Munin

Table of Contents

Background

Munin is a server/node pair that graphs, htmlifies and optionally sends out notifications about data it gathers. It's designed to let it be easy to graph new datasources.

The Node

munin-node

Munin-node is a perlscript listening to port 4949 using the Net::Server Perl module. It reads all the scripts in /etc/munin/plugins (or CONFDIR/plugins/, to be more spesific) on startup. The node accepts these commands:

```
nodes
      List availbale nodes
list [node]
      list available scripts for [node]
config <script>
      output configuration for <script>
fetch <script>
      output script value <script>
version
      Output version string
quit
      disconnect
```

Scripts

These scripts can be in your language of choice: bash, perl, python, C, or anything else that your system can execute. The scripts can be run in several modes, the important ones being without parameters, and with the "config"-parameter. When run with "config" as parameter, the script should output the configuration of the graph. An example with the "load" graph, which has one field (also called "load"):

```
jo@yes:~$ ./load config
graph_title Load average
graph_args --base 1000 -l 0
graph_vlabel load
graph_scale no
load.label load
load.warning 10
load.critical 120
```

The plugin can output quite a few options:

`graph_title`
: The title of the graph, defaults to the servicename.

`create_args`
: If set, the arguments will be passed on to rrdcreate.

`graph_args`
: If set, the arguments will be passed on to rrdgraph.

`graph_width`
: Sets the width (in pixels) of the graph. Defaults to 400.

`graph_height`
: Sets the height (in pixels) of the graph. Defaults to 175.

`graph_order`
: In witch order to draw the fields. Can also include path aliases on the form alias=domain;host:graph.datasource. See further down for details.

`graph_vlabel`
: Y-axis label of the graph.

`graph_vtitle`
: Y-axis label of the graph. NOTE: Deprecated, use graph_vlabel. If the graph is COUNTER or DERIVE based, the variable `${graph_period}` can be used to access the current scale (second, minute, hour, day).

`graph_info`
: A description of the graph contents.

`graph_total`
: If set, summarise all the datasources' values and use the value of graph_total as a label.

`graph_scale`
: Default on. If set, enables scaling of awg/min/max/cur values.

`graph_period`
: Default "second". Set to "minute" to scale (almost) all graphs that are COUNTER or DERIVE based, to show data per minute instead of per second.

`graph_sums`
: Creates two additional graphs for services using COUNTER or DERIVE fields. The new graphs show values per hour and day. NOTE: This feature requires rrdtool version 1.0.39 or above.

`graph`
: Set to "yes" or "no". Decides wether to draw the graph. Defaults to "yes".

`update`

Set to "yes" or "no". Decides wether munin-update should fetch data for the graph. Defaults to "yes".

`host_name`
: Override which host name this plugin is run for. Ugly - see further down on how to do this in the node configuration files instead, which is more elegant.

`{field}.label`
: REQUIRED. Name of the datasource. You can have many datasouces in one graph.

`{field}.cdef`
: RPN-expression. Modify the values before graphing. See the FAQ for examples.

`{field}.draw`
: What to draw from the data source: AREA, LINE1-3. Defaults to LINE2.

`{field}.graph`
: Set to "no" or "yes. Decides wether to graph the data source. Defaults to yes.

`{field}.max`
: Maximum value. If the fetched value is above "max", it will be discarded.

`{field}.min`
: Minimum value. If the fetched value is below "min", it will be discarded.

`{field}.negative`
: Name of field to 'mirror' on the opposite side of zero. See the FAQ for examples.

`{field}.skipdraw`
: Disables drawing of datasource. NOTE: Deprecated - use {field}.graph instead.

`{field}.info`
: A description of the field.

`{field}.type`
: Type of datasource, COUNTER, ABSOLUTE, DERIVE and GAUGE, defaults to GAUGE. Read "man rrdcreate" for more info.

`{field}.line`
: Draw a line (HRULE) associated with the field. Format is <value>[:colour[:label]]. The default colour is the same as the field colour, or red if it's a single-field graph. Default label is unset.

`{field}.warning`
: Used by munin-nagios. Can be a max value or a range sepereated by colon. E.g. "min:", ":max", "min:max", "max".

`{field}.critical`
: Same as above.

{field} is limited to 19 characters, and the characters [a-zA-Z0-9_]. The fist character cannot be a number.

