syslog-ng

Note: With systemd's journal, syslog-ng is not needed by most users.

Related articles

rsyslog

Contents

Overview

syslog-ng takes incoming log messages from defined '**sources**' and forwards them to the appropriate **destinations**, based on powerful **filter** directives. In a typical simple set-up, syslog-ng will read messages from three sources:

1. the default `/dev/log` device, where most logs are sent
2. syslog-ng "internal" log messages
3. `/proc/kmsg` kernel messages

Sources are defined using the "source" directive. These incoming messages are then filtered according to defined filters ("filter" keyword), i.e. according to originating program or log level, and sent to the appropriate "destination". Destinations include log files (e.g.

`/var/log/messages.log`), printing messages on a console and remote servers. The pivotal function is **log**. This function defines which filters should be applied to a certain source, and where the resulting messages should be sent to.

Enable syslog-ng with the `syslog-ng.service` service file. As of *systemd* 216, messages are no longer forwarded to syslog by default. Syslog-ng did not become journald aware until months later with the release of syslog-ng 3.6. This meant that if you were running systemd 216 or greater and syslog-ng you needed to set the option `ForwardToSyslog=yes` in `/etc/systemd/journald.conf` to actually use *syslog-ng* with *journald*.

If you use a current **`syslog-ng (https://www.archlinux.org/packages/?name=syslog-ng)`**, it is not necessary to change the option because **syslog-ng** pulls the messages from the journal. If you have set `ForwardToSyslog=yes` you should revert it to `ForwardToSyslog=no` in order to avoid the overhead associated with the socket and to avoid **needless error messages in the log (https://github.com/balabit/syslog-ng/issues/314)**. If on the other hand you do not want to store your logs twice and turn *journald*'s `Storage=none`, you **will** need `ForwardToSyslog=yes`, as *syslog-ng* tries to follow the 'journald' journal file.

Sources

syslog-ng receives log messages from a source. To define a source you should follow the following syntax:

```
source <identifier> { source-driver(params); source-driver(params); ... };
```

You can look at the identifiers and source-drivers in the **official manuals (http://www.balabit.com/support/documentation/)**. This will follow the manual to explain the configuration file above. The unix-stream() source-driver opens the given AF_UNIX **socket** and starts listening on it for messages. The internal() source-driver gets messages generated by syslog-ng.

Therefore, the following means: `src` gets messages from the `/dev/log` socket and syslog-ng.

```
source src { unix-stream("/dev/log"); internal(); };
```

The kernel sends log messages to `/proc/kmsg` and the file() driver reads log messages from files. Therefore, the following means: kernsrc gets messages from file `/proc/kmsg`

```
source kernsrc { file("/proc/kmsg"); };
```

In the default configuration file after emerging syslog-ng, the source is defined as:

```
source src { unix-stream("/dev/log"); internal(); pipe("/proc/kmsg"); };
```

Reading messages by `pipe("/proc/kmsg")` gives a better performance but because it opens its argument in read-write mode can be a security hazard as the **syslog-ng admin guide (http://www.balabit.com/sites/default/files/documents/syslog-ng-v3.0-guide-admin-en.html/index.html-single.html#configuring_sources_pipe)** states in section 3.3.3:

"Pipe is very similar to the file() driver, but there are a few differences, for example pipe() opens its argument in read-write mode, therefore it is not recommended to be used on special files like `/proc/kmsg`." (You can follow this discussion in **this post (http://forums.gentoo.org/viewtopic-t-558161.html)**.)

To open a port to read data from a remote server a source must be defined with this syntax:

```
source s_net { udp(); };
```

for UDP or

```
source s_net { tcp(); };
```

to receive log messages via TCP. Both listen on port 514.

syslog-ng and systemd journal

Starting with syslog-ng version 3.6.1 the default `system()` source on Linux systems using systemd uses journald as its standard `system()` source.

