

---

# QueueMetrics - Advanced Configuration Manual

Loway

Revision \$Revision: 1.5 \$

Revision History  
\$Date: 2011/05/17 10:21:15 \$

L

## Table of Contents

Acknowledgements .....	2
ViciDial integration .....	2
Prerequisites .....	2
Changes to QueueMetrics database .....	2
Changes to ViciDial .....	3
Changes to QueueMetrics .....	4
Serving QueueMetrics through Apache .....	4
Prerequisites .....	4
Installing mod_jk .....	4
Configuration of Apache and mod_jk .....	5
Virtual host configuration .....	5
Creating Virtualhosts in Tomcat .....	6
Troubleshooting .....	6
Serving QueueMetrics over Apache/SSL .....	6
Prerequisites .....	7
Configure the Name-Based SSL Virtual Hosts .....	7
Install the Apache HTTP Server and Its SSL/TLS Module .....	7
Configure the global Apache Settings .....	7
Configure the global SSL/TLS Settings .....	7
Create DNS records .....	8
Create the Application Directories .....	8
Install the CRT, CSR, and KEY files .....	8
Configure the Virtual Hosts .....	8
Verify the Configuration .....	9
Setup the service for automated startup .....	10
Troubleshooting .....	10
Summary of Log Files used by Apache .....	10
Notes .....	10
Installing QM as a ROOT webapp .....	11
Prerequisites .....	11
Changes to Tomcat .....	11
Changes to QueueMetrics .....	11
Enabling GZIP compression in Tomcat .....	11
Prerequisites .....	11
Changes to Tomcat .....	12
Changes to QueueMetrics .....	12
For further reading .....	12
Advanced QueueMetrics monitoring .....	12
Prerequisites .....	12
Assessing memory problems .....	12
Remote monitoring with VisualVM .....	13
Database connection pooling .....	14
Prerequisites .....	14
Changes to the MySQL server .....	14
Changes to QueueMetrics .....	14
Changes to Tomcat .....	14
Monitoring and fixing "slow queries" in MySQL .....	16
Prerequisites .....	17
Changes to MySQL server .....	17
Changes to QueueMetrics .....	17
Moving the queue_log table to InnoDB .....	17
Prerequisites .....	17
Changes to MySQL .....	17
Changes to QueueMetrics .....	17
Using Master-master database replication for strong high-availability .....	17
Prerequisites .....	18

Changes to MySQL .....	18
Changes to QueueMetrics .....	20
Native MySQL logging of queue_log data .....	20
Prerequisites .....	20
Separating audio recordings in a daily folder .....	21
Prerequisites .....	21
Archiving script .....	21
Changes to QueueMetrics .....	22
Making older files accessible .....	22
Show ringing phones in the realtime page .....	22

## Acknowledgements

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

- Matt Florell from the ViciDial Group, USA <matff@vicial.com [mailto:matff@vicial.com]>
- Matthew J. Roth of InterMedia Marketing, USA
- Jens von Bulow of Xantech, South Africa
- Rahul Rajan of M.H.Alshaya Co

## 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 frontends. It is capable of inbound, outbound, and blended call handling.

It can also be easily integrated with QueueMetrics.

For more information, see <http://www.vicial.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 them for QueueMetrics or ViciDial or general Asterisk usage) be on the same time zone and time, aligned with a sub-second precision by a 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 vicial@'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 be easily set from the ViciDial Admin # System Settings page:

The screenshot shows the ViciDial Admin interface. The left sidebar contains navigation links for Administration, Users, Campaigns, Lists, Scripts, Filters, In-Groups, User Groups, Remote Agents, Admin, Call Times, Shifts, Phones, Templates, Carriers, Servers, Conferences, System Settings (selected), System Statuses, Voicemail, Audio Store, Music On Hold, Text To Speech, and Reports. The main content area is titled 'MODIFY VICIDIAL SYSTEM SETTINGS' and lists various system parameters. The QueueMetrics logging section is highlighted in green.

Parameter	Value
Version	2.2.0b0.5
DB Schema Version	1191
DB Schema Update Date	2010-01-07 13:34:20
Auto User-add Value	101
Install Date	2009-12-23
Use Non-Latin	0
Webroot Writable	1
VICIDIAL Agent Disable Display	ALL
Allow SIPSAK Messages	0
Admin Home URL	./vicedial/welcome.php
Enable Agent Transfer Logfile	0
Timeclock End Of Day	0000
Timeclock Last Auto Logout	2010-01-08
Agent Screen Header Date Format	MS_DASH_24HR 2008-06-24 23:59:59
Agent Screen Customer Date Format	AL_TEXT_AMPM OCT 24 2008 11:59:59 PM
Agent Screen Customer Phone Format	US_PARN (000)000-0000
Agent API Active	1
Agent Only Callback Campaign Lock	1
Central Sound Control Active	1
Sounds Web Server	192.168.198.112
Sounds Web Directory	p1zx7dcpmta2yc85vpjzwp50rvzlh
Active Voicemail Server	192.168.198.112
Auto Dial Limit	4
Outbound Auto-Dial Active	1
Max FILL Calls per Second	40
Allow Custom Dialplan Entries	1
User Territories Active	1
Enable Second Webform	1
Enable TTS Integration	1
QC Features Active	0
QC Last Pull Time	2009-12-23 11:30:53
Enable QueueMetrics Logging	0
QueueMetrics Server IP	
QueueMetrics DB Name	
QueueMetrics DB Login	
QueueMetrics DB Password	
QueueMetrics URL	
QueueMetrics Log ID	VIC
QueueMetrics EnterQueue Prepend	NONE

