarCRM

Integration with SugarCRM is implemented by mean of an external script that is to be copied on the Apache webroot folder. The script is supplied within the addon library present on https://github.com/Loway/OpenQueueMetricsAddOns . When properly configured, the QueueMetrics agent call history page opens an external URL for each taken call. This feature is be used to trigger the provided PHP script. The script searches among contacts in the SugarCRM database using the current caller-id, and opens either the caller's record, if present on SugarCRM database, or preloads a new contact page filling in the calling party number.

The steps for installing and configuring the script are reported below. We suppose to use the Apache webserver where SugarCRM is running. We install the script on the Apache webserver main document root:

- Download the 'nusoap library from http://sourceforge.net/projects/nusoap
- Extract the source zip on the main document web root and rename it to "nusoap", chown to the *apache* user and set proper permissions
- Copy the *QueueMetrics_SugarCRM.php* file on the web root folder, chown to the *apache* user and set proper permissions
- Generate a *queuemetrics* user for SugarCRM. The *queuemetrics* user should be able to perform searches and access contacts. Annotate the password generated by SugarCRM (when creating the *queuemetrics* user, you must use a valid e-mail address where SugarCRM will send the password)
- Open the *QueueMetrics_SugarCRM.php* file and edit the variables:
  - `$server_url` (set it to your SugarCRM server address/name)
  - `$username` (the user you created on the step above)
  - `$password` (set it to the password generated by SugarCRM for the queuemetrics user)

- Edit the *configuration.properties* file you can find on the QueueMetrics installation folder. Look for the `default.crmapp` key and change it in order to point to your Apache server. The URL should contain a reference to the current caller ID, for this reason, the dynamic token `[C]` must be included. The URL will be something like:

```
http://10.10.1.1/QueueMetrics_SugarCRM.php?callid=[C]
```

- Restart QueueMetrics and log on as agent. Open the live agent page.

Each time a new call is shown in the agent page, your browser will open the SugarCRM contact page, if any.

> The very first time the contact page will open, you'll be asked for login on the SugarCRM. This is because you want to access to SugarCRM with your proper username and password and not as the *queuemetrics* user.

A more advanced configuration

You may want to avoid the SugarCRM authentication procedure that happens when the agent receives the first call. To do this you need to implement some logic into the *QueueMetrics_SugarCRM.php* script where you retrieve the agent's username and password before redirecting to the SugarCRM page. The logic is not already implemented because we don't know how agent codes and SugarCRM username and passwords are defined in your organization, but we can provide some hints on how to retrieve the needed information from QueueMetrics.

In order to propagate the agent code to the script, you need to change the URL you defined in the *configuration.properties* file adding the dynamic agentcode parameter. You should have something like:

```
http://10.10.1.1/QueueMetrics_SugarCRM.php?callid=[C]&agentcode=[A]
```

Don't forget to restart QueueMetrics and logoff and logon again to the agent page. As soon as the script will be called, the variable `$agent` in the script will be populated with the calling agent code. What you have to do is to implement some logic that, starting from this code, retrieves the username and the password used by that specific agent to authenticate on SugarCRM.

```
if ($agent != '') {

    // Insert here your code for agent SugarCRM username and password retrieval
    // The default behavior is to use a single account for all agents
    // Using default behavior requires agent authentication on SugarCRM pages

    // $username = Sugar CRM username for this agent
    // $password = Sugar CRM password for this agent
    // $autologon = TRUE;
}
```

You need to properly set the `$username` and `$password` variables, then comment out the `$autologon` flag row. The script will generate a SugarCRM session that will be inserted in the URL used to open the contact page.

Integration with VTigerCRM

Integration with VTigerCRM could be done in two different, mutually exclusive ways.

Integrate with QueueMetrics live page

Integration with VTigerCRM is implemented through an external script to be copied on the Apache webroot folder. The script is available on the addon library available at https://github.com/Loway/OpenQueueMetricsAddOns . When properly configured, the QueueMetrics agent call history page opens an external URL for each call taken by the agent. This URL triggers the provided PHP script. The script searches among contacts in the CRM database using the current caller-id, and opens either the caller's record, if present on the CRM database, or preloads a new contact page filling in the calling party number.

The steps for installing and configuring the script are reported below. We suppose to use the same Apache webserver where the VTigerCRM is running. We install the script on the Apache webserver main document root:

- Download the *Php Zen json libraries* from http://framework.zend.com/releases/ZendFramework-1.6.1/ZendFramework-1.6.1-minimal.zip and uncompress them
- Update the PHP shared library search path (add the Zen Json folder to the `include_path` key in the `php.ini` file)
- Install the PHP Pear framework and type `pear install HTTP_Client` from a command line shell
- Copy the *QueueMetrics_VTigerCRM.php* script to the webroot folder
- Create a *queuemetrics* user on VTigerCRM, with permissions to look at contacts and retrieve the required access key generated by VTigerCRM on the user page settings
- Open the *QueueMetrics_VTigerCRM.php* file and edit the variables:
  - `$server_url` (set it to your VTigerCRM server address/name)
  - `$username` (the user you created on the step above)
  - `$accessKey` (set it to the *accessKey* generated by VTigerCRM for the *queuemetrics* user)
- Configure QueueMetrics to open an external URL for each received call. The URL should contain the caller number as callid parameter

```
http://10.10.1.1/QueueMetrics_VTigerCRM.php?callid=[C]
```

- Restart QueueMetrics and log on as agent. Open the live agent page.

Each time a new call is shown in the agent page, your browser will open the VTigerCRM contact page, if any.

The very first time the contact page will open, you'll be asked for login on the VTigerCRM. This is because you want to access to VTigerCRM with your proper username and password and not as the *queuemetrics* user.

Integrate with VTigerCRM PBX Manager Module

VTigerCRM already provide a PBX Manager module that integrates with your Asterisk PBX. With the current implementation (VTiger CRM 5.4.0) the PBX Manager is able to open a popup whith caller id information each time a new call is directed to the extension configured by a VTiger user. The PBX Manager is able also to enable a handy click-to_call feature (that is out of the scope of this document to describe).

Unfortunately, the PBX Manager does not handle calls coming from a queue. You need to make a little modification to the VTigerCRM PBX Manager module source code in order to have the popup shown for calls coming from queues, as explained below.

First of all you need to activate and configure the PBX Manager Module on VTigerCRM.

- Log in to the VTiger panel as administrator
- Go to the CRM Settings, from the top right-most icon on the page
- Click on "Module Manager"
- From the list you have on that page, click on the hammer icon present on the PBX Manager row. This lets you to access to the PBX Manager settings
- Fill in the relevant Asterisk information: server IP, AMI port, AMI username and password. Stick with 1.6 Asterisk version (we don't provide Asterisk 1.4 integration)
- Press Update
- Go to the user preferences page and fill the section 7: *Asterisk Configuration* with your internal extension. Enable the "Receive Incoming Calls" tickbox

You then need to modify the script responsible for reading Asterisk AMI events and inject calls events on the VTiger database.

- On the VTiger webroot folder, open the file AsteriskClient.php present on the `/cron/modules/PBXManager` subfolder.
- Look for the `asterisk_handleResponse2` function. There is a set of if/else block.

```
if(
    $mainresponse['Event'] == 'Newexten' && (strstr($appdata, "__DIALED_NUMBER") ||
        strstr($appdata, "EXTTOCALL"))
  ) {
    ...
    ...
    } else if($mainresponse['Event'] == 'OriginateResponse'){
    ...
```

Change it inserting a new `else` block as reported below.

```
if(
    $mainresponse['Event'] == 'Newexten' && (strstr($appdata, "__DIALED_NUMBER") ||
        strstr($appdata, "EXTTOCALL"))
  ) {
    ...
    ...
    } else if($mainresponse['Event'] == 'AgentCalled'){

            $uniqueid = $mainresponse['Uniqueid'];
            $channel = $mainresponse['ChannelCalling'];
            $splits = explode('/', $channel);
            $callerType = $splits[0];

            $splits = explode('/', $mainresponse['AgentCalled']);
            $extension = $splits[1];

            $parseSuccess = true;
    } else if($mainresponse['Event'] == 'OriginateResponse'){
    ...
```

Save and run the script as by VTiger documentation.

Each Agent could log-in on the QueueMetrics agent page and, through this, log-in on the preferred queues specifying the internal extension already set on the VTiger user preferences. As soon as a new call coming from the queue is received on that extension, a popup is presented in the VTiger pages.

Securing QueueMetrics (Tomcat) With A SSL Certificate

Reposted from deobfuscate's blog - http://deobfuscate.net/2013/08/15/securing-queuemetrics-tomcat-with-a-ssl-certificate/

Introduction

These instructions should hopefully help you with implementing on your QueueMetrics installation which runs on Tomcat 6 as of this post. A CSR will be generated on the server and processed by an internal Microsoft certificate authority. This document will also describe how to redirect HTTP traffic to HTTPS to ensure encryption. Digicert was also used for reference.

Instructions

SSH into the server and change directories to where the keystore utility resides:

```
cd /usr/local/queuemetrics/jdk1.6.0_22/bin/
```

Generate an empty Java keystore:

```
./keytool -genkey -alias foo -keystore keystore.jks
./keytool -delete -alias foo -keystore keystore.jks
```

Generate a Java keystore and key pair:

```
./keytool -genkey -alias queuemetrics.domain.net -keyalg RSA -keystore keystore.jks
-keysize 2048
```

Generate a certificate signing request (CSR) for an existing Java keystore:

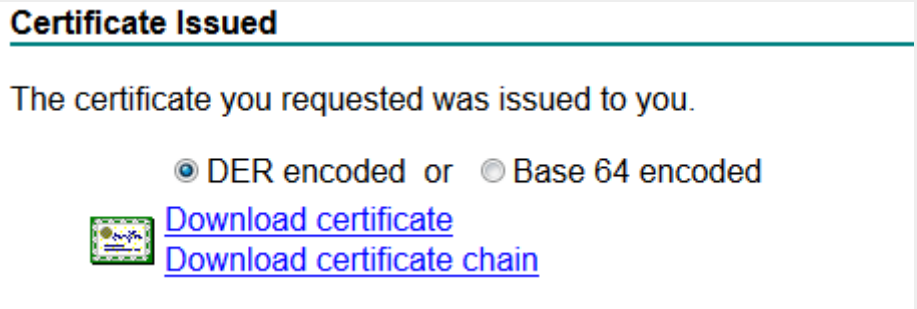
```
/usr/local/queuemetrics/jdk1.6.0_22/bin/keytool -certreq -alias
queuemetrics.domain.com -keystore keystore.jks -file queuemetrics.domain.com.csr
```

Copy the CSR:

```
cat queuemetrics.domain.net.csr
```

Go to *https://ca.domain.com/certsrv/* and request certificate then select advanced certificate and paste in the CSR. This process will be the same but have a different interface when using another CA.

Download the certificate (not certificate chain) with DER encoding as well as the CA root certificate as both will be needed.



Upload the certificates to the server and move the files to */usr/local/queuemetrics/jdk1.6.0_22/bin/*

Install the CA root certificate:

```
keytool -import -trustcacerts -alias root \
        -file ca_root.cer -keystore keystore.jks
```

Install the site certificate:

```
/usr/local/queuemetrics/jdk1.6.0_22/bin/keytool -import -trustcacerts \
   -alias server -file queuemetrics_domain_net.cer \
   -keystore queuemetrics.domain.net.jks
```

Edit the Tomcat configuration file and enter the path to the certificate in the keystoreFile section, enter the keystore password in the keepass section, SSLEnabled should be set to true, and the port should be set to 443 as that is the default port for HTTPS.

```
nano /usr/local/queuemetrics/tomcat/conf/server.xml
```

```
<Connector port="443" maxHttpHeaderSize="8192" maxThreads="150"
minSpareThreads="25" maxSpareThreads="75" enableLookups="false"
disableUploadTimeout="true" acceptCount="100"
secure="true" SSLEnabled="true" clientAuth="false"
sslProtocol="TLS" keyAlias="server"
keystoreFile="/usr/local/queuemetrics/jdk1.6.0_22/bin/certificates
queuemetrics_domain_net.jks"
scheme="https" keypass="secret" />
```

This link will show you how to also have HTTP requests redirect to HTTPS.

Restart QueueMetrics:

```
service queuemetrics restart
```

Once this is done you should be able to access your QueueMetrics (or other Tomcat based website/application) by going to *https://queuemetrics.domain.com* and assuming you have the CA installed on your system as well will not be told that the certificate is invalid.

Removing obsolete Diffie-Hellman ciphers

On modern browsers, the default Tomcat SSL configuration will show an error like "Server has a weak, ephemeral Diffie-Hellman public key". In order to overcome this error, you have to disable the weak Diffie-Hellman ciphers in your server.xml file.

This is how the *Connector* stanza should look like:

```
<Connector port="8443"
maxHttpHeaderSize="8192"
maxThreads="150"
minSpareThreads="25"
maxSpareThreads="75"
enableLookups="false"
disableUploadTimeout="true"
acceptCount="100"
secure="true"
SSLEnabled="true"
clientAuth="false"
keyAlias="<yourdomain.com>"
ciphers="TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
TLS_RSA_WITH_AES_128_CBC_SHA256,TLS_RSA_WITH_AES_128_CBC_SHA,
TLS_RSA_WITH_AES_256_CBC_SHA256,TLS_RSA_WITH_AES_256_CBC_SHA"
sslProtocol="TLS"
keystoreFile="<your keystorefile>"
scheme="https"
keypass="<your keystore password>" />
```

Thanks to Julian Franke for the suggestion.

Offlining a part of the queue_log table

You may want to put offline a part of the *queue_log* table in order to reduce disk consumption on the server. This should not give you a large performance gain - as all accesses to the table are index-based, performance will be stable even with very large tables - but will make backups easier and quicker.