If you wish to use both the journald and syslog-ng files, ensure the following settings are in effect. For systemd-journald, in the `/etc/systemd/journald.conf` file, `Storage=` either set to `auto` or unset (which defaults to auto) and `ForwardToSyslog=` set to `no` or unset (defaults to no). For `/etc/syslog-ng/syslog-ng.conf`, you need the following `source` stanza:

```
source src {
  system();
  internal();
};
```

If, on the other hand, you wish *not* to retain the journald logs, but only syslog-ng's text logs, set `Storage=volatile` and `ForwardToSyslog=yes` in `/etc/systemd/journald.conf`. This will store journald in ram. As of syslog-ng 3.6.3, syslog-ng is using journald as the system(); source so if you set `Storage=none`, the systemd journal will drop all messages and **not** forward them to syslog-ng.

After the change **restart** the `systemd-journald.service` and `syslog-ng.service` daemons.

Destinations

In syslog-ng, log messages are sent to files. The syntax is very similar to sources:

```
destination <identifier> {destination-driver(params); destination-driver(params); ... };
```

You will be normally logging to a file, but you could log to a different destination-driver: pipe, Unix socket, TCP-UDP ports, terminals or to specific programs. Therefore, this means sending authlog messages to `/var/log/auth.log`:

```
destination authlog { file("/var/log/auth.log"); };
```

If the user is logged in, `usertty()` sends messages to the terminal of the specified user. If you want to send console messages to root's terminal if it is logged in:

```
destination console { usertty("root"); };
```

Messages can be sent to a pipe with `pipe()`. The following sends xconsole messages to the pipe `/dev/xconsole`. This needs some more configuration, so you could look at the sub-section xconsole below.

```
destination xconsole { pipe("/dev/xconsole"); };
```

To send messages on the network, use `udp()`. The following will send your log data out to another server.

```
destination remote_server { udp("10.0.0.2" port(514)); };
```

Creating Filters for Messages

The syntax for the filter statement is:

```
filter <identifier> { expression; };
```

Functions can be used in the expression, such as the function `facility()` which selects messages based on the facility codes. The Linux kernel has a few facilities you can use for logging. Each facility has a log-level; where debug is the most verbose, and panic only shows serious errors. You can find the facilities, log levels and priority names in `/usr/include/sys/syslog.h`. To filter those messages coming from authorization, like *May 11 23:42:31 mimosinnet su(pam_unix)[18569]: session opened for user root by (uid=1000)*, use the following:

```
filter f_auth { facility(auth); };
```

The facility expression can use the boolean operators `and`, `or`, and `not`, so the following filter selects those messages not coming from authorization, network news or mail:

```
filter f_debug { not facility(auth, authpriv, news, mail); };
```

The function `level()` selects messages based on its priority level, so if you want to select informational levels:

```
filter f_info { level(info); };
```

Functions and boolean operators can be combined in more complex expressions. The following line filters messages with a priority level from informational to warning not coming from auth, authpriv, mail and news facilities:

```
filter f_messages { level(info..warn) and not facility(auth, authpriv, mail, news); };
```

Messages can also be selected by matching a regular expression in the message with the function `match("regex" value("TEMPLATE"))`. For example:

```
filter f_failed { match("failed" value("MESSAGE")); };
```

