

Writing syslog messages to MySQL

by: Rainer Gerhards, 09/01/2005

<http://www.securitydocs.com/library/3555>

Abstract

In this paper, I describe how to write syslog messages to a MySQL database. Having syslog messages in a database is often handy, especially when you intend to set up a front-end for viewing them. This paper describes an approach with [rsyslogd](#), an alternative enhanced syslog daemon natively supporting MySQL. I describe the components needed to be installed and how to configure them.

Background

In many cases, syslog data is simply written to text files. This approach has some advantages, most notably it is very fast and efficient. However, data stored in text files is not readily accessible for real-time viewing and analysis. To do that, the messages need to be in a database. There are various ways to store syslog messages in a database. For example, some have the syslogd write text files which are later feed via a separate script into the database. Others have written scripts taking the data (via a pipe) from a non-database-aware syslogd and store them as they appear. Some others use database-aware syslogds and make them write the data directly to the database. In this paper, I use that "direct write" approach. I think it is superior, because the syslogd itself knows the status of the database connection and thus can handle it intelligently (well ... hopefully ;)). I use rsyslogd to accomplish this, simply because I have initiated the rsyslog project with database-awareness as one goal.

One word of caution: while message storage in the database provides an excellent foundation for interactive analysis, it comes at a cost. Database i/o is considerably slower than text file i/o. As such, directly writing to the database makes sense only if your message volume is low enough to allow a) the syslogd, b) the network, and c) the database server to catch up with it. Some time ago, I have written a paper on [optimizing syslog server performance](#). While this paper talks about Window-based solutions, the ideas in it are generic enough to apply here, too. So it might be worth reading if you anticipate medium high to high traffic. If you anticipate really high traffic (or very large traffic spikes), you should seriously consider forgetting about direct database writes - in my opinion, such a situation needs either a very specialised system or a different approach (the text-file-to-database approach might work better for you in this case).

Overall System Setup

In this paper, I concentrate on the server side. If you are thinking about interactive syslog message review, you probably want to centralize syslog. In such a scenario, you have multiple machines (the so-called clients) send their data to a central machine (called server in this context). While I expect such a setup to be typical when you are interested in storing messages in the database, I do not describe how to set it up. This is beyond the scope of this paper. If you search a little, you will probably find many good descriptions on how to centralize syslog. If you do that, it might be a good idea to do it securely, so you might also be interested in my paper on [ssl-encrypting syslog message transfer](#).

No matter how the messages arrive at the server, their processing is always the same. So you can use this paper in combination with any description for centralized syslog reporting.

As I already said, I use rsyslogd on the server. It has intrinsic support for talking to MySQL databases. For obvious reasons, we also need an instance of MySQL running. To keep us focussed, the setup of MySQL itself is also beyond the scope of this paper. I assume that you have successfully installed MySQL and also have a front-end at hand to work with it (for example, [phpMyAdmin](#)). Please make sure that this is installed, actually working and you have a basic understanding of how to handle it.

Setting up the system

You need to download and install rsyslogd first. Obtain it from the rsyslog site. Make sure that you disable stock syslog, otherwise you will experience some difficulties.

It is important to understand how rsyslogd talks to the database. In rsyslogd, there is the concept of "templates". Basically, a template is a string that includes some replacement characters, which are called "properties" in rsyslog. Properties are accessed via the "Property Replacer". Simply said, you access properties by including their name between percent signs inside the template. For example, if the syslog message is "Test", the template "%msg%" would be expanded to "Test". Rsyslogd supports sending template text as a SQL statement to MySQL. As such, the template must be a valid SQL statement. There is no limit in what the statement might be, but there are some obvious and not so obvious choices. For example, a template "drop table xxx" is possible, but does not make an awful lot of sense. In practice, you will always use an "insert" statement inside the template.

An example: if you would just like to store the msg part of the full syslog message, you have probably created a table "syslog" with a single column "message". In such a case, a good template would be "insert into syslog(message) values ('%msg%')". With the example above, that would be expanded to "insert into syslog(message) values('Test')". This expanded string is then sent to the database. It's that easy, no special magic. The only thing you must ensure is that your template expands to a proper SQL statement and that this statement matches your database design.

Does that mean you need to create database schema yourself and also must fully understand rsyslogd's properties? No, that's not needed. Because we anticipated that folks are probably more interested in getting things going instead of designing them from scratch. So we have provided a default schema as well as build-in support for it. This schema also offers an additional benefit: rsyslog is part of [Adiscon's MonitorWare product line](#) (which includes open source and closed source members). All of these tools share the same default schema and know how to operate on it. For this reason, the default schema is also called the "MonitorWare Schema". If you use it, you can simply add [phpLogCon, a GPLed syslog web interface](#), to your system and have instant interactive access to your database. So there are some benefits in using the provided schema.

The schema definition is contained in the file "createDB.sql". It comes with the rsyslog package. Review it to check that the database name is acceptable for you. Be sure to leave the table and field names unmodified, because otherwise you need to customize rsyslogd's default sql template, which we do not do in this paper. Then, run the script with your favourite MySQL tool. Double-check that the table was successfully created.