Prerequisites

- A working QueueMetrics instance with MySQL storage

> Make sure you have enough disk space to carry out these operations - one of the few things MySQL does not like is finishing up disk space on a live system. So you should have at least enough space to make a full copy of the queue_log table, with indexes and all.

> The following procedure is to be run when the system is idle, as it uses a lot of I/O for large tables and may lock tables. On a very large table each query might take tens of minutes to run, so beware.

You may keep on having qloaderd uploading data to your normal table, but it would be better not to have QM running queries on it.

> you need to make a complete database backup before attempting this.

Moving data to a temporary table

First we create a "backup" table called *queue_log_old* to hold data we don't need anymore.

```
CREATE TABLE queue_log_old LIKE queue_log
```

Decide a timestamp in the past that will be your "cutoff point" - in our case it is 1326826989, that stands for "Jan 17 2012 @ 7:03:09pm UTC".

> There are a number of services that will do the conversion for you. For example, see http://www.unixtimestamp.com/index.php

You need also to know the partition you want to move.

```
INSERT INTO queue_log_old
  SELECT *
    FROM queue_log
   WHERE partition = 'P001'
     AND time_id < 1326826989
```

And then delete it from the queue_log table:

```
DELETE FROM queue_log
      WHERE partition = 'P001'
        AND time_id < 1326826989
```

And optimize the queue_log table so space is claimed back:

```
OPTIMIZE TABLE queue_log
```

at this point, you might run a backup of the new "queue_log_old" table, save its contents to disk and delete it.

Backing up the temporary table

You may use the *mysqldump* tool to create a backup of the table:

```
$> mysqldump queuemetrics queue_log_old > queue_log_old.sql
$> bzip2 queue_log_old.sql
```

You may then drop the table to have it removed.

```
DROP TABLE queue_log_old
```

Restoring data

In case you need to put data back on the main table (e.g. because you want to be able to access it again through QM):

```
INSERT INTO queue_log
     SELECT *
       FROM queue_log_old
```

Removing duplicate rows from the queue_log table

If multiple instances of the qloaderd were run at the same time, it is possible that data was loaded multiple times. The correct way to handle this is to use the *Data Queue Partial Update* mode - see http://manuals.loway.ch/QLoader-chunked/ar01s03.html - and reload any queue_log files involved.

If this is not possible, for example, because that data was rotated out of the system and original queue_log files are unavailable, it is possible to do this operation at the SQL level.

> If you do not feel confortable doing this, Loway offers *Per-Incident Support Tickets* so that a qualified technician may connect to your QueueMetrics system and perform this procedure for you.

Prerequisites

- A working QueueMetrics instance with MySQL storage
- About 2x the current size of the *queue_log* table as free disk space
- Make sure all qloaderd instances are stopped

> Make sure you have enough disk space to carry out these operations - one of the few things MySQL does not like is finishing up disk space on a live system. So you should have at least enough space to make a full copy of the queue_log table, with indexes and all.

> The following procedure is to be run when the system is idle, as it uses a lot of I/O for large tables and may lock tables. On a very large table each query might take tens of minutes to run, so beware.

> You need to make a complete database backup before attempting this.

Reality check

Before you attempt this procedure, make sure you run the following query:

```
SELECT `partition`, time_id, unique_row_count, count(*)
  FROM queue_log
 GROUP BY `partition`, time_id, unique_row_count
HAVING count(*) > 1;
```

It should return exactly zero results. If it is not so, the following procedure may not work correctly.

Also, you should be aware of how much data is available and under which partitions:

```
SELECT `partition`, count(*)
  FROM queue_log
 GROUP BY `partition`
 ORDER BY `partition` ASC
```

Make sure that you have no partitions which name starts with "o_".

Spotting duplicate rows

If you want to know **which** rows are duplicates, this is easy to perform in SQL:

```
SELECT  `partition`, `time_id`, `call_id`, `queue`,
        `agent`, `verb`, `data1`, `data2`, `data3`, `data4`, `data5`, `serverid`
  FROM queue_log
 WHERE `partition` = "P001"
   AND `time_id`   > 0
 GROUP BY `partition`, `time_id`, `call_id`, `queue`,
          `agent`, `verb`, `data1`, `data2`, `data3`, `data4`, `data5`, `serverid`
HAVING count(*) > 1
```

Make sure you set the `partition` correctly and that you specify a minimum `time_id` so that you can avoid applying the query to the whole table, which might be very slow.

Loading unique rows

First, we make a backup of the queue_log table into a temporary table (this is not strictly needed, as you should already have a full database backup before attempting this - but may come in handy in case you need to restore quickly):

```
CREATE TABLE queue_log_backup  AS
SELECT *
  FROM queue_log;
```

Then we move all data from their current partition to a partition of the same name but prefixed with "o_". From now onwards, QueueMetrics will not find data in the database until the procedure completes.

```
UPDATE queue_log
   SET `partition` =  concat( "o_", `partition`);
```

Now we keep the highest unique_row for each duplicate row belonging to a partition starting with "o_" and load it to a partition with their original name:

```
INSERT INTO queue_log
```

```
SELECT substring( partition, 3) as `partition`, `time_id`, `call_id`, `queue`,
       `agent`, `verb`, `data1`, `data2`, `data3`, `data4`, `data5`, `serverid`,
       max(`unique_row_count`)
  FROM queue_log
 WHERE `partition` LIKE 'o_%'
 GROUP BY `partition`, `time_id`, `call_id`, `queue`,
          `agent`, `verb`, `data1`, `data2`, `data3`, `data4`, `data5`, `serverid`
```

This query might take a while to run. Data will reappear in the "correct" partition, and is still available in the temporary "o_" partition as well.

Now check with QueueMetrics that you have no more duplicate rows.

If you still have, run a database restore.

Cleaning up

First we remove the temporary table:

```
DROP TABLE queue_log_backup;
```

Then we remove data from temporary partitions:

```
DELETE FROM queue_log
 WHERE `partition` LIKE 'o_%';
```

After large inserts and deletes, it is better to resort the table and optimize it so that QueueMetrics can access it efficiently.

```
ALTER TABLE queue_log
   ORDER BY `partition` ASC, `time_id` ASC, `unique_row_count` ASC;

OPTIMIZE TABLE queue_log;
```

Do not forget to restart the *qloaderd* when done.

Fixing broken indexes on the queue_log table

Sometimes indexes on the `queue_log` table end up in a broken state. They keep existing, but the database engine is not able to use them any longer. What you see is that queries asking for a large dataset are very very slow, and the system is nearly unusable. The database uses a lot of CPU and I/O while QueueMetrics is sitting idle most of the time.

> If you do not feel confortable doing this, Loway offers *Per-Incident Support Tickets* so that a qualified technician may connect to your QueueMetrics system and perform this procedure for you.

Does this apply to you?

The first symptoms of this issue are:

- very high load on the database
- all of a sudden, reports that used to take seconds start taking minutes

To go further into the issue, you first need to set up slow query logging. See e.g. https://stackoverflow.com/questions/2403793/how-can-i-enable-mysqls-slow-query-log-without-restarting-mysql - a log file will be created with all slow queries.

When you do, you will find entries like the following:

```
# Time: 180624  1:56:34
# User@Host: queuemetrics[queuemetrics] @ localhost [127.0.0.1]
# Thread_id: 18  Schema: queuemetrics  QC_hit: No
# Query_time: 289.292460  Lock_time: 0.000124  Rows_sent: 1063140  Rows_examined:
47432990
SET timestamp=1529819794;
SELECT `time_id` , `call_id` , `queue` , `agent` , `verb` , `data1` , `data2` ,
`data3` , `data4` , `data5` , `unique_row_count`   FROM queue_log  WHERE  `partition`
= 'P001' AND       (`time_id` >= '1527804000' AND `time_id`<= '1529819505' )  AND
`queue` IN ( '', 'NONE'  ,  '100'  , '200'  , '303'  )   ORDER BY `time_id` ASC ,
`unique_row_count` ASC;
```

In this case, the database scanned 47M rows to fetch only 1M and took almost five minutes. Not good. In theory, all accesses on the table should be by index, so the number of rows examined and sent should be more or less the same.

> This problem is often misleading because if you run an an "EXPLAIN" the database will say that there is a multiple column index and that it is being considered and in fact used. But the query is still slow and there is a big difference between rows sent and examined.

Prerequisites

- A working QueueMetrics instance with MySQL storage
- A backup of the current database
- At least 2x the current size of the *queue_log* table as free disk space
- Make sure all Qloaderd/Uniloader instances are stopped. If not you risk data loss.
- QueueMetrics is switched off.

Make sure you have enough disk space to carry out these operations - one of the few things MySQL does not like is finishing up disk space on a live system. So you should have at least enough space to make a full copy of the queue_log table, with indexes and all.

The following procedure is to be run when the system is idle, as it uses a lot of I/O for large tables and may lock tables. On a very large table each query might take tens of minutes to run, so beware.

You need to make a complete database backup before attempting this.

How it works

To force the database to create a new index, we create a new `queue_log_2` table with the correct indexing schema, copy all data into it and rename it to `queue_log`. After it is renamed, we restart QueueMetrics that will start reading from the new table.

Procedure

First, we create a copy of the `queue_log` table as `queue_log_2`.

```
CREATE TABLE queue_log_2 LIKE queue_log;
```

We copy all data from the old table to the new one. As the schema is identical, we do not need to define the exact fields:

```
INSERT INTO queue_log_2
     SELECT *
       FROM queue_log
   ORDER BY `partition` ASC, `time_id` ASC, `unique_row_count` ASC;
```

The query above might take a significant amount of time, as data will be copied and indexes rebuilt.

Now we rename the old table to `queue_log_old`, and rename the new table to `queue_log`.

```
RENAME TABLE queue_log to queue_log_old;

RENAME TABLE queue_log_2 TO queue_log;
```

We can restart QueueMetrics and Uniloader/QLoader.

Cleaning up

After testing that everything is okay and that queries are now way quicker, we can drop the `queue_log_old` table:

```
DROP TABLE queue_log_old;
```

Printing all QueueMetrics users and agents in one go

It is sometimes useful to get a tabular output of:

- all users in a QueueMetrics instance that are currently enabled and not holding the masterkey (as no users should hold the masterkey in production)
- their current class and extra keys
- their configured agent aliases, if any
- their current location, supervisor and agent group
- the set of queues they are supposed to work on

so that you can see at a glance the current state and whether there are any misconfigurations.

The following query does just that:

```
SELECT U.login AS LOGIN,
  U.real_name AS NAME,
  C.nome_classe as CLASS,
  C.chiavi as CLASS_KEYS,
  U.chiavi_utente as USER_KEYS,
  U.ultimo_logon as LAST_LOGON,
  AG.descr_agente as AGENT_CODE,
  AG.aliases as FRIENDLY_NAMES,
  LOC.loc_name as LOCATION,
  SUP.login as SUPERVISOR,
  GRO.group_name as AGENT_GROUP,
  ( SELECT group_concat(CP.nome_coda SEPARATOR ', ') AS QUEUES
       FROM code_possibili CP
     WHERE CP.agenti_membri LIKE concat( '%', U.login, '%')
        OR CP.agenti_spilloff_1 LIKE concat( '%', U.login, '%')
        OR CP.agenti_spilloff_2 LIKE concat( '%', U.login, '%')
  ) AS QUEUES

FROM arch_users U
JOIN arch_classes C on C.id_classe = U.classe
LEFT JOIN agenti_noti AG on  AG.nome_agente = U.login
LEFT JOIN locations LOC on LOC.id_location = AG.location
LEFT JOIN arch_users SUP on SUP.user_id = AG.supervised_by
LEFT JOIN agent_groups GRO on GRO.id_group = AG.group_by
WHERE U.abilitato = 1
  AND U.masterkey = 0
ORDER BY U.login ASC
```

By running it against the QueueMetrics database from any MySQL shell, you will get a complete report you can export or reprocess as you see fit.

Bulk renaming audio files

Some PBXs - especially the ones based on FreePBX 2.11 - may save a prepending + sign in audio file names. Those file names will look like:

```
q-123-+12225555688-20140520-071636-140058812.32217.wav
```

Where *+12225555688* is the number dialed. These names are not compatible with some versions of QueueMetrics.

Removing the initial character in new files

Add the following to */etc/asterisk/extension_custom.conf*:

```
[from-pstn-custom]
exten => _X!,1,GotoIf($["${CALLERID(num):0:1}" = "+"]?plusstart:noplusstart)
exten => _X.,n(plusstart),NoOp(Changing Caller ID number from ${CALLERID(num)} to
${CALLERID(num):1})
exten => _X.,n,Set(CALLERID(num)=${CALLERID(num):1})
exten => _X.,n,Set(CALLERID(ANI)=${CALLERID(num)})
exten => _X!,n(noplusstart),NoOp(Caller ID does not need adjustment)
```

Then load the new dialplan code with:

```
asterisk -rx 'dialplan reload'
```

This will fix all calls going forward.

Renaming old files to remove the plus character

To fix all calls retroactively, you need to rename the existing call recordings like in:

```
for d in `find /var/spool/asterisk/monitor -mindepth 3 -type d`;do
    cd $d;
    for i in *.wav; do
        mv $i ${i/+/}
    done
done
```

This recipe was originally contributed by one user on our forums who wished to remain anonymous.

Setting up QueueMetrics WebRTC Softphone

0. Introduction

Since QueueMetrics 19.04.1, QueueMetrics offers a stable WebRTC softphone based on the JsSIP library. This softphone can be used by agents, through the **QueueMetrics Realtime Agent Page**, or by supervisors and administrators through the **Wallboard Page**. This new feature allows agents to work without the need for a physical phone, while supervisors can monitor calls in realtime without leaving the wallboard page. In this tutorial we will go through all the steps needed to set up and start using the softphone.

> This tutorial is intended for QueueMetrics 19.04.1. If you have an older version of QueueMetrics, please consider updating to the current release.

1. QueueMetrics TLS Setup

The WebRTC technology requires both QueueMetrics and Asterisk to work with HTTPS to guarantee the appropriate level of security. For this reason we need to make sure that QueueMetrics is running in https.