here is a list of templates :

```
 "AMPM", "BSDTAG", "DATE, C_DATE, R_DATE, S_DATE", "DAY, C_DAY, R_DAY, S_DAY", "FACILITY", "FACILITY_NUM", "FULLDATE, C_FULLDATE, R_FULLDATE, S_FULL
DATE", "FULLHOST", "FULLHOST_FROM", "HOUR, C_HOUR, R_HOUR, S_HOUR", "HOUR12, C_HOUR12, R_HOUR12, S_HOUR12", "HOST", "HOST_FROM", "ISODATE, C_ISODAT
E, R_ISODATE, S_ISODATE", "LEVEL_NUM", "LOGHOST", "MIN, C_MIN, R_MIN, S_MIN", "MONTH, C_MONTH, R_MONTH, S_MONTH", "MONTH_ABBREV, C_MONTH_ABBREV, R_M
ONTH_ABBREV, S_MONTH_ABBREV", "MONTH_NAME, C_MONTH_NAME, R_MONTH_NAME, S_MONTH_NAME", "MONTH_WEEK, C_MONTH_WEEK, R_MONTH_WEEK, S_MONTH_WEEK", "MSEC,
C_MSEC, R_MSEC, S_MSEC", "MSG or MESSAGE", "MSGHDR", "MSGID", "MSGONLY", "PID", "PRI", "PRIORITY or LEVEL", "PROGRAM", "SDATA, .SDATA.SDID.SDNAME",
"SEC, C_SEC, R_SEC, S_SEC", "SOURCEIP", "SEQNUM", "STAMP, R_STAMP, S_STAMP", "SYSUPTIME", "TAG", "TAGS", "TZ, C_TZ, R_TZ, S_TZ", "TZOFFSET, C_TZOFFS
ET, R_TZOFFSET, S_TZOFFSET", "UNIXTIME, C_UNIXTIME, R_UNIXTIME, S_UNIXTIME", "USEC, C_USEC, R_USEC, S_USEC", "YEAR, C_YEAR, R_YEAR, S_YEAR", "WEEK,
C_WEEK, R_WEEK, S_WEEK", "WEEK_ABBREV, C_WEEK_ABBREV, R_WEEK_ABBREV, S_WEEK_ABBREV", "WEEK_DAY, C_WEEK_DAY, R_WEEK_DAY, S_WEEK_DAY", "WEEKDAY, C_WEE
KDAY, R_WEEKDAY, S_WEEKDAY", "WEEK_DAY_NAME, C_WEEK_DAY_NAME, R_WEEK_DAY_NAME, S_WEEK_DAY_NAME".
```

To filter messages received from a particular remote host, the `host()` function must be used:

```
filter f_host { host( "192.168.1.1" ); };
```

Log Paths

syslog-ng connects sources, filters and destinations with log statements. The syntax is:

```
log {source(s1); source(s2); ...
filter(f1); filter(f2); ...
destination(d1); destination(d2); ...
flags(flag1[, flag2...]); };
```

The following for example sends messages from `src` source to `mailinfo` destination filtered by `f_info` filter.

```
log { source(src); filter(f_mail); filter(f_info); destination(mailinfo); };
```

Tips and Tricks

After understanding the logic behind syslog-ng, many possible and complex configuration are possible. Here there are some examples.

Have syslog-ng reload the configuration file

You can make syslog-ng re-evaluate the configuration file. You can do so manually by sending a `SIGHUP` to the process, or call the reload function with systemctl:

```
# systemctl reload syslog-ng
```

Failover Logging to Remote Host

This setup shows how to send the default unencrypted syslog packets across both TCP and UDP protocols, using the standard port (514) and an alternate port. This is sending the same output to the same machine 4 different ways to try and make sure packets make it. Mostly useful if you are debugging a remote server that fails to reboot. The different ports and protocols are to make it past any firewall filters or other network problems. Also useful for port-forwarding and using tunnels. Something like this setup is ideal to tunnel across an ssh connection that the prone-to-failover host initiates through a reverse connection.

```
#sending to a remote syslog server on TCP and UDP ports (not encrypted)
destination askapache_failover_loghost {
    tcp("208.86.158.195" port(25214));
    udp("208.86.158.195" port(25214));
    udp("mysyslog1.dyndns.org" port(514));
};
log {
    source(src);
    destination(askapache_failover_loghost);
};
```

And then on the loghost receiving these logs:

```
#a USB redirected console for flexible viewing
destination debugging_console {
    file("/dev/ttyU1");
};

# listens on IP addresses and ports, sets the incoming settings
source prone_to_failover_host {
    tcp(ip(208.86.158.195),port(25214));
    udp(ip(208.86.158.195) port(25214));

    udp(default-facility(syslog) default-priority(emerg));
    tcp(default-facility(syslog) default-priority(emerg));
}

# log it
log {
    source(prone_to_failover_host);
    destination(debugging_console);
};
```

Move log to another file

In order to move some log from `/var/log/messages` to another file:

```
#sshd configuration
destination ssh { file("/var/log/ssh.log"); };
filter f_ssh { program("sshd"); };
log { source(src); filter(f_ssh); destination(ssh); };
```

Configuring as a loghost

Configuring your system to be a loghost is quite simple. Drop the following into your configuration, and create the needed directory. With this simple configuration, log filenames will be based on the **FQDN** of the remote host, and located in `/var/log/remote/`. After creating the remote directory, reload your syslog-ng configuration.

```
source net { udp(); };
destination remote { file("/var/log/remote/${FULLHOST}-log"); };
log { source(net); destination(remote); };
```

Improve Performance

syslog-ng's performance can be improved in different ways:

Write every so often

It seems that the old `sync(X)` **option** is called `flush_lines(X)` now, where the writing to the file is buffered for `X` lines. Default is 0 (no buffering).

