Audit NodeBrain Module

Release 0.9.03

Audit NodeBrain Module
December 2014
NodeBrain Open Source Project

Release 0.9.03

Author: Ed Trettevik

History

2012-06-04	Title: *Audit NodeBrain Module* Author: Ed Trettevik <eat@nodebrain.org> Publisher: NodeBrain Open Source Project
2012-06-04	Release 0.9.02 • This is a first edition.

Preface

This manual is intended for users of the Audit NodeBrain Module, a plug-in for monitoring system and application log files. The reader is expected to be familiar with the basic concepts of NodeBrain. See www.nodebrain.org for general information on NodeBrain.

The function this module performs was included in the original prototype of NodeBrain as a "log listener", but the code was later removed from the intepreter and repackaged as the Audit module.

Documents

NodeBrain Guide - Information on using `nb`
NodeBrain Tutorial - A gentle introduction to `nb` and the rule language
NodeBrain Language - Rule language syntax and semantics
NodeBrain Library - C API

Document Conventions

Sample code and input/output examples are displayed in a monospace font, indented in HTML and Info, and enclosed in a box in PDF or printed copies. Bold text is used to bring the reader's attention to specific portions of an example. In the following example, the first and last line are associated with the host shell and the lines in between are input or output unique to NodeBrain. The `define` command is highlighted, indicating it is the focus of the example. Lines ending with a backslash \ indicate when a command is continued on the next displayed line. This is supported by the language within source files, but not for other methods of command input. If you copy an example of a command displayed over multiple lines, you must enter it as a single line when used outside the context of a source file.

```
$ nb
> define myFirstRule on(a=1 and b=2) mood="happy";
> assert mood="sad";
> show mood
mood = "sad"
> assert a=1,b=2,c=3,d="This is an example of a long single line that",\
    e="we depict on multiple lines to fit on the documnet page";
2008/06/05 12:09:08 NB000I Rule myFirstRule fired(mood="happy")
> show mood
mood = "happy"
> quit
$
```

Table of Contents

1 Concepts

The Audit module implements nodes that monitor lines of text written to system and application log files. This enables event monitoring in situations where an application provides no other mechanism for communicating events to a monitoring system.

1.1 Translators

NodeBrain translators are used to convert a sequence of text strings into a sequence of NodeBrain commands. Regular expressions are used to find specific patterns of text and matching strings are combined with new text to construct NodeBrain commands. An Audit node follows the end of a growing text file and passes each new line into a configured translator. See the *NodeBrain Language Reference* for information on coding translators.

1.2 Log File Rotation

An enabled Audit node opens the log file on a scheduled frequency and reads from the last end-of-file to the new end-of-file and closes the file. Before reading, the new size of the file is checked. If the file is smaller than the last time, the Audit node starts over and reads from the beginning of the file.

1.3 Missed Messages

New lines in a log file are only monitored by an Audit node while the node is enabled (listening). If the node is disabled or the NodeBrain agent is stopped, new lines are ignored and never processed. When the agent is restarted and/or the node is enabled, subsequent new lines will be monitored. An option to maintain a separate cursor file providing the offset of the next line to monitor, with a check for log rotation based on inode will be included in a future release.

It is also possible for a log file to rotate and fill fast enough to become larger than the size it was the last time the Audit node read to end-of-file. This is unlikely, but possible if the log file in uneven bursts and the Audit node's polling frequency is too slow relative to the rotation frequency.

If your application can not tolerate missed messages, however seldom, you should select a different option than the Audit node for getting messages into NodeBrain. It is sufficient for many situations, but not all.

2 Tutorial

If Edison had a needle to find in a haystack, he would proceed at once with the diligence of the bee to examine straw after straw until he found the object of his search.... I was a sorry witness of such doings, knowing that a little theory and calculation would have saved him ninety per cent of his labor. —Nikola Tesla (1857–1943), *New York Times*, October 19, 1931

Effective review of system and application logs can be like trying to find a needle in a haystack. It requires at least one Edison and one Tesla working as a team. The Audit node module works like Edison when reviewing logs so you can work like Tesla.