For detailed instructions on how to setup QueueMetrics in HTTPS, you can refer to Serving QueueMetrics over Apache/SSL .

> If you are using FreePBX, chances are that port 80 and port 443 are already in use. In that case just substitute port 80 and 443 in the manual with 81 and 444.

2. FreePBX Extension Setup

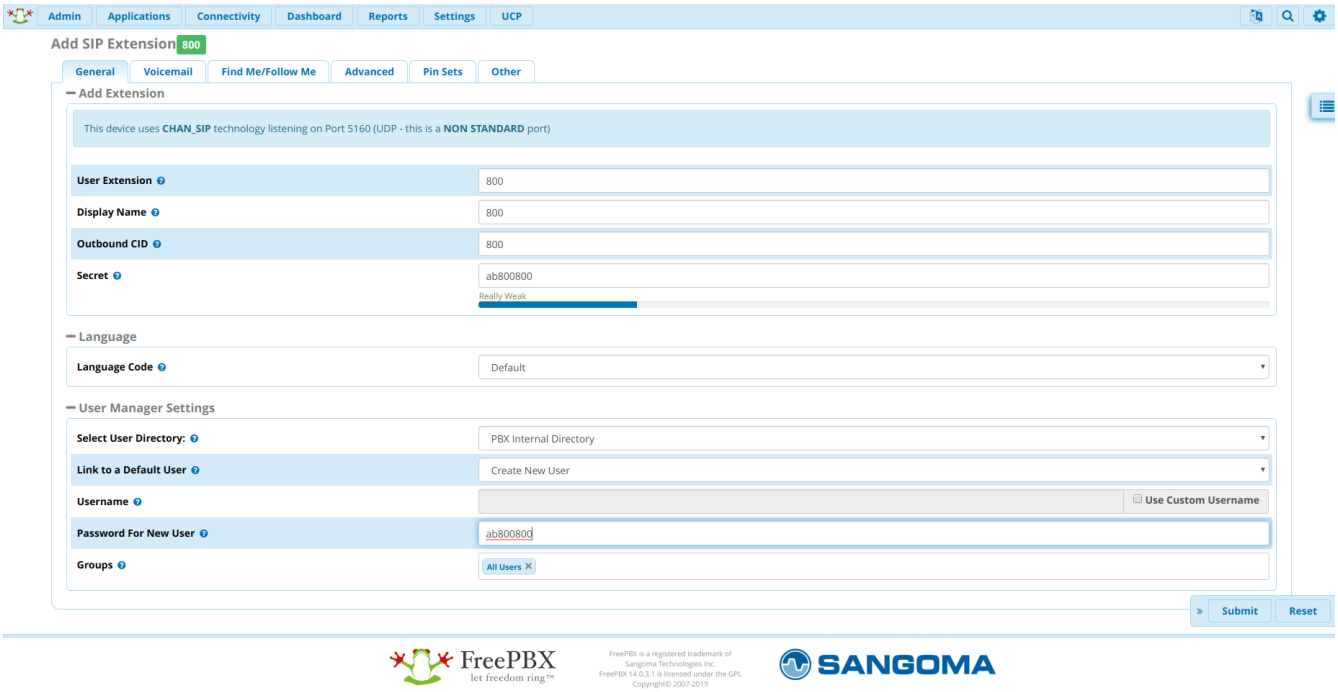
> For the sake of this tutorial, we will use FreePBX 14.0.3.1. Any later version should work fine, as well as previous releases, as long as they have WebRTC support and TLS enabled.

What we need to do now, is to create an extension for one of our agents. This will allow the agent to register to the PBX using its own QueueMetrics Softphone, by using this extension.

To create an extension, access FreePBX Administrator page and head to **Applications** ⇒ **Extensions** ⇒ **Add extension** ⇒ **Add new Chan_SIP extension**.

Here we will enter the data for our extension and linked user.

Click on **Submit** and then **Apply Config** to finalize the extension creation.



We then need to enable WebRTC for this extension so we click on the edit symbol under Actions and we go to the Advanced tab.

Here we need to set the following options:

Transport must be set to **All - WSS Primary**

Enable AVPF must be set to **Yes**

Force AVP must be set to **Yes**

Enable ICE Support must be set to **Yes**

Enable rtcp Mux must be set to **Yes**

Enable Encryption must be set to **Yes (SRTP only)**

Click On **Submit** then on **Apply Config**

Now we need to go to **Admin** ⇒ **User Management** to enable WebRTC for this user.

Click on the **Edit Symbol** under **Action** then go to the **UCP** tab.

From there we go to the **Phone** tab.

Note: at this point if this is a fresh FreePBX Installation, a message will appear saying the following:

```
You have no default certificates setup in Certificate Manager
```

This means that the extension cannot be allowed to be used through WebRTC, because it needs an **SSL certificate** to ensure an HTTPS connection for the extension.

To fix this, we have two different ways:

- Create a **Self-Signed SSL Certificate**
- Purchase a **valid SSL Certificate**

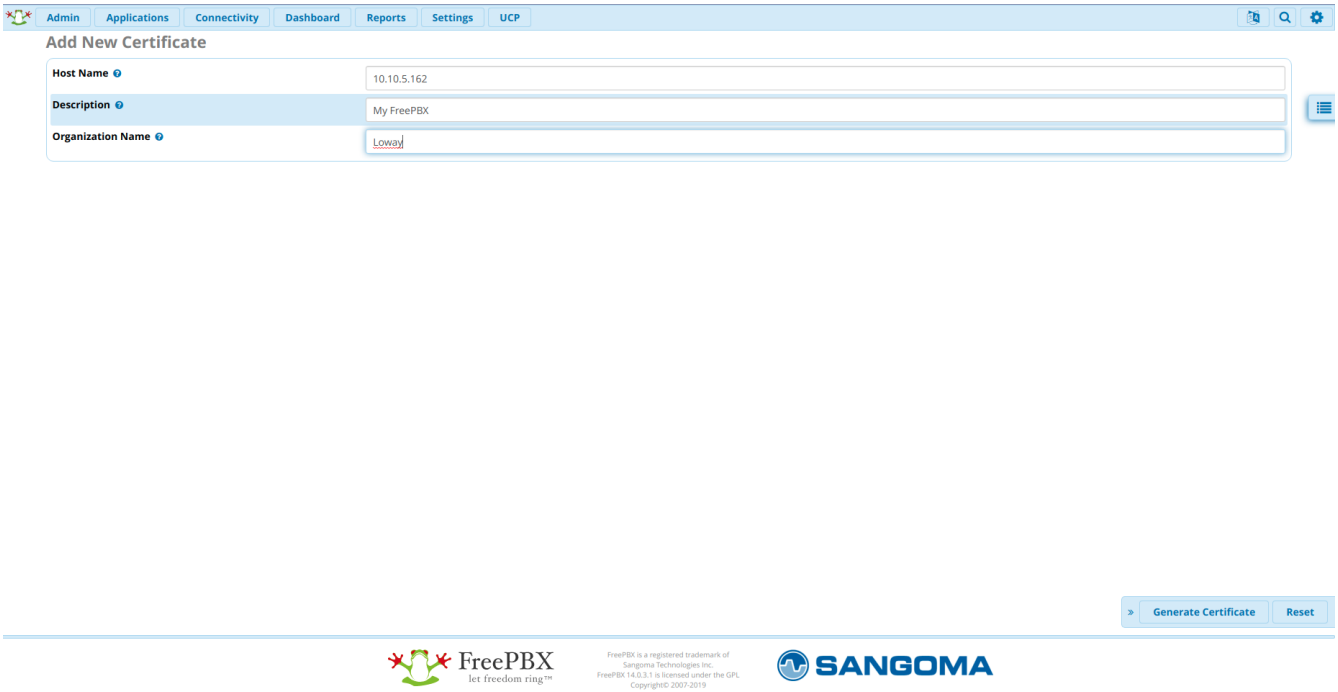
Creating Self-Signed SSL Certificate

Creating your own Self-Signed Certificate, is a valid solution if you are only connecting to your extension from inside your own network. A self signed certificate is essentially a "homemade" certificate, not guaranteed by a Certificate authority.

To create a self signed certificate, we need to go to **Admin** ⇒ **Certificate Management**.

Here we can click on **New Certificate** ⇒ **Generate Self-Signed Certificate**, and input the following information:

- Host name: The ip address of our FreePBX
- Description: A description of the certificate
- Orgnization name: Your company name



We click on **Generate Certificate** to generate the certificate.

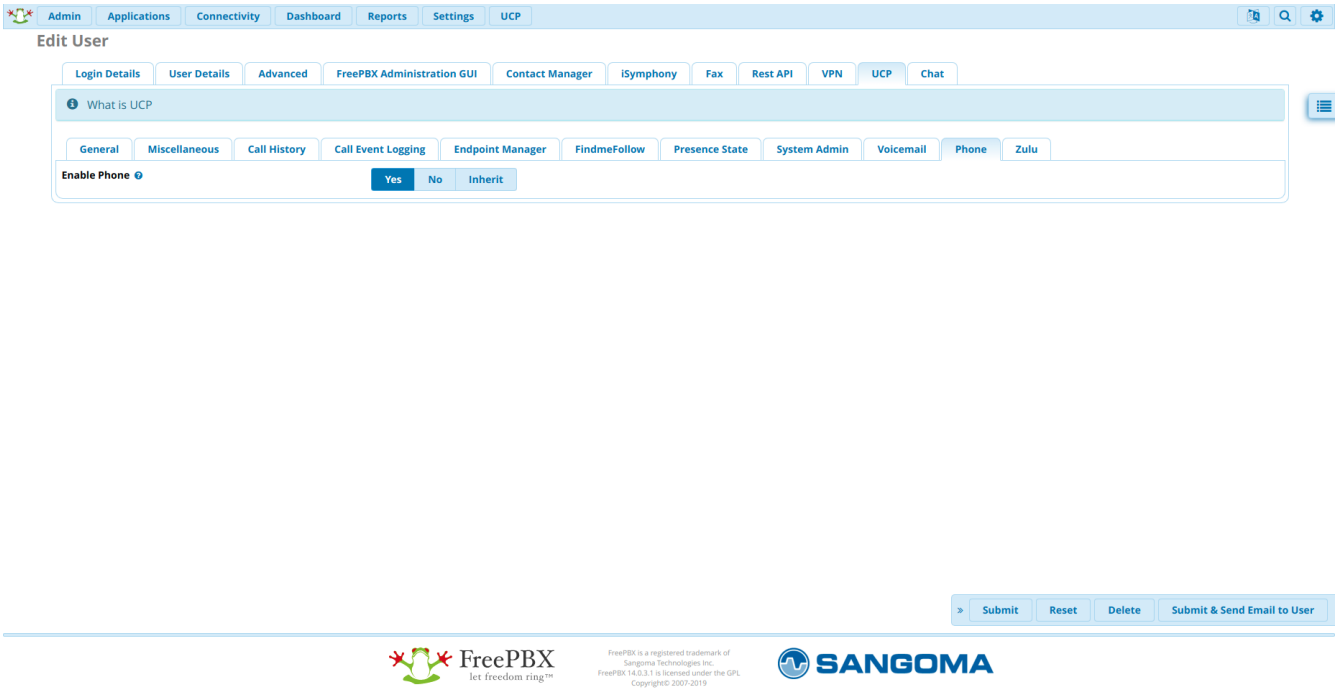
This will bring us back to the **Certificate Management** page, where we need to do one last thing.

If we do not have other certificates set on the system, we need to move our mouse to the empty cell beneath the **Default** column in the table. This will show a grey **Check** symbol in the cell, basically allowing you to set this certificate as the default SSL Certificate for your extensions. We need to click on the empty cell to set this certificate as the default certificate.

Certificate	Description	Type	Default	Action
10.10.5.162	My FreePBX	Self Signed	✔	

Now that we set this certificate as the default for our system, we can head back to the **Phone** tab in user 800's settings.

We should now see that there is an option to **Enable Phone**, that we have to set to yes.



and set **Enable WebRTC Phone** to yes, we click on Submit then Apply Config.

Then we need to go back to the extension settings in **Applications** ⇒ **Extensions** ⇒ **Edit Extension** (pencil icon on the right) ⇒ **Advanced**, and set

- **Enable DTLS** to yes

We click on Submit then Apply Config, and move on to the next section.

> If you need to import a valid certificate instead of creating a self signed one, you just need to click on the **Upload Certificate** option, under **New Certificate** in **Certificate Management**, and follow the instructions there. Remember to set the certificate as the default certificate after you are done.