- 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 "vicedial"
- QueueMetrics DB password: the database password, in our example "qm"
- QueueMetrics URL: the login UR for QM, e.g. "http://1.2.3.4:8080/queuemetrics"
- QueueMetrics LogID: leave it to VIC (this is an ID for the server)
- QueueMetrics EnterQueue Prepend: This field is used to allow for prepending of one of the vicedial\_list data fields in front of the phone number of the customer for customized QueueMetrics reports. Default is NONE to not populate 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 vicedial_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.ClassicXmlRpcListenerRT
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 create freely 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

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

- You need 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 be defined; default values are '/usr/local/queuemetrics/tomcat' and '/usr/local/queuemetrics/java' respectively.

\todo \*\* CHECK paths.

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/httpd/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

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
    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
    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.

## 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

```

```
\todo ** WHERE IS THE FILE?
```

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

```
request.registerRequests=false
```

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

## Serving QueueMetrics over Apache/SSL

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 of 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 virtual 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.ke
    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_
</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.
               port 443 namevhost https-test1.example.com (/etc/httpd/vhosts.d/000-https-test1.example.

```

```

        port 443 namevhost https-test2.example.com (/etc/httpd/vhosts.d/001-https-test2.example.com)
*:80          is a NameVirtualHost
              default server https-test1.example.com (/etc/httpd/vhosts.d/000-https-test1.example.com)
        port 80 namevhost https-test1.example.com (/etc/httpd/vhosts.d/000-https-test1.example.com)
        port 80 namevhost https-test2.example.com (/etc/httpd/vhosts.d/001-https-test2.example.com)
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)
[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!?

```

## References

- 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](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](http://httpd.apache.org/docs/2.2/mod/mod_alias.html)

## Installing QM as a ROOT webapp

QueueMetrics is usually deployed as a webapp which 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

### 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"
```

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

Restart QueueMetrics.

### Changes to QueueMetrics

None necessary.

## Enabling GZIP compression in Tomcat

It is possible to 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 stanza (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

Recent versions of the Java JVM offer 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 6. If this is not your case, you should upgrade the 'queuemetrics-java' package.

## Assessing memory problems

If you feel like you are having memory issues, you should take multiple memory and thread dumps spaced a couple of hours each, 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 developed by Sun 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.dev.java.net/>

## Allowing remote access

The standard JVM settings for QM 'do not' allow remote access over network, for obvious security reasons. In order to allow it, you should add the following line to '/etc/init.d/queuemetrics' (all on one line):

```
export JAVA_OPTS="-Dcom.sun.management.jmxremote.port=9999
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false $JAVA_OPTS"
```

Restart the JVM after adding it. You can change the port (in this case we set it to 9999) for a minimal security.

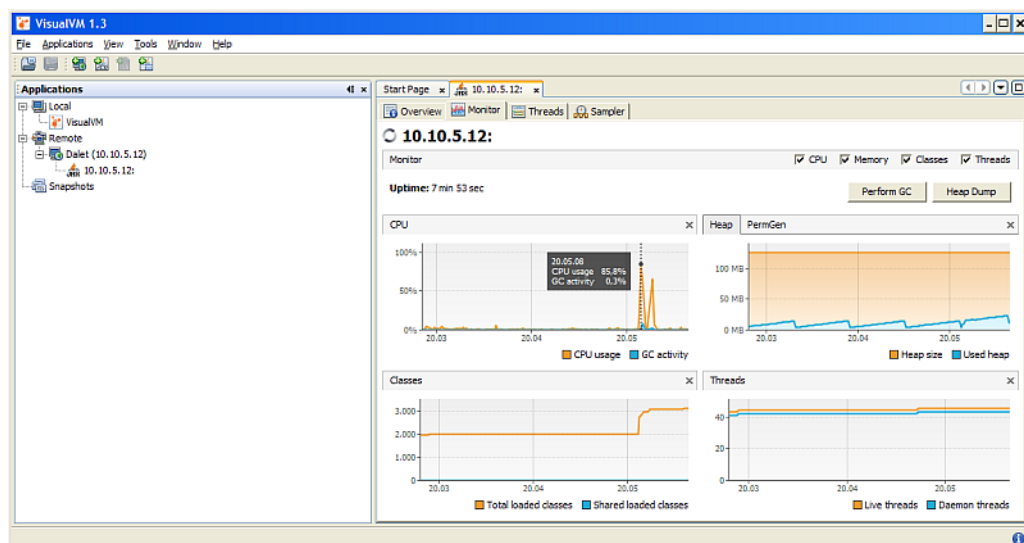
## Starting VisualVM

To start VisualVM, you run 'bin/visualvm.exe'.

When started, click on "Add server" 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 (9999 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:9999".

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

## 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 connection 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 getting started, 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>

        <!-- 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 to 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&jdbcComplia
</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-xxx.jar', to the '/common/lib' directory of your Tomcat installation and remove it from 'WEB-INF/lib' of your QueueMetrics instance.

Restart QueueMetrics.

## Monitoring and fixing "slow queries" in MySQL