An Audit node is similar to the Translator node covered in an earlier tutorial, but differs in the way lines of text are input for translation. An Audit node starts at the end of a log file and periodically checks for new lines to translate. When a log file rolls, the Audit node starts at the beginning of the new log file.

The content of system and application log files can vary significantly depending on the mix of applications on a system and how they are configured. A good strategy is to treat log entries as worthy of investigation by default. Duplicate suppression and other flood protection techniques are helpful when using this strategy. As new log entries are reported and investigated, you can decide if they should be suppressed or handled in a special way.

2.1 Agent Rules

The `syslog.nb` file below provides an agent with an Audit node to monitor a log file called `syslog`. It specifies a translator named `syslog.nbx` and a polling interval of 10 seconds. (A longer interval is recommended for real applications.) It also includes a deduplication cache.

```
#!/usr/local/bin/nb -d
# File: tutorial/Audit/syslog.nb
set log="syslog.log",out=".";
define syslog node cache(~(h(8))):(~(1h):route,appl,group,node,object,severity,text(1));
syslog. define alarm if(text._hitState):$ -|mail.form \
  source=tutorial route="${route}" appl="${appl}" group="${group}" \
  node="${node}" severity="${severity}" text="${text}" >> mail.log
syslog. define audit node audit("syslog","syslog.nbx",~(10s));
```

2.2 Sample Log

Here's a small sample of a log file in the format to use for this tutorial. A copy of this file is stored as `tutorial/Audit/syslog.sample`.

```
Feb  1 19:00:04 smk001 sshd[3972]: Accepted publickey for myuser \
  from ::ffff:192.168.1.100 port 53403 ssh2
Feb  1 19:00:06 smk001 sshd[3980]: Accepted publickey for myuser \
  from ::ffff:192.168.1.101 port 53410 ssh2
Feb  1 19:00:16 smk001 kernel: z90crypt: probe_crypto_domain -> \
  Unable to find crypto domain: No devices found
Feb  1 19:00:46 smk001 kernel: z90crypt: probe_crypto_domain -> \
  Unable to find crypto domain: No devices found
Feb  1 19:01:16 smk001 kernel: z90crypt: probe_crypto_domain -> \
  Unable to find crypto domain: No devices found
Feb  1 19:01:19 smk001 su: (to root) myuser on /dev/pts/1
```

2.3 Translation Rules

Here's a small translator called `syslog.nbx` designed for the log format above.

```
# File: tutorial/Audit/syslog.nbx
([a-zA-Z]+ +\d+ \d\d:\d\d:\d\d [^ ]+ ){
  (^-- MARK --)
  (^\/USR\/SBIN\/CRON\[\d+\]: [^ ]+ CMD)
  (^last message repeated \d+ times)
  (^kernel: ){
    (^z90crypt: probe_crypto_domain -> Unable to find crypto domain: No devices found)
    (^end_request: I\/O error)
    (^dasd_erp.*:Perform logging requested)
    (^dasd:.*:ERP successful)
    :syslog. assert ("syslog","syslog","OS","","","normal","SYS0000 kernel: $[=]");
    }
  (^su: ){
    (^pam_unix2: session (started|finished) for user (nobody|root|wwwadm|cyrus), service su)
    (^\(to (nobody|cyrus)\) root on none)
    :syslog. assert ("syslog","syslog","OS","","","normal","SYS0000 su: $[=]");
    }
  (^sshd\[\d+\]: ){
    (^Accepted password for myuser from ::ffff:.* port \d+ ssh2)
    (^Accepted publickey for myuser from ::ffff:.* port \d+ ssh2)
    (^error: Could not get shadow information for NOUSER)
    (^(?'preport'.*port) \d+ ){
      :syslog. assert ("syslog","syslog","OS","","","normal",\
         "SYS0000 sshd[*]: $[preport]port * $[=]");
      }
    :syslog. assert ("syslog","syslog","OS","","","normal","SYS0000 sshd[*]: $[=]");
    }
  }
():syslog. assert ("syslog","syslog","OS","","","critical","SYS0000 $[-]");
```