3. Enable DTMF Feature code

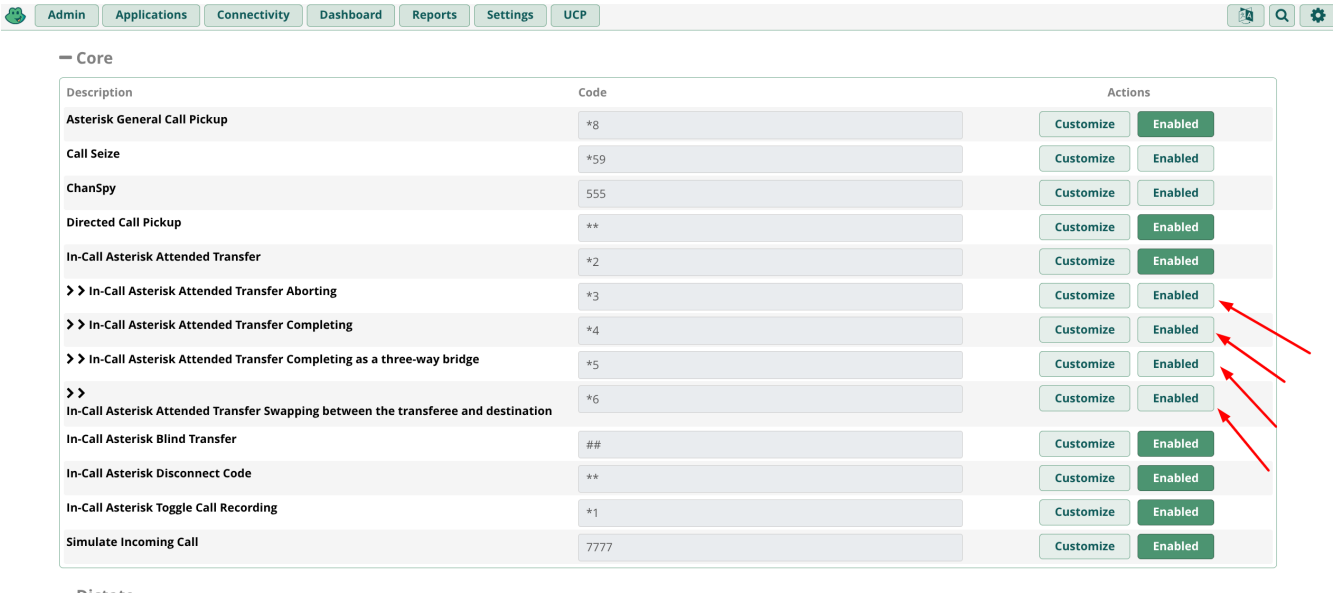
Starting from FreePBX 16 and above, you must manually enable the DTMF-based feature codes in order for softphone transfer functions to work properly. These features include:

- Cancel
- Confirm
- Join
- Swap

To enable these:

1. Go to Admin → Feature Codes
2. In the Core section, locate the following feature codes:
    - In-Call Asterisk Attended Transfer Aborting

    - In-Call Asterisk Attended Transfer Completing
    - In-Call Asterisk Attended Transfer Completing as a three-way bridge
    - In-Call Asterisk Attended Transfer Swapping between the transferee and destination
3. Make sure each one is enabled (green toggle).
4. Do not change the default codes — only enable them.
5. Click Apply Config to activate the changes.



Configuring QueueMetrics

To Create and agent, linked to the Asterisk Extension we created earlier, go to **Edit Agents** → **Create New**.

We call the agent **Agent/800** and fill in the Agent Description field as we please. We then fill the following fields:

- **WebPhone Username** with the username we previously set (in this case 800).
- **WebPhone Password** with the password we previously set (in this case ab800800).
- **WebPhone Realm** with the address of our FreePBX machine. (in this case 10.10.5.162).

Agent Detail

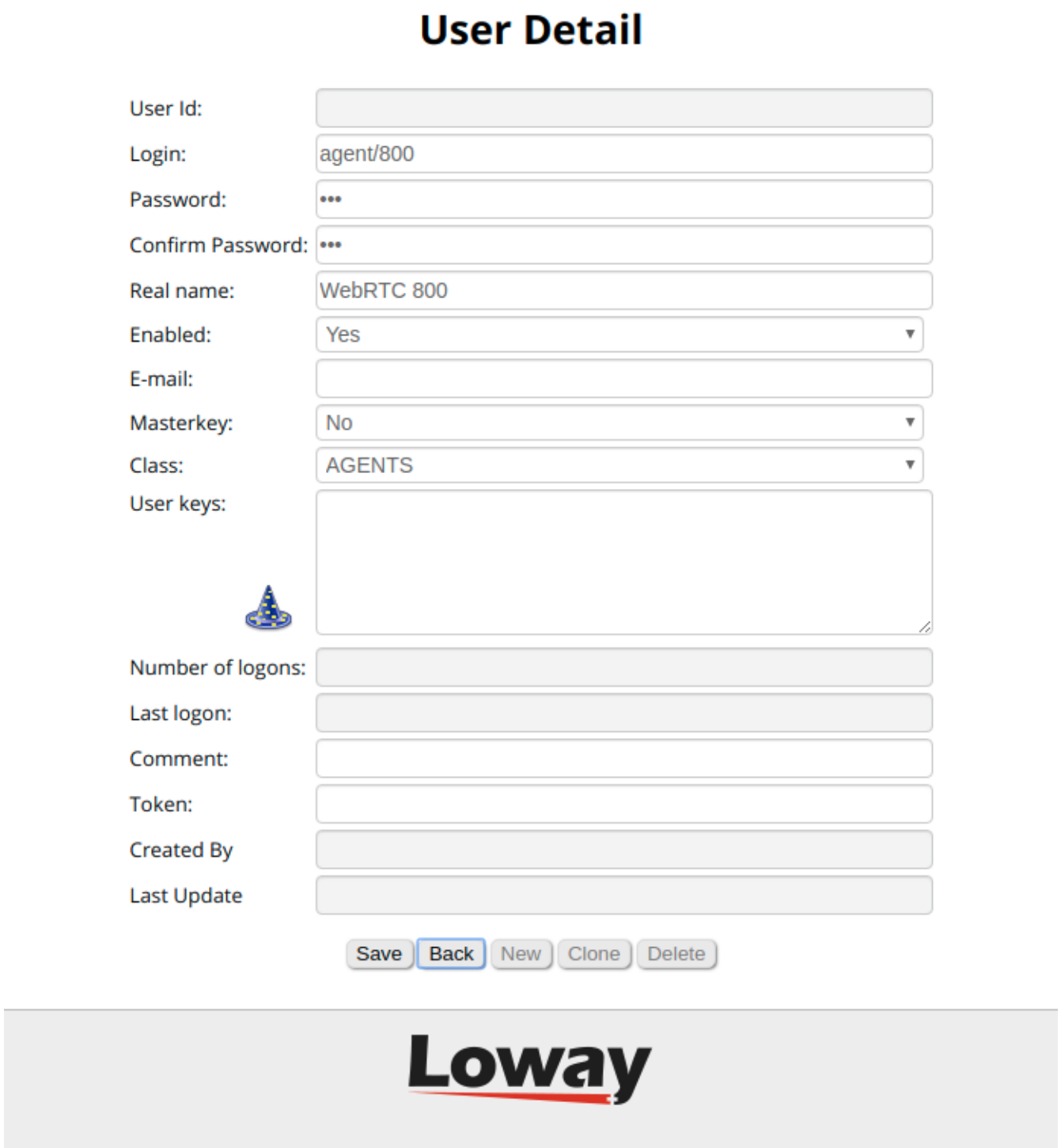
Asterisk agent code: E.g.: Agent/101	Agent/800
Agent description:	WebRTC 800
Asterisk aliases: Separate multiple aliases with a "|" symbol	
Server	-
Agent location:	-
Agent group:	-
VNC monitoring URL:	Test
Current terminal:	
Instant messenger address:	Test
WebPhone Username:	800
WebPhone Password:	ab800800
WebPhone Realm:	10.10.5.162
WebPhone SIP Uri:	
Supervisor:	-
Agent Keys:	
Payroll Code:	
External Reference ID:	
Created By	
Last Update	

Save Back New Clone Delete

We click on Save then go back to the **Homepage**.

Now we need to create a **User** for Agent/800, so we go to **Edit Users** → **Create New**.

We set the **Login** field to 800, the **Password** to 800 and we set the **Class** field to **AGENTS**.



We click on Save and we go back to the homepage.

The last thing we need to do is to set one parameter in the **Explore System Parameters** page. Scroll down to the end of the page untill you see the following parameter: **Web Socket URL for the connection**.

We need to set this parameter to:

```
wss://10.10.5.164:8089/ws
```

Using the correct IP address for our FreePBX machine. Then we **Save** at the bottom.

Agent Page Softphone Settings

Maximum interval in seconds between WebSocket reconnection attemps. Default value is 30.	30
Minimum interval in seconds between WebSocket reconnection attemps. Default value is 2.	2
Time (in seconds) after which an incoming call is rejected if not answered. Default value is 60.	30
List of ICE Servers to use.	
Web Socket URL for the connection.	wss://10.10.5.162:8089/ws

Save Back

After saving, you need to log off and on again for the parameters to be loaded.

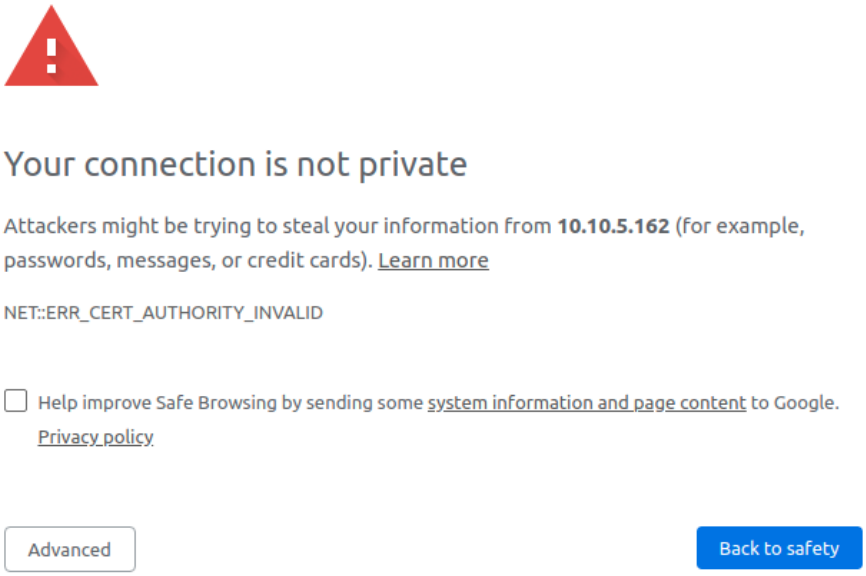
The last step we need to do, is to add a security exception in our browser for the WebSocket we just configured.

this step is necessary only if you are using a **Self Signed Certificate**.

To do this, **log out** from QueueMetrics and go to the following url:

```
https://10.10.5.162:8089/
```

Where 10.10.5.162 must be substituted with your own PBX Address. This will trigger a security warning from your browser (Probably), and you need to acknowledge the warning and add the source as a security exception.



Once you do that, you should see a **Not Found** error. This is normal and it means that the exception was added correctly.

Now we're good to go! We go back to our QueueMetrics login page, and we login as Agent/800.

From the Widget Menu we select Soft Phone and here is our WebRTC client.

QueueMetrics

Wallboard Softphone

Since QueueMetrics 19.04.1, the Wallboard page includes a Softphone panel, with the same features as the one present in the agent page.

After completing the previous configuration process, it is very easy to setup, as we merely need two steps to accomplish this:

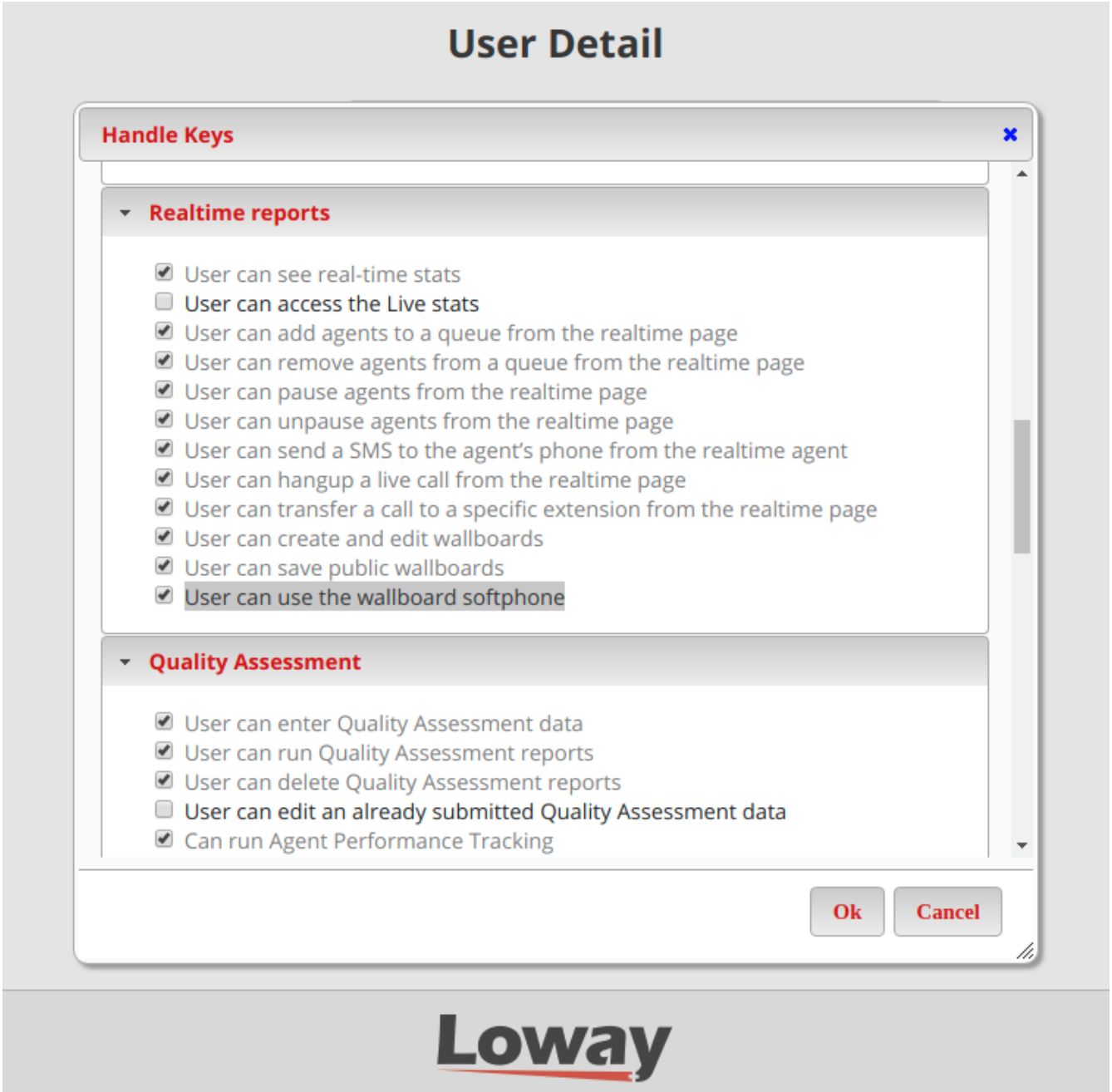
- Add the correct security key to the user
- Set the default configuration for the Softphone

Adding the Softphone security key

To do this, we need to head to the **User Configuration** page, and open the settings for the user to whom we want to assign the ability to use the softphone.