Next, we need to tell rsyslogd to write data to the database. As we use the default schema, we do NOT need to define a template for this. We can use the hardcoded one (rsyslogd handles the proper template linking). So all we need to do is add a simple selector line to /etc/rsyslog.conf:

```
*.* >database-server,database-name,database-userid,database-passwor
```

In many cases, MySQL will run on the local machine. In this case, you can simply use "127.0.0.1" for *database-server*. This can be especially advisable, if you do not need to expose MySQL to any process outside of the local machine. In this case, you can simply bind it to 127.0.0.1, which provides a quite secure setup. Of course, also supports remote MySQL instances. In that case, use the remote server name (e.g. mysql.example.com) or IP-address. The *database-name* by default is "syslog". If you have modified the default, use your name here. *Database-userid* and *-password* are the credentials used to connect to the database. As they are stored in clear text in rsyslog.conf, that user should have only the least possible privileges. It is sufficient to grant it INSERT privileges to the systemevents table, only. As a side note, it is strongly advisable to make the rsyslog.conf file readable by root only - if you make it world-readable, everybody could obtain the password (and eventually other vital information from it). In our example, let's assume you have created a MySQL user named "syslogwriter" with a password of "topsecret" (just to say it bluntly: such a password is NOT a good idea...). If your MySQL database is on the local machine, your rsyslog.conf line might look like in this sample:

```
*.* >127.0.0.1,syslog,syslogwriter,topsecret
```

Save rsyslog.conf, restart rsyslogd - and you should see syslog messages being stored in the "systemevents" table!

The example line stores every message to the database. Especially if you have a high traffic volume, you will probably limit the amount of messages being logged. This is easy to accomplish: the "write database" action is just a regular selector line. As such, you can apply normal selector-line filtering. If, for example, you are only interested in messages from the mail subsystem, you can use the following selector line:

```
mail.* >127.0.0.1,syslog,syslogwriter,topsecret
```

Review the [rsyslog.conf documentation](#) for details on selector lines and their filtering.

You have now completed everything necessary to store syslog messages to the MySQL database. If you would like to try out a front-end, you might want to look at phpLogCon, which displays syslog data in a browser. As of this writing, [phpLogCon](#) is not yet a powerful tool, but it's open source, so it might be a starting point for your own solution.

On Reliability...

Rsyslogd writes syslog messages directly to the database. This implies that the database must be available at the time of message arrival. If the database is offline, no space is left or something else goes wrong - rsyslogd can not write the database record. If rsyslogd is unable to store a message, it performs one retry. This is helpful if the database server was restarted. In this case, the previous connection was broken but a reconnect immediately succeeds. However, if the database is down for an extended period of time, an immediate retry does not help. While rsyslogd could retry until it finally succeeds, that would have negative impact. Syslog messages keep coming in. If rsyslogd would be busy retrying the database, it would not be able to process these messages. Ultimately, this would lead to loss of newly arrived messages.

In most cases, rsyslogd is configured not only to write to the database but to perform other actions as well. In the always-retry scenario, that would mean no other actions would be carried out. As such, the design of rsyslogd is limited to a single retry. If that does not succeed, the current message is will not be written to the database and the MySQL database writer be suspended for a short period of time. Obviously, this leads to the loss of the current message as well as all messages received during the suspension period. But they are only lost in regard to the database, all other actions are correctly carried out. While not perfect, we consider this to be a better approach then the potential loss of all messages in all actions.

In short: try to avoid database downtime if you do not want to experience message loss.

Please note that this restriction is not rsyslogd specific. All approaches to real-time database storage share this problem area.

Conclusion

With minimal effort, you can use rsyslogd to write syslog messages to a MySQL database. Once the messages are arrived there, you can interactively review and analyse them. In practice, the messages are also stored in text files for longer-term archival and the databases are cleared out after some time (to avoid becoming too slow). If you expect an extremely high syslog message volume, storing it in real-time to the database may outperform your database server. In such cases, either filter out some messages or think about alternate approaches involving non-real-time database writing (beyond the scope of this paper).

The method outline in this paper provides an easy to setup and maintain solution for most use cases, especially with low and medium syslog message volume (or fast database servers).

I have set up a site to [demo web access to syslog data](#). It is build using the steps outlined here and uses phpLogCon as the front-end. You might want to visit it to get a glimpse of how such a beast might look.

Feedback Requested

I would appreciate feedback on this paper. If you have additional ideas, comments or find bugs, please [let me know](#).

References and Additional Material

www.rsyslog.com - the rsyslog site

demo.rsyslog.com - demo site for rsyslog and the phpLogCon web interface

[Paper on Syslog Server Optimization](#)

Revision History

2005-08-02 * [Rainer Gerhards](#) * initial version created

2005-08-03 * [Rainer Gerhards](#) * added references to demo site

Copyright

Copyright (c) 2005 [Rainer Gerhards](#) and [Adiscon](#).

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license can be viewed at <http://www.gnu.org/copyleft/fdl.html>.