2.4 Start the Agent

Before starting the agent, you must touch syslog to make sure it exists.

```
$ ./syslog.nb
2009/02/01 19:56:57 NB000I Argument [1] -d
2009/02/01 19:56:57 NB000I Argument [2] ./syslog.nb
> #!/usr/local/bin/nb -d
> # File: tutorial/Audit/syslog.nb
> set log="syslog.log",out=".";
2009/02/01 19:56:57 NB000I NodeBrain nb will log to syslog.log
> define syslog node cache(~(h(8))):(~(1h):route,appl,group,node,object,severity,text(1));
> syslog. define alarm if(text._hitState):$ -|mail.form \
    source=tutorial route="${route}" appl="${appl}" group="${group}" \
    node="${node}" severity="${severity}" text="${text}" >> mail.log
> syslog. define audit node audit("syslog","syslog.nbx",~(10s));
2009/02/01 19:56:57 NB000I Loading translator "syslog.nbx"
---------- --------
# File: tutorial/Audit/syslog.nbx
([a-zA-Z]+ +\d+ \d\d:\d\d:\d\d [^ ]+ ){
  (^-- MARK --)
  (^\/USR\/SBIN\/CRON\[\d+\]: [^ ]+ CMD)
  (^last message repeated \d+ times)
  (^kernel: ){
    (^z90crypt: probe_crypto_domain -> Unable to find crypto domain: No devices found)
    (^end_request: I\/O error)
    (^dasd_erp.*:Perform logging requested)
    (^dasd:.*:ERP successful)
    :syslog. assert ("syslog","syslog","OS","","","normal","SYS0000 kernel: $[=]");
    }
  (^su: ){
    (^pam_unix2: session (started|finished) for user (nobody|root|wwwadm|cyrus), service su)
    (^\(to (nobody|cyrus)\) root on none)
    :syslog. assert ("syslog","syslog","OS","","","normal","SYS0000 su: $[=]");
    }
  (^sshd\[\d+\]: ){
    (^Accepted password for myuser from ::ffff:.* port \d+ ssh2)
    (^Accepted publickey for myuser from ::ffff:.* port \d+ ssh2)
    (^error: Could not get shadow information for NOUSER)
    (^(?'preport'.*port) \d+ ){
      :syslog. assert ("syslog","syslog","OS","","","normal",\
         "SYS0000 sshd[*]: $[preport]port * $[=]");
      }
    :syslog. assert ("syslog","syslog","OS","","","normal","SYS0000 sshd[*]: $[=]");
    }
  }
():syslog. assert ("syslog","syslog","OS","","","critical","SYS0000 $[-]");
---------- --------
2009/02/01 19:56:57 NB000I Translator "syslog.nbx" loaded successfully.
2009/02/01 19:56:57 NB000I Source file "./syslog.nb" included. size=425
2009/02/01 19:56:57 NB000I NodeBrain nb[6749,3352] daemonizing
$
```

2.5 Grow the Log

Use a shell script to append the sample log file `syslog.sample` to the monitored `syslog` file three times.

```
# File: tutorial/Audit/syslog.sh
cat syslog.sample >> syslog
cat syslog.sample >> syslog
cat syslog.sample >> syslog
```