Avoid redundant processing and disk space

A single log message can be sent to different log files several times. For example, in the initial configuration file, we have the following definitions:

```
destination cron { file("/var/log/cron.log"); };
destination messages { file("/var/log/messages"); };
filter f_cron { facility(cron); };
filter f_messages { level(info..warn)
       and not facility(auth, authpriv, mail, news); };
log { source(src); filter(f_cron); destination(cron); };
log { source(src); filter(f_messages); destination(messages); };
```

The same message from the `cron` facility will end up in both the `cron.log` and `messages` files. To change this behavior we can use the `final` flag, ending up further processing with the message. Therefore, in this example, if we want messages from the `cron` facility not ending up in the messages file, we should change the cron's log sentence by:

```
log { source(src); filter(f_cron); destination(cron); flags(final); };
```

another way is to exclude the `cron` facility from `f_messages` filter:

```
filter f_messages { level(info..warn) and not facility(cron, auth, authpriv, mail, news); };
```

PostgreSQL Destination

This section will use two roles: `syslog` and `logwriter`. `syslog` will be the administrator of the database `syslog` and `logwriter` will only be able to add records to the `logs` table.

No longer needed to create table for logs. syslog-ng will create automatically.

```
psql -U postgres
```

```
postgres=# CREATE ROLE syslog WITH LOGIN;
postgres=# \password syslog    # Using the \password function is secure because
postgres=# CREATE ROLE logwriter WITH LOGIN;
postgres=# \password logwriter # the password is not saved in history.
postgres=# CREATE DATABASE syslog OWNER syslog;
postgres=# \q # You are done here for the moment
```

Edit `pg_hba.conf` to allow `syslog` and `logwriter` to establish a connection to PostgreSQL.

```
/var/lib/postgres/data/pg_hba.conf
```

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD

host    syslog      logwriter   192.168.0.1/24        md5
host    syslog      syslog      192.168.0.10/32       md5
```

Tell PostgreSQL to reload the configuration files:

```
# systemctl reload postgresql
```

Edit `/etc/syslog-ng/syslog-ng.conf` so that it knows where and how to write to PostgreSQL. syslog-ng will utilize the `logwriter` role.

```
...
#
# SQL logging support
#

destination d_pgsql {
  sql(type(pgsql)
  host("127.0.0.1") username("logwriter") password("password")
  database("syslog")
  table("logs_${HOST}_${R_YEAR}${R_MONTH}${R_DAY}") #or whatever you want, example ${HOST}" for hosts, ${LEVEL}" for levels.. etc
  columns("datetime timestamp with time zone", "host varchar(32)", "program varchar(16)", "pid varchar(16)", "message varchar(200)")
  values("$R_ISODATE", "$HOST", "$PROGRAM", "$PID", "$MSG")
  indexes("datetime", "host", "program", "pid", "message"));
};

log { source(src); destination(d_pgsql); };
```

Finally, restart syslog-ng.

```
# systemctl restart syslog-ng
```

And check to see if things are being logged.

```
psql -U logwriter -d syslog
syslog=> SELECT * FROM <your table name> ORDER BY datetime DESC LIMIT 10;
```

ISO 8601 timestamps

Before :

```
#logger These timestamps are not optimal.
#tail -n 1 /var/log/messages.log
Feb 18 14:25:01 hostname logger: These timestamps are not optimal.
#
```

Add `ts_format(iso);` to `/etc/syslog-ng/syslog-ng.conf` in the options section. Example:

```
options {
  stats_freq (0);
  flush_lines (0);
  time_reopen (10);
  log_fifo_size (1000);
  long_hostnames(off);
  use_dns (no);
  use_fqdn (no);
  create_dirs (no);
  keep_hostname (yes);
  perm(0640);
  group("log");
  ts_format(iso);      #make ISO-8601 timestamps
  #frac-digits(3);     #optional time to nearest millisecond
};
```

Then:

```
# systemctl reload syslog-ng
```

After :

```
#logger Now THAT is a timestamp!
#tail -n 2 /var/log/messages.log
Feb 18 14:25:01 hostname logger: These timestamps are not optimal.
2010-02-18T20:23:58-05:00 electron logger: Now THAT is a timestamp!
#
```

RFC 3339 timestamps

Same as above, except use `rfc3339` instead of `iso` for `ts_format`

Log Levels

Log levels are defined separately for each logged facility in syslog-ng config. Available log levels are listed in /usr/include/sys/syslog.h :

```
define LOG_EMERG       0       /* system is unusable */
define LOG_ALERT       1       /* action must be taken immediately */
define LOG_CRIT        2       /* critical conditions */
define LOG_ERR         3       /* error conditions */
define LOG_WARNING     4       /* warning conditions */
define LOG_NOTICE      5       /* normal but significant condition */
define LOG_INFO        6       /* informational */
define LOG_DEBUG       7       /* debug-level messages */
```

Macros and Variables

Macros can be used in both templates, and in destination file names. **Macros of syslog-ng OSE (http://www.balabit.com/sites/default/files/documents/syslog-ng-ose-3.4-guides/en/syslog-ng-ose-v3.4-guide-admin/html/reference-macros.html)**.

The following code will write the log lines to `/var/log/test.log` in the format of `macroname=value@`.

```
template t_test { template("PROGRAM=$PROGRAM@PID=$PID@BSDTAG=$BSDTAG@TAG=$TAG@TAGS=$TAGS@FACILITY=$FACILITY@FACILITY_NUM=$FACILITY_NUM@LEVEL=$LEVEL@
LEVEL_NUM=$LEVEL_NUM@PRI=$PRI@PRIORITY=$PRIORITY@FULLHOST=$FULLHOST@FULLHOST_FROM=$FULLHOST_FROM@HOST=$HOST@HOST_FROM=$HOST_FROM@LOGHOST=$LOGHOST@MS
GHDR=$MSGHDR@MSGID=$MSGID@MSGONLY=$MSGONLY@MSG=$MSG@MESSAGE=$MESSAGE@SOURCE=$SOURCE@SOURCEIP=$SOURCEIP@SOURCE_IP=$SOURCE_IP@SEQNUM=$SEQNUM@UNIXTIME=
$UNIXTIME@FULLDATE=$FULLDATE@ISODATE=$ISODATE@DATE=$DATE@STAMP=$STAMP@TZ=$TZ@TZOFFSET=$TZOFFSET@SEC=$SEC@MIN=$MIN@HOUR=$HOUR@HOUR12=$HOUR12@DAY=$DAY
@WEEK=$WEEK@WEEK_DAY=$WEEK_DAY@WEEK_DAY_ABBREV=$WEEK_DAY_ABBREV@WEEK_DAY_NAME=$WEEK_DAY_NAME@MONTH=$MONTH@MONTH_ABBREV=$MONTH_ABBREV@MONTH_NAME=$MON
TH_NAME@MONTH_WEEK=$MONTH_WEEK@YEAR=$YEAR@YEAR_DAY=$YEAR_DAY
\n"); template_escape(no); };

destination d_test { file("/var/log/test.log" template(t_test)); };

log { source(s_local); destination(d_test); flags(final); };
```

You can create your own value list as below once syslog-ng is restarted with: `tail /var/log/test.log|tr "@" "\n"`

```
PROGRAM=kernel
PID=
BSDTAG=4A
TAG=04
TAGS=.source.s_local
FACILITY=kern
FACILITY_NUM=0
LEVEL=warning
LEVEL_NUM=4
PRI=4
PRIORITY=warning
```