Here we click on the **Wizard hat** icon, that will open the security key menu. Here we assign the following key to the user:

- User can use the wallboard softphone



Once we do this, we can save and go back to the homepage.

Setting the default softphone configuration

From QueueMetrics' Homepage we go to **Edit System Parameters** or **Explore System Parameters**.

The properties that we can set in the **Edit System Parameters** page are the following (with some example values):

- default.wallboardphone.connection_recovery_max_interval=30
- default.wallboardphone.connection_recovery_min_interval=2
- default.wallboardphone.display_name=200
- default.wallboardphone.no_answer_timeout=60
- default.wallboardphone.password=ab200200

- default.wallboardphone.username=200
- default.wallboardphone.server=10.10.5.181
- default.wallboardphone.iceservers=stun:stun01.sipphone.com|stun:stun01.sipphone.com|stun:stun.fwdnet.net
- default.wallboardphone.websocketurl=wss://10.10.5.181:8089/ws

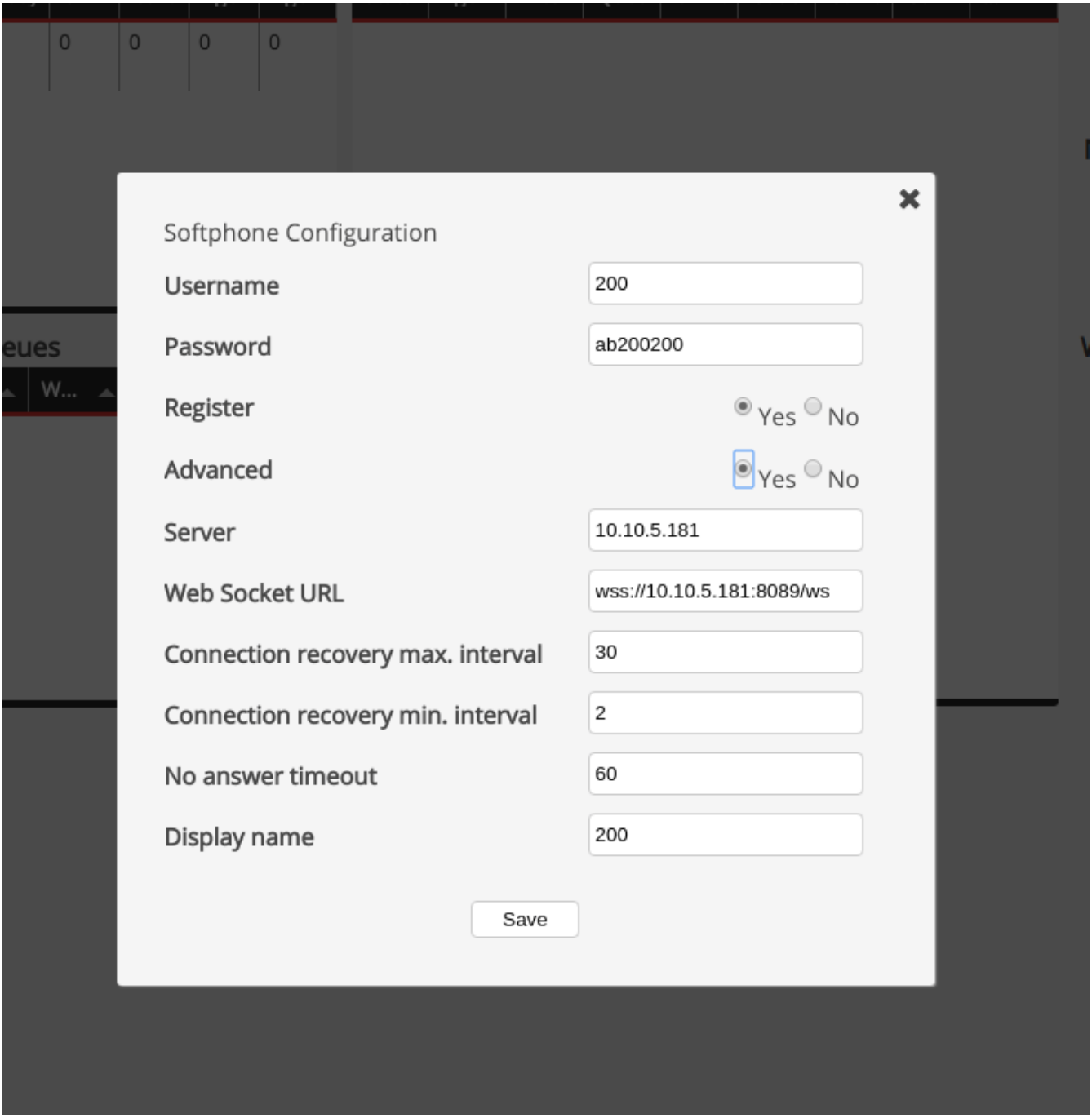
in the **Explore System Parameters** page you can search for each parameter in the top search bar, and set it from the resulting input box.

these values will be the **default for each user** that uses the wallboard softphone, if you want to change individual usernames and passwords for each user, this is done by changing the settings directly in the wallboard **Softphone Settings** panel.

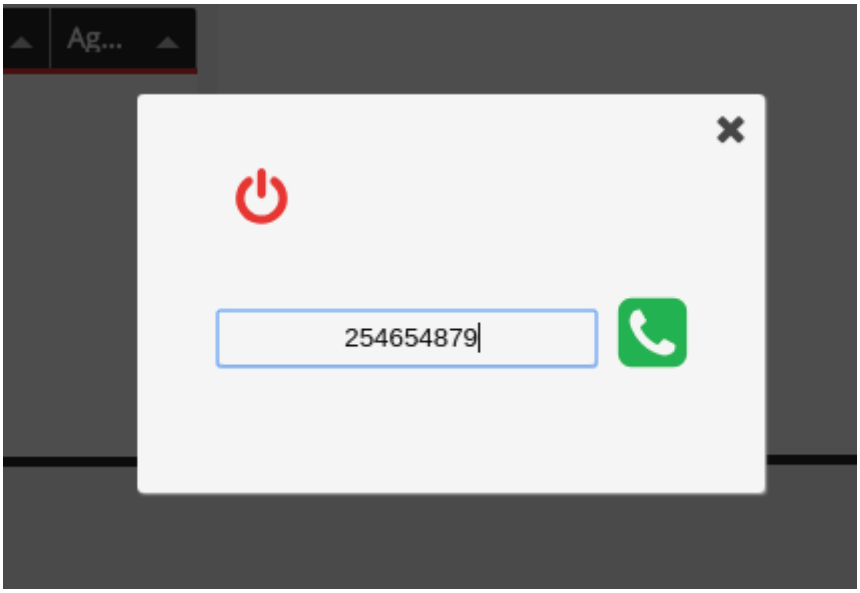
Using the Wallboard Softphone

Once these steps are completed, we can head to the wallboard page, and we should be able to see two new icons on the top bar:

The **cogs** icon opens the **Softphone Settings panel**, where the user can change its own settings that will be saved as **User preferences** for that user, and will persist between sessions.



The **phone** icon opens the actual **Softphone panel**, which can be used the same way as the **Agent Page Softphone**.


Ag...
254654879

QueueMetrics running with UTF-8 charset

QueueMetrics, since version 14.10.5 build 937 is compliant with UTF-8 charset meaning that is possible to use UTF-8 characters when defining agents, QA forms, notes and, more in general, all items in the report.

For new QueueMetrics intalls, where Tomcat and the database are installed together with rpm, there is nothing to do except to use the proper qloader on the Asterisk server (since 14.10.5 build 937 the qloader and the wqloader packages are named with build number aligned with QueueMetrics). For already installed QueueMetrics, instead, a set of steps needs to be performed in order to generate the correct environment.

Below is a list of these steps.

- Change the default database charset

```
ALTER DATABASE `queuemetrics` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

- For each table in the database, change the default charset by issuing

```
ALTER TABLE table CONVERT TO CHARACTER SET utf8 COLLATE utf8_general_ci;
```

where *table* should be replaced with the name of the table you're working on. Repeat this step for all tables present in the QueueMetrics database.

qm_tasks table ships with a key that needs to be limited before table conversion. If you receive an error when converting this table, please issue the following step.

```
ALTER TABLE `qm_tasks` DROP INDEX `pFamily` ,
ADD INDEX `pFamily` ( `pFamily` ( 20 ) , `pID` ( 20 ) );
ALTER TABLE qm_tasks CONVERT TO CHARACTER SET utf8 COLLATE utf8_general_ci;
```

- Update your Tomcat with the version you can find on QueueMetrics website. Tomcat 6.0.43-23 rpms is the minimum requirement able to properly handle UTF-8. For yum based installs, from a bash on QueueMetrics server, issue the following command:

```
yum update queuemetrics-tomcat
```

- Update your QueueMetrics with a version greater than 14.10.5 build 937. From yum based installs, from a bash on QueueMetrics server, issue the following command:

```
yum update queuemetrics
```

- Edit the SQL connector string used by QueueMetrics to connect to the database. Through a bash in your QueueMetrics server, locate the web.xml file (for yum based install this file is located in the /usr/local/queuemetrics/qm-current/WEB-INF folder). **Append** to the param-value key in the JDBC_URL section the following codes:

```
&amp;useUnicode=true&amp;characterEncoding=utf8&amp;characterSetResults=utf8
```

- Restart QueueMetrics
- Update the qloader with a version greater than 1.31.937. For a standard yum based install, through a bash in your asterisk box, issue the following command:

```
yum update qloaderd
```

Historical queue log data present into the database needs to be reloaded. This could be done **only** if you have the original queue_log file generated by asterisk for the whole dataset period. In order to to this: stop the qloader daemon, empty the queue_log table then start the qloader daemon. The qloader will start to push the whole dataset since the beginning of the queue_log file.

QA informations already present in the database cannot be migrated. For this reason, historical QA data cannot be guaranteed.

Running post-call satisfaction IVRs and pushing their results to QM

QueueMetrics includes a comprehensive Quality Assessment (QA) subsystem; it is usually used by specialized staff doing call reviews by listening to recordings of existing calls and scoring them on specific QA forms.

It would sometimes be useful to interview callers at the end of their interaction in order to track **perceived** quality. These interviews are usually a one- or two-question survey, in which they answer the question "Are you satisfied with our services?", where the answer is usually an IVR where they can enter a yes/no choice.

This way, you can get a general overview of perceived quality across time, and the QA team can use the results of this process to focus on cases where the caller was not satisfied.

Implementing this scenario with QueueMetrics is very simple. Here is how it goes:

- By the end of each call, when the call is almost finished and the caller has no more questions, the agents asks the caller whether they would like to answer a simple satisfaction inquiry.
- If they do, they are blind-transferred to an IVR, where a simple message is played back and they are given the choice to answer 1 to say they are happy with the interaction and 0 to say they are not.
- If they answer this question, this result is pushed to QueueMetrics and stored into a specialized QA form.
- From the QA reports in QueueMetrics, you can then get quality overviews, per queue and per agent.
- You use these results to target a more specific, internal QA review for calls or agents where problems appear to be more frequent.

Prerequisites

- A working QueueMetrics system

Implementation

Step 1: Calling queues

As calls in QueueMetrics are identified by their first unique-id, it is important that the original unique-id is stored in an inheritable variable, so that it can be accessed from other channels. This is because the original call is then bridged to the agent's channel, that in turn will transfer the call to the IVR.

```
exten => _900,1,Wait(1)
exten => _900,n,Answer
exten => _900,n,Playback(queue-welcome)
```

```
exten => _900,n,Set(CHANNEL(musicclass)=mymusic)
exten => _900,n,Set(__QUE=myqueue)
exten => _900,n,Set(__UID=${UNIQUEID})
exten => _900,n,Queue(${QUE},t,,,300)
exten => _900,n,Hangup
```

Please not how variables *QUE* and *UID* are made inheritable, and how we set the option *t* on the queue so that the agent can do an unattended transfer to the IVR by pressing "#" and then entering the IVR extension.

Step 2: Creating a QA form in QueueMetrics

First, we will have to go to QueueMetrics and create a QA item that will store the results. We will call it "SAT" and it will be of type Yes/No; here, the value 100 will be tracked as Yes and 0 will be tracked as No.

> You could use all values from 0 to 100, or have some fixed points to represent "Bad / Neutral / Good / Very Good" instead.

Now, let's create a simple QA form to store this information. A QA form in QueueMetrics cannot be edited / changed after the fact, and must be submitted all in one go.

We create a QA form called "SATISF", with only one section we call "Results", and to the section "Results" we add our item "SAT".

At this point, we need to create a remote user to upload this information. We go to the QueueMetrics User page, and create a new user we call "qasubmit", with password "passw0rd", belonging to class ROBOTS and including security key "QATRACK".

Step 3: Downloading the pushQA script

Though QueueMetrics requires a simple JSON API, most Asterisk systems are compiled without CURL support, so you cannot directly call a webservice from the dialplan.

We created a simple Bash script that is available in the project https://github.com/Loway/OpenQueueMetricsAddOns under "log-ivr-to-qaform" that is able to send a score for a single call.

After you download it, copy it to */var/lib/asterisk/agi-bin/* and make it readable and executable for Asterisk.

Then edit the first lines to enter the QM URL, the username and password, the name of your form and the name of the item you need to fill in.