```

# Time: 090603 23:07:28
# User@Host: queuemetrics[queuemetrics] @ localhost [127.0.0.1]

```

```
# Query_time: 18 Lock_time: 0 Rows_sent: 3260 Rows_examined: 2474525
SELECT time_id , call_id , queue , agent , verb , data1 , data2 , data3 , data4
FROM queue_log WHERE partition = 'P001' AND (time_id >= '1243980000' AND time_id <= '1577833260')
AND queue IN ( ' ', 'NONE' , 'none' , 'ABCD' )
ORDER BY time_id ASC , unique_row_count ASC;
```

18 seconds for what is essentially a simple query. Compare the "rows sent" versus "rows examined". There are 2.4m rows in the database. (2.4m rows is not a lot of data, I have worked on tables with 100m rows in them).

That can only mean our query is not using an index or our indexes are not working.

So, I did an "explain"...and saw that you do have a multiple column index and that it is being considered and in fact used. But the mysql query is still slow.

```
mysql> explain SELECT time_id , call_id , queue , agent , verb , data1 , data2 , data3 , data4
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table      | type | possible_keys          | key | key_len | ref  | rows | Ext |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE      | queue_log | ref  | idx_sel,partizione_b | partizione_b | 22      |      | const | 310 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.10 sec)
```

## Prerequisites

- A working QueueMetrics instance

## Changes to MySQL server

xxx

## Changes to QueueMetrics

None necessary.

## 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

## Changes to MySQL

- Add index?
- Convert to MySQL?

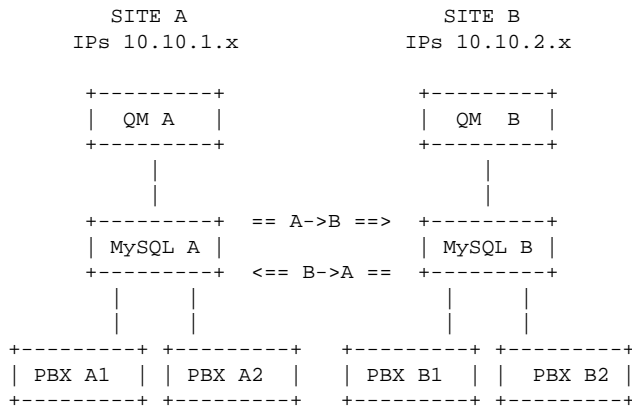
## Changes to QueueMetrics

None necessary.

## 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 an usable (local) system even in case the WAN is unavailable.



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.

Here is 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 their 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 Qloaders 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: exampleddb
      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 their 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 to write 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 instead; 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 have only 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.

\todo TO BE COMPLETED

## 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 find them more 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 recording as per 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.



### Warning

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.



### Warning

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 to ring. With modifications listed here is possible to have ringing phone information as soon as a call enter 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 could not be written in the *queue\_log* but the standard RINGNOANSWER information is available directly from asterisk.

Realtime ringing information is feed to QueueMetrics by mean 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
```

We suppose here to have three members tied to the extensions 100, 101 and 102 defined in the *from-internal-custom* context. We suppose that the extension 200@from-internal-custom [mailto:200@from-internal-custom] is the entry point for the queue *queue\_dps*.

The *extensions.conf* should be defined like:

```
[from-internal-custom]

exten => 100,1,System( echo "${EPOCH}|${PCHANNEL}|queue-dps|SIP/${EXTEN}|AGENTATTEMPT" >> /var/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/
exten => 101,n,Dial(SIP/101)
```

```
exten => 101,n,Hangup()

exten => 102,1,System( echo "${EPOCH}|${PCHANNEL}|queue-dps|SIP/${EXTEN}|AGENTATTEMPT" >> /var/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.