```
FULLHOST=www.askapache.com
FULLHOST_FROM=www.askapache.com
HOST=www.askapache.com
HOST_FROM=www.askapache.com
LOGHOST=
MSGHDR=kernel:
MSGID=
MSGONLY=Firewall: *INVALID* IN=eth0 OUT= MAC=00:00 SRC=x.x.x.x DST=198.101.159.98 LEN=40 TOS=0x00 PREC=0x00 TTL=113 ID=7730 DF PROTO=TCP SPT=52369 D
PT=80 WINDOW=0 RES=0x00 ACK RST URGP=0
MSG=Firewall: *INVALID* IN=eth0 OUT= MAC=00:00 SRC=x.x.x.x DST=198.101.159.98 LEN=40 TOS=0x00 PREC=0x00 TTL=113 ID=7730 DF PROTO=TCP SPT=52369 DPT=8
0 WINDOW=0 RES=0x00 ACK RST URGP=0
MESSAGE=Firewall: *INVALID* IN=eth0 OUT= MAC=00:00 SRC=x.x.x.x DST=198.101.159.98 LEN=40 TOS=0x00 PREC=0x00 TTL=113 ID=7730 DF PROTO=TCP SPT=52369 D
PT=80 WINDOW=0 RES=0x00 ACK RST URGP=0
SOURCE=s_local
SOURCEIP=127.0.0.1
SOURCE_IP=
UNIXTIME=1369742458
FULLDATE=2013 May 28 08:00:58
ISODATE=2013-05-28T08:00:58-04:00
DATE=May 28 08:00:58
STAMP=2013-05-28T08:00:58-04:00
TZ=-04:00
TZOFFSET=-04:00
SEC=58
MIN=00
HOUR=08
HOUR12=
DAY=28
WEEK=21
WEEK_DAY=3
WEEK_DAY_ABBREV=Tue
WEEK_DAY_NAME=Tuesday
MONTH=05
MONTH_ABBREV=May
MONTH_NAME=May
MONTH_WEEK=4
YEAR=2013
YEAR_DAY=148
```

See Also

- Netconsole A kernel module that sends all kernel log messages (i.e. dmesg) over the network to another computer, without involving user space (e.g. syslogd).

External Links

- syslog-ng OSE Project Page (http://www.balabit.com/network-security/syslog-ng/opensource-logging-system/overview)
- Portal to syslog-ng Documentation (http://www.balabit.com/support/documentation/)
  - The syslog-ng 3.4 Administrator Guide (http://www.balabit.com/sites/default/files/documents/syslog-ng-ose-3.4-guides/en/syslog-ng-ose-v3.4-guide-admin/html/index.html)
  - List of syslog-ng 3.4 Parameters (http://www.balabit.com/sites/default/files/documents/syslog-ng-ose-3.4-guides/en/syslog-ng-ose-v3.4-guide-admin/html/syslog-ng-parameter-index.html)
  - List of syslog-ng 3.4 Macros (http://www.balabit.com/sites/default/files/documents/syslog-ng-ose-3.4-guides/en/syslog-ng-ose-v3.4-guide-admin/html/reference-macros.html)
- syslog-ng Project Page on Freshmeat (http://freshmeat.net/projects/syslog-ng/)
- Gentoo syslog-ng wiki (https://wiki.gentoo.org/wiki/Syslog-ng)
- Gentoo Security Handbook on Logging (http://www.gentoo.org/doc/en/security/security-handbook.xml?part=1&chap=3)
- What is Syslog? Logging with PostgreSQL HOWTO (http://www.pcwdld.com/what-is-syslog-including-servers-and-ports)
- ISO_8601 Wikipedia page for ISO 8601
- RFC 3164 (http://tools.ietf.org/html/rfc3164) - The BSD syslog Protocol
- RFC 5424 (http://tools.ietf.org/html/rfc3164) - The Syslog Protocol

    - **RFC 5425 (http://tools.ietf.org/html/rfc5425)** - Transport Layer Security (TLS) Transport Mapping for Syslog
    - **RFC 5426 (http://tools.ietf.org/html/rfc5425)** - Transmission of Syslog Messages over UDP
    - **RFC 5427 (http://tools.ietf.org/html/rfc5425)** - Textual Conventions for Syslog Management
    - **RFC 5428 (http://tools.ietf.org/html/rfc5425)** - MIB for PacketCable and IPCablecom-Compliant Devices
- **RFC 3339 (http://tools.ietf.org/html/rfc3339)** - Date and Time on the Internet: Timestamps

Retrieved from "https://wiki.archlinux.org/index.php?title=Syslog-ng&oldid=497362"