```
QM=http://127.0.0.1:8080/queuemetrics
USER=qasubmit:passw0rd
FORM=SATISF
ITEM=SAT
```

The script expects as parameters:

- the Unique-id of the call that we want to grade
- the Queue that this call was processed upon
- The actual score (0-100)

If you want to use a QA form with multiple items, all the items must be set at once. See the QueueMetrics JSON API manual for more information on how to pass multiple items and add comments to a form.

This approach has the big advantage of being simple, but will create a new process for each request. This may not work well in high-load scenarios; in these cases, a FastAGI script will likely do.

Step 4: Setting up an IVR

This IVR will be used to send the IVR choice to the QA system:

```
exten => _999,1,Answer
exten => _999,n,Background(ivr_satisfaction)
exten => _999,n,Wait(10)
exten => _999,n,Hangup()

; Not happy
exten => 0,1,Playback(beep)
exten => 0,n,System(/var/lib/asterisk/agi-bin/pushQA.sh ${UID} ${QUE} 0)
exten => 0,n,Playback(thank-you)
exten => 0,n,Hangup

; Happy
exten => 1,1,Playback(beep)
exten => 1,n,System(/var/lib/asterisk/agi-bin/pushQA.sh ${UID} ${QUE} 100)
exten => 1,n,Playback(thank-you)
exten => 1,n,Hangup

; Other choice
exten => i,1,Playback(beep)
exten => i,2,Hangup()
```

Now, as soon as a call is completed, its result is stored in QueueMetrics and is immediately available for further quality inspection.

Keeping track of current client IP for VNC monitoring

A client has 100s of hotdesking agents who they want to keep an eye on. Got tightVNC installed which has a web VNC port on port 5800. It is obviously tricky updating the IP address on each new machine they log into. So this trigger does it for you.

Thanks to Felim Whiteley for contributing.

Prerequisites

- VNC server on port 5800

Required steps

Add the following trigger to the database:

```
DELIMITER //
DROP TRIGGER IF EXISTS vnc_url_update //
CREATE TRIGGER vnc_url_update AFTER INSERT ON arch_syslog
FOR EACH ROW BEGIN
IF (NEW.action = 1001 AND NEW.text1 LIKE "agent/%") THEN
UPDATE agenti_noti SET vnc_url = CONCAT("http://", NEW.text2, ":5800")
WHERE NEW.text1 = agenti_noti.nome_agente;
ELSEIF (NEW.action = 1002 AND NEW.text1 LIKE "agent/%") THEN
UPDATE agenti_noti SET vnc_url = ""
WHERE NEW.text1 = agenti_noti.nome_agente;
END IF;
END;//
DELIMITER ;
```

It will track agent log-ins and update the VNC URL automatically.

Serving QueueMetrics through a NGINX proxy

You may want to serve QueueMetrics through a NGINX front-end. This is often a win because:

- You can run multiple distinct services on the same virtual host.
- It is very easy to set up SSL and to secure the instance for the public internet.
- You can have flexible caching of static resources.
- You have a simple choke point to trace all requests going through, and can easily implement more complex configuration.
- You do not need to learn and edit Tomcat's own configuration.

It is trivial to set up a proxy; QueueMetrics requires a number of headers so that it can reliably rebuild both an "internal" URL and a public-facing URL as needed.

We will show an example where your QueueMetrics is located at `http://127.0.0.1:8080/queuemetrics/` and your server name is `qm.myserver.com`.

> Nginx will listen on ports 80, 443. If you have anything listen on these ports, you need to adapt the configuration

Prerequisites

- A working QueueMetrics instance
- Nginx version 1.12+

NGINX configuration

How your */etc/nginx/nginx.conf* file should look:

```
user nginx nginx;
worker_processes auto;

error_log /var/log/nginx/error.log info;

pid /run/nginx.pid;

events {
    worker_connections 1024;
    use epoll;
}

http {
        include /etc/nginx/mime.types;
        default_type application/octet-stream;
```

```
        log_format main
                '$remote_addr - $remote_user [$time_local] '
                '"$request" $status $bytes_sent '
                '"$http_referer" "$http_user_agent" '
                '"$gzip_ratio"';
        client_header_timeout 10m;
        client_body_timeout 10m;
        send_timeout 10m;
        connection_pool_size 256;
        client_header_buffer_size 1k;
        request_pool_size 4k;
        gzip off;
        output_buffers 1 32k;
        postpone_output 1460;
        sendfile on;
        tcp_nopush on;
        tcp_nodelay on;
        keepalive_timeout 75 20;
        ignore_invalid_headers on;
        large_client_header_buffers 4 128k;
        server_tokens off;

    include /etc/nginx/conf.d/queuemetrics_http.conf;
        include /etc/nginx/conf.d/queuemetrics_https.conf;
}
```

How your */etc/nginx/conf.d/queuemetrics_http.conf* should look:

```
server {
    listen 80;
    server_name qm.myserver.com;

    access_log /var/log/nginx/qm.myserver.com.access_log main;
    error_log /var/log/nginx/qm.myserver.com.error_log info;
    root /var/www;

    # IF YOU WANT TO FORCE SSL, UNCOMMENT BELOW
    # rewrite ^ https://$server_name$request_uri? permanent;

    # you don't want your queuemetrics to be indexed on google searches...
    add_header X-Robots-Tag "noindex";

    location /queuemetrics {
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Port $server_port;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-Webapp /queuemetrics;
        proxy_set_header X-Real-IP $remote_addr;
```

```
        proxy_set_header X-Url-Scheme $scheme;
        proxy_pass http://127.0.0.1:8080/queuemetrics;
        proxy_read_timeout 240s;
    }
}
```

How your */etc/nginx/conf.d/queuemetrics_https.conf* should look:

```
server {
    listen 443 ssl http2;
    server_name qm.myserver.com;

    access_log /var/log/nginx/qm.myserver.com.access_log main;
    error_log /var/log/nginx/qm.myserver.com.error_log info;

    ssl on;
    ssl_certificate /etc/ssl/nginx/qm.myserver.com.crt;
    ssl_certificate_key /etc/ssl/nginx/qm.myserver.com.key;
    ssl_dhparam /etc/ssl/nginx/dhparam;

    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers "EECDH+aRSA+AESGCM !EECDH+ECDSA+AESGCM !EECDH+ECDSA+SHA384
!EECDH+ECDSA+SHA256 !EECDH+aRSA+SHA384 !EECDH+aRSA+SHA256 !EDH+aRSA !aNULL !eNULL !LOW
!MD5 !EXP !PSK !SRP !DSS !RC4 !EECDH+aRSA+RC4 !3DES";
    ssl_session_cache shared:SSL:50m;
    ssl_session_timeout 5m;
    add_header Strict-Transport-Security "max-age=31536000; preload" always;

    root /var/www;

    # you don't want your queuemetrics to be indexed on google searches...
    add_header X-Robots-Tag "noindex";

    location /queuemetrics {
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Port $server_port;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-Ssl on;
        proxy_set_header X-Forwarded-Webapp /queuemetrics;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Url-Scheme $scheme;
        proxy_pass http://127.0.0.1:8080/queuemetrics;
        proxy_read_timeout 240s;
    }
}
```

The file */etc/ssl/nginx/dhparam* can be generated with:

```
openssl dhparam -dsaparam -out /etc/ssl/nginx/dhparam 4096
```

Make sure that the directory */etc/ssl/nginx* exists before issuing the command

Self-signed ssl certificate

An alternative to buying an ssl certificate or to obtaining it from Letsencrypt, is to make your own self-signed certificate. Since browsers like Google Chrome will complain about the Subject Alternative Name, we will generate a certificate in a different way than you might find over the web.

You need to create a file called *config.ssl* in whatever place you are confortable, because this is a temporary file.

How your *config.ssl* should look:

```
[req]
distinguished_name = req_distinguished_name
x509_extensions = v3_req
prompt = no

[req_distinguished_name]
C = CH
ST = Switzerland
L = Stabio
O = queuemetrics self-signed
OU = queuemetrics self-signed
CN = qm.myserver.com

[v3_req]
keyUsage = keyEncipherment, dataEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names

[alt_names]
DNS.1 = qm.myserver.com
DNS.2 = www.qm.myserver.com
```

How to issue the certificate:

```
openssl req \
        -x509 \
        -sha512 \
        -nodes \
        -days 3650 \
```

```
        -newkey rsa:4096 \
        -keyout /etc/ssl/nginx/qm.myserver.com.key \
        -out /etc/ssl/nginx/qm.myserver.com.crt \
        -config config.ssl
```

Be sure that you are issuing the command in the same directory where there is your *config.ssl* You need to restart nginx to apply the changes you've done in the configuration file.

Redirect the requests from Tomcat to Nginx

Tomcat will still listen on port 8080 and it is still reachable, so at this point you may want to redirect the requests to nginx. It is also useful if you want to maintain the compatibility between the links that each client has stored in its bookmark and the new url provided by Nginx.

The *iptables PREROUTING chain* will help you to do the job with the following rule:

```
iptables -t nat -A PREROUTING -p tcp --dport 8080 -j REDIRECT --to-port 80
```

Troubleshooting

If you go to *System diagnostic tools* then *View Configuration* and then open the menu for *HTTP Configuration*, you will be able to see:

- *C: localWebappUrl* is the "inner" address where Tomcat is serving the webapp - eg. `http://1.2.3.4:8080/queuemetrics`
- *Q: getPublicQmBaseUrl* is the "public" address that QueueMetrics is being served from, e.g. `https://qm.myserver.com/queuemetrics`
- *C: realRemoteIP* is the IP address of your client, as forwarded by the proxy
- *Header: ...* records show the HTTP headers that come with the request, both coming from your browser and the proxy.

Further developments

- If you need the webapp to work under a different assumed name, you need to rewrite cookies so that QueueMetrics can place them with the correct path
- You may want to cache static resources, so they are served directly by Nginx.
- By default, we are providing a configuration with *gzip off* because of the BREACH Vulnerability (http://breachattack.com/#howitworks). You may want to add the gzip compression at your own risk.

Self-host audio recordings with AudioVault and Nginx

Self-hosting call recordings on your QueueMetrics-Live instance can save money, and make management easier. In this guide, we'll show you how to use AudioVault and Nginx as a secure proxy on ports 4005 and 4006, allowing your QueueMetrics-Live instance or remote QueueMetrics server to play back recordings directly from reports for a low cost SSL certificate.

> In this guide we are going to focus on FreePBX version 16; as well as using two different ports, 4005 and 4006, since 80 and 443 are usually reserved by FreePBX.

Prerequisites

- a hosted FreePBX instance;
- a domain name for the FreePBX server (in this guide, "audiovault.mydomain.com");
- a valid SSL certificate for the domain (not self-signed nor Let's Encrypt);
- a QueueMetrics-Live instance;
- port 4006 opened in your firewall;
- recordings divided in sub-folders by year/month/day.

Setting up the PBX server

On the PBX server, we simply need to make sure that the recordings are present (usually in the `/var/spool/asterisk/monitor/` folder). Make sure the Uniloader service is installed (at least version 21.4).

You can check the Uniloader version with:

```
uniloader -v
```

If Uniloader is not installed, use the following commands:

```
wget https://yum.loway.ch/loway.repo -O /etc/yum.repos.d/loway.repo
yum install uniloader
```

Creating the AudioVault service

Create and edit the "uniloader-audiovault" file with:

```
nano /etc/init.d/uniloader-audiovault
```

With:

```
#!/bin/bash
#
# Startup script for Uniloader AudioVault.

# Source function library.
. /etc/rc.d/init.d/functions

pidfile=${PIDFILE-/var/run/uniloader-audiovault.pid}
logfile=${LOGFILE-/var/log/asterisk/uniloader-audiovault.log}
lockfile=${LOCKFILE-/var/lock/subsys/uniloader-audiovault}

prog=$0
RETVAL=0

start() {
    echo -n $"Starting AudioVault:"
    uniloader av serve \
    --path file:/var/spool/asterisk/monitor/%YY/%MM/%DD/ \
    --token MYTOKEN \
    --public-url https://audiovault.mydomain.com:4006 \
    --bind-to localhost:4005 \
    --pid $pidfile \
    >>$logfile 2>&1 &
    echo_success
    echo
    touch ${lockfile}
    return 1
}

stop() {
    echo -n $"Stopping AudioVault:"
    kill $(cat ${pidfile}) 2> /var/null
    sleep 1
    RETVAL=$?
#    echo_success
#    echo
    if [ $RETVAL = 0 ]
    then
  rm -f ${lockfile} ${pidfile}
    fi
    return $RETVAL
}

status() {
  if [ ! -f ${pidfile} ]
  then
    echo "$prog is not running"
    RETVAL=3
```

```
  else
    echo "$prog is running..."
    RETVAL=0
  fi
  return $RETVAL
}
# See how we were called.
case "$1" in
  start)
        start
    ;;
  stop)
        stop
    ;;
  restart)
    stop
    start
    ;;
  status)
  status
  ;;
  force-reload)
    stop
    start
    ;;
  *)
  echo $"Usage: $prog {start|stop|restart|status|force-reload}"
  exit 1
esac

exit $RETVAL
```

You will need to edit the following variable with:

- `path file:....` → the path to the recordings
- `token` → the token, basically the password that you chose, to access the recordings
- `public-url` → the domain name, where the recordings will be found, as seen from QueueMetrics Live
- `:4006` → we are going to use this port, you can change it if you need to
- `bind-to localhost` → in this case we are going to be using port 4005, you can change it if you need to

Add permissions using:

```
chmod 757 /etc/init.d/uniloader-audiovault
```

Reload the Unit changed on disk:

```
systemctl daemon-reload
```

Enable the uniloader-audiovault service with:

```
systemctl enable uniloader-audiovault
```

And finally, start the AudioVault service with:

```
service uniloader-audiovault start
```

Setting up the Nginx service

Make sure Nginx is installed on your server (at least version 1.16).

Edit the Nginx configuration file in this folder:

```
nano /etc/nginx/nginx.conf
```

Like this:

```
user nginx nginx;
worker_processes auto;

error_log /var/log/nginx/error.log info;

pid /run/nginx.pid;

events {
    worker_connections 1024;
    use epoll;
}

http {
        include /etc/nginx/mime.types;
        default_type application/octet-stream;
        log_format main
                '$remote_addr - $remote_user [$time_local] '
                '"$request" $status $bytes_sent '
                '"$http_referer" "$http_user_agent" '
                '"$gzip_ratio"';
        client_header_timeout 10m;
        client_body_timeout 10m;
        send_timeout 10m;
        connection_pool_size 256;
        client_header_buffer_size 1k;
```

```
        request_pool_size 4k;
        gzip off;
        output_buffers 1 32k;
        postpone_output 1460;
        sendfile on;
        tcp_nopush on;
        tcp_nodelay on;
        keepalive_timeout 75 20;
        ignore_invalid_headers on;
        large_client_header_buffers 4 128k;
        server_tokens off;

        include /etc/nginx/conf.d/audiovault_https.conf;
}
```

Create a new “audiovault_https.conf” configuration file here:

```
/etc/nginx/conf.d/audiovault_https.conf
```

Like this:

```
server {
        listen 4006 ssl http2;
        server_name audiovault.mydomain.com;

        access_log /var/log/nginx/audiovault.mydomain.com.access_log main;
        error_log /var/log/nginx/audiovault.mydomain.com.error_log info;

        ssl on;
        ssl_certificate /etc/ssl/nginx/audiovault_mydomain_com.crt;
        ssl_certificate_key /etc/ssl/nginx/audiovault_mydomain_com.key;
        ssl_dhparam /etc/ssl/nginx/dhparam;

        ssl_protocols TLSv1.2;
        ssl_prefer_server_ciphers on;
        ssl_ciphers "EECDH+aRSA+AESGCM !EECDH+ECDSA+AESGCM !EECDH+ECDSA+SHA384
!EECDH+ECDSA+SHA256 !EECDH+aRSA+SHA384 !EECDH+aRSA+SHA256 !EDH+aRSA !aNULL !eNULL !LOW
!MD5 !EXP !PSK !SRP !DSS !RC4 !EECDH+aRSA+RC4 !3DES";
        ssl_session_cache shared:SSL:50m;
        ssl_session_timeout 5m;
        add_header Strict-Transport-Security "max-age=31536000; preload" always;

        root /var/www;


        location / {
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header X-Forwarded-Host $host;
                proxy_set_header X-Forwarded-Port $server_port;
```

```
                proxy_set_header X-Forwarded-Proto $scheme;
                proxy_set_header X-Forwarded-Server $host;
                proxy_set_header X-Forwarded-Ssl on;
                proxy_set_header X-Forwarded-Webapp /audiovault;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Url-Scheme $scheme;
                proxy_pass http://127.0.0.1:4005;
                proxy_read_timeout 240s;
        }
}
```

Create the following folder:

```
mkdir -p /etc/ssl/nginx
```

Generate the “dhparam” file with the following command:

```
openssl dhparam -dsaparam -out /etc/ssl/nginx/dhparam 4096
```

Don’t forget to copy your .crt and .key files in the /etc/ssl/nginx/ folder.

Make sure to enable the Nginx service with:

```
systemctl enable nginx
```

And restart the service with:

```
service nginx restart
```

Opening port 4006 in FreePBX

In order for QueueMetrics to be able to see the call recordings, we will need to make sure Port 4006 is open, and accessible from outside your network.

If you are using FreePBX it would be enough to create a new Custom Service:

`FreePBX Homepage → Connectivity → Firewall → Services (menu on the right) → Custom Services → Create new service:`

```
Description: AudioVault
Protocol: Both
Port Range: 4006
```

Save and apply the changes.

Setting up the QueueMetrics-Live instance

Now, the only thing to do, is to make sure QueueMetrics knows where to look for the call recordings.

Go to your QueueMetrics-Live instance and add the following parameters at the end of the file:

`Homepage → Settings → Edit system parameters`

```
# AudioVault configuration
audio.server=it.loway.app.queuemetrics.callListen.listeners.JsonListener
audio.jsonlistener.url=https://audiovault.mydomain.com:4006/search/
audio.jsonlistener.method=POST
audio.jsonlistener.searchtoken=MYTOKEN
audio.jsonlistener.verbose=false
audio.html5player=true
```

- `audio.jsonlistener.url` → is the URL to the AudioVault recordings and the port you are using, followed by `/search/`
- `audio.jsonlistener.searchtoken` → is the token that we chose previously, when creating the AudioVault service

Log out and then back in QueueMetrics, for the changes to take effect.

You can now open the call details of a call, and find the audio recordings in the QA tab.

Troubleshooting

Call recordings are not visible

- Make sure that the call recordings are accessible in the /var/spool/asterisk/monitor/ and the name of the audio files include the Asterisk Unique ID of the call.
- Verify that your SSL certificate is installed correctly on your server, using a service like geocertsssl ssl-checker.
- Make sure port 4006 is open in your firewall, as stated above.

The report page is slow to load the recording

The recordings need to be divided in sub-folders by year, month and day, for better performance. If you need to change the settings on your FreePBX, you can have a look at this guide.

Understanding MySQL storage and clustering

QueueMetrics works by parsing an ACD event file called *queue_log* that Asterisk produces during its normal operations. QueueMetrics is built to run with two different storage models:

- *The Flat File storage model* lets the data reside in a file on the Asterisk server, and is the default mode;
- *The MySQL storage model* uploads the data to a database using a small loader script called Qloaderd and has QueueMetrics read it back from that database.

Running through a flat-file storage is usually enough for small call centers having both Asterisk and QueueMetrics living on the same server. This is okay for small setups (like 10-20 agents) where the size of the *queue_log* file will hardly ever reach the tens of megabytes.

On the contrary, if you run a larger setup, the advantages of the MySQL storage model become more important and worth the extra setup:

- You can install QueueMetrics on a separate server from the main Asterisk server. This means that as your Asterisk server grows busier, you don't want to risk that someone running a monster analysis through QueueMetrics may slow it down.
- You get a lot of added flexibility: you can have MySQL, Asterisk and QueueMetrics live on different servers, and choose the best deployment model according to your actual usage load. You can also create hot-backup scenarios with a secondary QueueMetrics server that will take over in case the primary server should be down, and you can take full advantage of the database's replication model to handle high load and high-availability requirements.
- You get better efficiency: MySQL is optimized to maximize disk access efficiency when dealing with a subset of the whole data (e.g. getting today's information for just one queue will not need scanning your whole ACD history)
- You can have multiple Asterisk servers sending data to the same QueueMetrics instance and you can use that one instance to report on all activity through all servers: we call this *clustering*. This makes it possible to build very large call centers in a simple way, just by adding more inexpensive hardware to handle more calls and by putting yourself in a situation where a problem on a server will not take down the whole call center but just a fraction of it.

Enabling MySQL storage

In order to enable MySQL storage, you have to:

- Install Qloaderd on your Asterisk server and point it to the main QueueMetrics database
- Tell QueueMetrics to run reports from the database

Understanding MySQL storage

In order to set up the system, we first have to understand the concepts involved.

- The QueueMetrics **database** is a MySQL database loaded with the QueueMetrics default database structure. This is usually installed with QueueMetrics (as it is needed to make it work) and can be on any machine, as QueueMetrics will connect to it via a TCP/IP socket. When we refer to the database in the context of MySQL storage, we are usually thinking about a table called *queue_log*; that's where Asterisk log information is uploaded to.
- The database is logically divided into one or more **partitions**; these are virtual tables that reside within one single physical table. Partitions are useful as they let you have more than one Asterisk queue data set within the same database. This is useful e.g. for testing, or for sending multiple *queue_log* data sets to the same physical database for clusters. You can call a partition any name you like, as long as it does not exceed 5 characters.
- Qloaderd is a small Perl script that will upload the data Asterisk produces to one chosen partition. It is built to be lightweight and pretty smart; in case it is restarted, it will automatically detect which data is already present in the database and upload only missing data. It runs on the Asterisk server and points to the MySQL server that QueueMetrics uses; in order to run it, you must decide a partition for it to upload data to. If multiple copies of Qloaderd are run at once, each copy must point to its own partition, or data corruption will surely happen.
- QueueMetrics will access MySQL data by using a special file name, that is usually "sql:partition". In order to see what is going on with your database, QueueMetrics offers a special MySQL storage inspection mode to see which partitions are available, which are active and how much data is in them.

Installing Qloaderd

First you have to install Qloaderd and then you'll have to configure it to suit your needs.

Installing on TrixBox or other CentOS based Linux distros

If you run TrixBox or any other CentOS/RHEL base Linux distro, you can install Qloaderd very easily using the provided *yum* package manager:

```
wget https://yum.loway.ch/loway.repo -O /etc/yum.repos.d/loway.repo
yum install qloaderd
```

If in the future you would like to upgrade it, you will do so by running:

```
yum update qloaderd
```

The installation process installs all necessary components and dependencies, Qloaderd itself and starts it immediately. Qloaderd is installed in */usr/local/qloader* and its startup script is installed as */etc/init.d/qloaderd*. If you just installed Qloaderd using *yum*, skip to the section called Customizing Qloaderd

Manual Qloaderd installation

In order to download the Qloaderd package, go to the Downloads page on QueueMetrics' website.

Make sure you have the following packages available on your system, as Qloaderd will use Perl's DBI to connect to MySQL:

```
libdbi
libdbi-dbd-mysql
libdbi-drivers
```

Unpack the Qloaderd tar-ball and 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:

```
cd /usr/local/qloader
dos2unix qloader.pl
chmod a+x qloader.pl
```

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 should very likely copy it from *Other-initscript* to */etc/init.d* on your system.

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

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

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

```
chkconfig --add qloaderd
```

(or the equivalent for your distribution).

Customizing Qloaderd

Before using Qloaderd, you must define:

- The name and address of the MySQL database server
- On which partition is it supposed to upload data to
- The Asterisk log file it must upload
- Its own error log file

To set the name and address of the MySQL server, edit the file */usr/local/qloader/qloader.pl* and change the following lines as needed:

```
my $mysql_host = "10.10.3.5";
my $mysql_db   = "queuemetrics";
my $mysql_user = "queuemetrics";
my $mysql_pass = "javadude";
```

Please make sure that the MySQL server will allow connecting from the Asterisk server - this can usually be obtained by running a SQL statement on the database server, such as:

```
grant all privileges on queuemetrics.* to 'queuemetrics'@'10.10.3.100'
           identified by 'javadude';
```

where the string "10.10.3.100" in the example is the internal IP address of the Asterisk box we're installing Qloaderd on.

To set the other parameters, edit the file */etc/sysconfig/qloaderd* and change the following parameters:

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

If you are unsure what to set the partition to, just leave it to P001 for the moment. The *queuelog* parameter is the name of the file Asterisk produces and it is to be uploaded. The *logfile* parameter is a log file where Qloaderd will write its activity to and that you can look up to debug problems.

Starting and stopping Qloaderd

To test if everything is in order, you can now start Qloaderd by running:

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

It should start up and start writing to its own log file. To see if everything is okay, you just run:

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

And it should output something like:

```
 |Fri Sep 14 09:33:24 2007|QueueMetrics MySQL loader - $Revision: 1.3 $
 |Fri Sep 14 09:33:24 2007|Partition P001 - PID 2827 - TZ offset: 0 s.
        - Heartbeat after 900 s.
 |Fri Sep 14 09:33:24 2007|Now connecting to DB qm14 on 10.10.3.5 as
        user queuemetrics with password queuemetrics
 |Fri Sep 14 09:33:24 2007|Ignoring all timestamps below 0
```

As you can see, it states what it is trying to do and will start uploading data. Every 100 lines of uploaded data or 900 seconds of Asterisk ACD inactivity, it will output a reference line that tells how much data it has uploaded in the current usage session.

If you see something like:

```
 |Fri Sep 14 09:25:49 2007|QueueMetrics MySQL loader - $Revision: 1.3 $
 |Fri Sep 14 09:25:49 2007|Partition P001 - PID 2749 - TZ offset: 0 s.
        - Heartbeat after 900 s.
 |Fri Sep 14 09:25:49 2007|Now connecting to DB log_code on 10.10.3.5 as
        user ldap with password ldappo
E|Fri Sep 14 09:25:49 2007|---ERROR FOUND--
E|Fri Sep 14 09:25:49 2007|Error type: dr
E|Fri Sep 14 09:25:49 2007| Statement:
E|Fri Sep 14 09:25:49 2007|     Error: Unknown database 'log_code'
E|Fri Sep 14 09:25:49 2007|Waiting 15s before reattempting to connect
```

This means that it was not possible to connect to the database - very likely your server, database, user, password or access grants are wrong. Please note that in case of an error, it will simply try again - this way, no matter what happens to your database, Qloaderd will try again and again until it can establish a working connection and will then upload data.

You can stop or restart Qloaderd using the following commands:

```
/etc/init.d/qloaderd stop

/etc/init.d/qloaderd restart
```

Testing if data is actually being uploaded

To make sure that data is being uploaded correctly, you should log on to the database server, open the MySQL shell and issue a command like:

```
select partition, queue, count(*) as n_records
```

```
from queue_log
group by partition, queue
order by partition, queue
```

The result should look something like this:

```
+-----------+------------------+-----------+
| partition | queue            | n_records |
+-----------+------------------+-----------+
| P003      | myqueue          |         9 |
| P003      | NONE             |       121 |
| P003      | queue-abc        |      2096 |
| P003      | queue-test       |      1341 |
| P003      | UNK              |        17 |
| P01       | qq-group         |     33000 |
| P01       | cust-rajax       |       204 |
| P01       | NONE             |      8139 |
| RT        | NONE             |      8064 |
| RT        | q1               |      9216 |
| RT        | q2               |      9216 |
+-----------+------------------+-----------+
11 rows in set (0.16 sec)
```

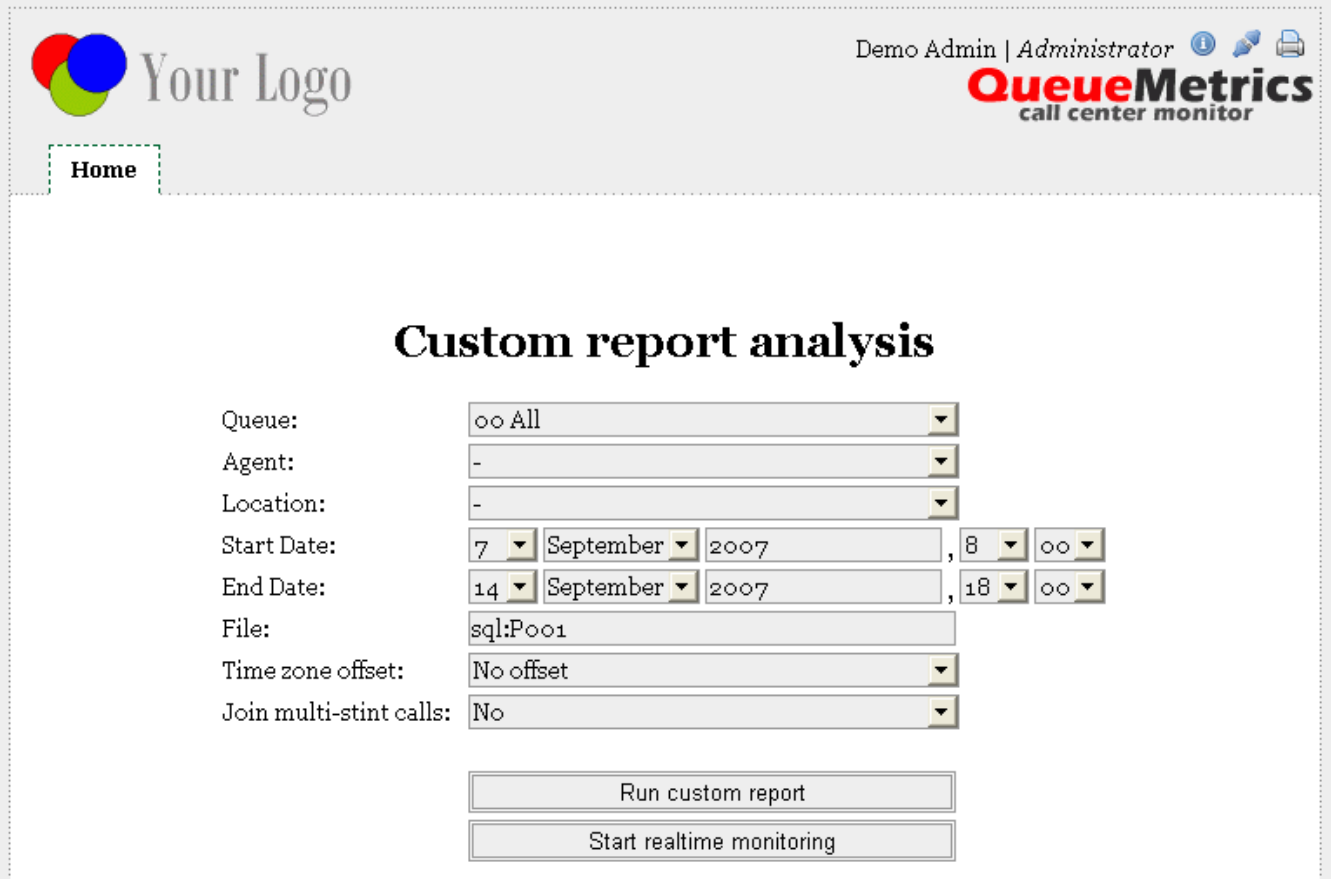
This report shows:

- That we are using three distinct partitions: P003, P01 and RT. (At first you will only find one).
- For each partition, we see the Asterisk queue names involved plus the special keyword NONE
- For each queue, we get an idea of how many records it generated, i.e. how big it was. As a rough estimate, consider that each call generates an average of around three or four records (but this is dependent on how your call center is set up).

As your Asterisk system runs and data is uploaded into the database, you should see the figures for the number of records rise between repetitions of the same query. If the figures don't change, or you do not see you partition at all, this means that Qloaderd is not uploading data.

Setting up QueueMetrics

Setting up QueueMetrics is very straightforward: if you click on "Run custom reports" and enter "sql:P001" as the filename, you can check if your partition P001 contains data for the queue you just selected.



Just remember to configure the queues you want to report on, before running this test.

Changing the defaults

Of course, you'll want QueueMetrics to automatically use the partition you choose as its default data source - you can easily do this by editing the file configuration.properties and setting:

```
# This is the default queue log file.
default.queue_log_file=sql:P001
```

After restarting QueueMetrics, the MySQL storage source will be used as a default.

Using the QueueMetrics database inspector

As it is not always easy to understand how much data is in the database, QueueMetrics offers a database inspection tool to easily see which data is available.

You can enter it by clicking on the "Mysql storage information" link from the "Edit QueueMetrics settings" section (if you do not find that link, make sure your admin user holds the key USR_MYSQL).

If it takes a while to enter the page, this is perfectly normal; in order to produce the data seen through the inspector, it performs a series of table scans. As adding the correct indexes for faster analysis would use a lot of disk space and slow down the database system considerably versus an occasional use of the inspector, they were not included with QueueMetrics.

Your Logo

Demo Admin | *Administrator*

QueueMetrics
call center monitor

Home

Current storage info

Total number of rows in table: 151.253
Total table space: 15,9 M (Data: 11,2 M - Indexes: 4,7 M)

Partition	Entries	N. calls	From:	To:	Days of data:	Last heartbeat:	
[illegible]	14.024	646	2007-05-29 05:04	2007-08-21 08:00	84,0 days	2007-08-21 08:15	...
[illegible]	52.518	9.349	2007-06-28 06:00	2007-07-24 07:01	26,5 days	2007-07-29 07:32	...
P004	48	8	2007-08-19 08:25	2007-09-14 09:56	25,8 days	2007-09-14 09:56	...
Pa	53.541	326	2006-01-17 01:36	2007-08-23 08:43	583,0 days	2007-08-23 08:43	...
[illegible]	18	6	2007-08-30 08:20	2007-08-30 08:21	0,1 days	2007-08-30 08:21	...
rt	31.104	4.608	2007-06-21 06:26	2007-06-23 06:25	2,0 days	2007-06-23 06:25	...

Note: the "Number of Calls" shown here is meant as a rough estimate and may differ from the one actually shown by the reports.

From this page we see which partitions are available for analysis. For each partition we can see:

- The total number of entries available
- An *estimated* number of calls
- The date of the oldest and most recent call entry
- How many days does the data span over
- The last heartbeat: if there is no data to upload (e.g. at night), the Qloaderd will in any case upload a Heartbeat record every 15 minutes. This makes it clear that, even if there is no recent data in it, a working Qloaderd is currently uploading data on that partition. If there is no new data and the last heartbeat is over 20 minutes old, this probably means that the Qloaderd process is not running.

If we click on the details of a partition, we see a screen like the following one:

Details for partition: rt

N. rows in partition: 31.104

Queues:

Queue	Entries	From:	To:	Days:
q1	9.216	2007-06-21 06:26	2007-06-23 06:25	2,0 days
q2	9.216	2007-06-21 06:26	2007-06-23 06:25	2,0 days

Agents:

Agent	Entries	From:	To:	Days:
Agent/101	5.184	2007-06-21 06:26	2007-06-23 06:25	2,0 days
Agent/102	6.912	2007-06-21 06:26	2007-06-23 06:25	2,0 days
Agent/103	6.336	2007-06-21 06:26	2007-06-23 06:25	2,0 days
Agent/104	6.912	2007-06-21 06:26	2007-06-23 06:25	2,0 days

This shows which queues and agents are available on the partition, and also shows the oldest and most recent records regarding each queue or each agent.

Running QueueMetrics to monitor a cluster of Asterisk servers

By using the Qloaderd mechanism, you can have a number of distinct Asterisk servers, each uploading data to a different partition of the same QueueMetrics database. This allows to run an analysis that spans the whole cluster, and not just one single Asterisk server.

To set up a cluster of Asterisk servers to be monitored through QueueMetrics, you should adhere to the following deployment rules:

- Agent names must be unique throughout the call center; that is, you cannot have an *Agent/101* working on Asterisk server A and another *Agent/101* working on Asterisk server B, as their data would be mixed and it would be quite hard to tell who did what.
- Queue names may be shared over more than one server, and QueueMetrics will report on that queue as if it had been processed on a single box.
- As the real-time monitoring is per-queue, you can run real-time monitoring on a queue by queue basis, no matter on which physical servers those queues are hosted on, or if they are hosted on more than one server.

This deployment permits call center scalability by adding more Asterisk servers, as needed, to handle the traffic.

> in order to run a clustered version of QueueMetrics, you need to use a special cluster licence key. If you need to test this feature before purchasing, you should ask for a cluster temporary key, as the ordinary temporary keys do not natively support cluster mode.

Installing Qloaderd on the Asterisk servers

To get started, you must install Qloaderd on each Asterisk server and make sure that it is uploading data to the main QueueMetrics database. Each server must be uploading data to a distinct partition, e.g. if you have three servers, you could upload to partitions named "S1", "S2" and "S3". In short, each instance of Qloaderd is configured exactly as if it were in the single-server MySQL storage model.

Setting up QueueMetrics

Setting QueueMetrics to work with clustering requires a little effort, as we must provide it with all the necessary information required to contact and distinguish each Asterisk server and perform its duties, e.g. logging on agents. We have to edit the *configuration.properties* file and define the machines that make up our cluster by giving each a short name. This name might be the same as the partition name or might be different - as you best see fit. In the following example, we will setup a cluster of two machines - one called "aleph" with its data uploaded on partition "P001" and one called "trix" with its data uploaded to partition "P002".

```
# Cluster to poll for RT data
# list of pipe-separated members of the cluster
cluster.servers=aleph|trix
```

This sets up a cluster made up of two Asterisk servers, called "aleph" and "trix". For each box, we then create an entry like the following one for "aleph":

```
cluster.aleph.manager=tcp:dial:12345@10.10.3.5
cluster.aleph.queuelog=sql:P001
cluster.aleph.monitored_calls=z:/qm_streamcall/server_aleph
cluster.aleph.callfilesdir=
cluster.aleph.audioRpcServer=
cluster.aleph.agentSecurityKey=
```

And we will do the same for the other server "trix".

```
cluster.trix.manager=tcp:admin:amp111@127.0.0.1
cluster.trix.queuelog=sql:P002
cluster.trix.monitored_calls=z:/qm_streamcall/server_trix
cluster.trix.callfilesdir=
cluster.trix.audioRpcServer=
cluster.trix.agentSecurityKey=
```

The explanation of the entries is:

- The *manager* entry refers to the Asterisk manager API interface for that machine, and is written as *tcp:[login]:[password]@[server]*. This is needed for Live monitoring and to login/logoff/pause agents. Make sure there is a matching entry in the file /etc/asterisk/manager.conf on each server!
- The queuelog file refers to which partition the data for this server will be pulled from. The format is the same as the sql:partition format we previously encountered.
- The monitored_calls entry points to a disk share where call recordings for this machine are found. This allows to listen to calls through the QueueMetrics browser interface.
- The other entries should be left blank.

Now change the following property:

```
# This is the default queue log file.
default.queue_log_file=cluster:*
```

And restart QueueMetrics. Now all queries are running on the whole cluster, and the real-time monitoring is running on the whole cluster as well.

For more detail on clustering in QueueMetrics, see the section *"Monitoring clusters with*

QueueMetrics" in the QueueMetrics user manual.

Tips and tricks for Qloaderd

Checking how much data is in the database on a daily basis

If you want a breakdown by day of the contents of the each partition, you can run the following query:

```
SELECT partition,
  FROM_UNIXTIME(time_id, '%Y%m%d' ) as date,
       count(*) as n_rows
  FROM queue_log
 GROUP BY partition, FROM_UNIXTIME( time_id, '%Y%m%d' )
 ORDER BY partition, FROM_UNIXTIME( time_id, '%Y%m%d' );
```

The result will look something like:

```
+-----------+----------+--------+
| partition | date     | n_rows |
+-----------+----------+--------+
| P01       | 20070329 |   4216 |
| P01       | 20070411 |      5 |
| P01       | 20070412 |      3 |
| rt        | 20070508 |   9365 |
| rt        | 20070509 |  13248 |
| rt        | 20070510 |   3883 |
+-----------+----------+--------+
6 rows in set (0.45 sec)
```

And will tell you how many rows were uploaded per partition, per day.

Optimizing queue_log access time

If you would like QueueMetrics to access the database faster, you can run the following query to reorganize data on disk:

```
ALTER TABLE `queue_log`
ORDER BY partition, time_id, unique_row_count
```

This query might take a very long time to complete if the database is big and will lock the table until completion, so it should be run in a moment when the system is idle (e.g. at night via a cron job).

Getting help

If you are still experiencing problems installing or running QueueMetrics, we suggest you check out the following resources:

- The QueueMetrics FAQs at https://www.queuemetrics.com/faq.jsp are a collection of common solved problems that people have previously experienced with QueueMetrics. If you are faced with an error message, this is the first place to look at.
- The official QueueMetrics User Manual can be read online at https://www.queuemetrics.com/manual_list.jsp and provides very detailed documentation on all aspects of QueueMetrics.
- The QueueMetrics forums at http://forum.queuemetrics.com will help you in pinpointing your problems and getting community support. They are also helpful in seeing what other people are doing with QueueMetrics.
- AstRecipes is a wiki collection of Asterisk "recipes", mainly aimed at call-center users see http://astrecipes.net
- If your issues are still unsolved, you may contact Loway - refer to https://www.queuemetrics.com/contact.jsp for all relevant contact information.