2.6 Review the Agent Log

After executing the `syslog.sh` script above, your `syslog.log` file should look like this.

```
2009/02/01 19:59:35 NB000I Agent log is syslog.log
2009/02/01 19:59:35 NM000I audit audit: Enabled audit of syslog using syslog.nbx
> syslog. assert ("syslog","syslog","OS","","","normal",\
    "SYS0000 su: (to root) myuser on /dev/pts/1");
2009/02/01 20:00:05 NB000I Rule syslog.alarm fired
: syslog. -|mail.form source=tutorial route="syslog" appl="syslog" group="OS" \
    node="" severity="normal" text="SYS0000 su: (to root) myuser on /dev/pts/1" >> mail.log
[6957] Started: -|mail.form source=tutorial route="syslog" appl="syslog" group="OS" \
    node="" severity="normal" \
    text="SYS0000 su: (to root) myuser on /dev/pts/1" >> mail.log
[6957] Exit(0)
> syslog. assert ("syslog","syslog","OS","","","normal", \
    "SYS0000 su: (to root) myuser on /dev/pts/1");
> syslog. assert ("syslog","syslog","OS","","","normal", \
    "SYS0000 su: (to root) myuser on /dev/pts/1");
> syslog. assert ("syslog","syslog","OS","","","normal", \
    "SYS0000 su: (to root) myuser on /dev/pts/1");
```

Notice that the translator only asserted the `su` log entries to the `syslog` cache. This is because the translator was coded to ignore the other entries in the sample log file. Notice also that the agent only generated an alarm once. This is because the `syslog` cache node was coded to ignore duplicates until 8:00 AM unless they are separated by 1 hour.

Although you used a system log for this tutorial, you can code a translator for application logs as well. In fact, the more unique your application log, the more likely you will need to construct your own custom log monitor.

If you started the agent for this tutorial, you should kill it now.

3 Commands

This section describes commands used with an Audit node.

3.1 Define

The `define` command is used to create an Audit node.

Syntax

auditDefineCmd	::= **define** ̊s *term* ̊s **node** [̊s *auditDef*] ●
auditDef	::= **audit(**"*filename*","*translatorName*",*schedule***)**;
filename	::= name of file to audit
translatorName	::= name of translator file (*.nbx)
schedule	::= *cellExpression*

The *filename* and *translatorName* arguments may be any cell expression that resolves to a string value containing a filename. The value at definition time is used. An audit node will not response to or recognize changes to the value of expressions for these parameters.

The *translatorName* by convention has a ".nbx" suffix. See "Translators" under the `DEFINE` command in the *NodeBrain Language Reference* for instructions on coding translator files.

The schedule parameter may be any cell expression. New lines in the audit file are translated each time the schedule parameter transitions to TRUE. Normally a time condition is used to check the audit file for new lines on a periodic basis (e.g., every 10 seconds) as show in the following example.

```
define logaudit node audit("/var/adm/messages","messages.nbx",~(10s));
```

3.2 Assert

Assertions are not supported by this module.

3.3 Disable

The `disable` commands may be used to stop the node from listening for syslog messages on the specified port.

```
disable node;
```

3.4 Enable

The `enable` command may be used to start listening for syslog messages on the specified port.

```
enable node;
```

An Audit node is automatically enabled when an agent goes into background mode (daemonizes). So the enable command is only required if you want to enable it when running in a different mode, or to re-enable the node after it has been disabled.

3.5 Node Commands

A trace mode can be toggled on or off to assist in debugging translator rules. When trace is on, lines from the file are displayed when processed.

```
node:trace
node:notrace
```

3.6 Module Commands

The Audit module currently implements no module commands.

4 Triggers

All the triggers of an audit node are implemented by the specified translator. Because all commands generated by an audit node's translator are interpreted within the audit node's context, it is necessary to design the rules to work in concert with the translator. If the translator generates `alert` commands, you will want `if` rules. If the translator generates `assert` commands, you will want `on` rules. It is not actually necessary to define rules within the audit node context. As an alternative, the translator may use a context prefix to direct commands to a different context. One common possibility is to define multiple audit nodes as children of a common node and have each audit node generate commands to the parent node. It is also common to use translators that generate commands to a variety of other nodes.

Index