Without options the script should only give out {name}.value (value):

```
jo@yes:~$ ./load
load.value 0.41
```

All scriptnames containing other characters than alphanumerics, "-", "_", and ".", or starting with "." will be skipped.

To run a plugin as a specific user and/or group, create a file in the plugin configuration dir (default is CONFDIR/plugin-conf.d/). This file is parsed as munin-node starts up. It can contain the following options:

[<plugin-name>]
: The following lines are for `plugin-name`.

`user <username|userid>`
: Run plugin as this user. Only works if munin-node is run as root.

`group <groupname|groupid>[, <groupname|groupid>] [...]`
: Run plugin as this group. If group is inside paranthesis, don't croak if it's nonexistant. Only works if munin-node is run as root.

`command <command>`
: Run command instad of plugin. "%c" will be expanded to what would otherwise have been run. E.g. "`command sudo -u root %c`". Nice to avoid running munin-node as root.

`allow <regex>`
: Allow hosts matching regex to run this plugin.

`deny <regex>`
: Deny hosts matching regex from running this plugin.

`timeout <seconds>`
: Use a timeout of <seconds> seconds instead of the default timeout of 10 seconds, when running this plugin.

`env.<var> <contents>`
: Set the environment variable `var` to `contents` before running the plugin.

Example:

```
[exim_mailstats]
group mail

[cps_*]
user root

# Will cause the variable "mysqlopts" to be set inside the plugins
[mysql_*]
env.mysqlopts --user foo --password fii
```

File locations

According to FHS, this is where you should place the files.

System package (Debian, RedHat, maybe others)

CONFDIR
: /etc/munin/

SBINDIR
: /usr/sbin/

LIBDIR
: /usr/share/munin/

STATEDIR
: /var/run/munin/

LOGDIR
: /var/log/munin/

Independent install (tarball)

CONFDIR
: /etc/opt/munin/

SBINDIR
: /opt/munin/sbin/

LIBDIR

/opt/munin/lib/
STATEDIR
/var/run/munin/
LOGDIR
/var/log/munin/

The Server

The server runs a cronjob as the user munin every 5 minutes. The cronjob runs munin-update,munin-limits,munin-graph and munin-html one by one. All scripts creates a lockfile in @@STATEDIR@@. Everytime a script starts, it checks if the pid in the lockfile is alive before starting.

munin.conf

This is the configuration-file for all serverscripts.

```
#Configfile for munin
dbdir       /var/lib/munin/
htmldir     /var/www/munin/
logdir      /var/log/munin
rundir      /var/run/munin/

#To send email notifications
contact.email.command mail -s "Notification from Munin" fnord@fnord.com
#To notify nagios
contact.nagios.command /usr/bin/send_nsca -H nagios-server.fnord.com -c /etc/send_nsca.cfg

#
# Edit and uncomment the following to start surveilance
#
#[machine.fnord.com]
#  address localhost
```

Explaination:

```
dbdir
      Rootdir for alle rrd-files (files go into <dbdir>/<domain>/)
htmldir
      Where the png's and htmlfiles end up
logdir
      Where to put logs
rundir
      Where to put state files
htaccess
      The default htaccessfile
tmpldir
      Where the templates reside
graph_strategy
      Set to "cron" to draw the graphs periodically via cron every 5
      minutes. Set to "cgi" to draw on-demand. (default cron)
cgiurl
```

```
      URL to the directory where the CGI scripts (for the time being only
      one) doing the graphing (if graph_strategy is "cron"). (default
      /cgi-bin)
cgiurl_graph
      URL to the CGI script doing the graphing (if graph_strategy is
      "cron"). (default ${cgiurl}/munin-cgi-graph)
fork
      If set, run updates of several hosts simultaneously. (default yes)
max_processes
      Set max number of simultaneous Munin processes.
nsca*
      Nagios options. See seperate section. Deprecated, use contacts
      instead.
contact.*
      Set contact information. See separate section.
contacts
      Set which contact entries to use ("none" for no contacts). Default
      is all contact entries existing under "contact" tree.
domain_order
      Change the order of domains. (Default is alphabetically sorted.)
local_address
      Set the local address to be used for connecting to the nodes.
[foo.com;machine.dom.ain]
      Add machine.dom.ain to group foo.com.
[machine.dom.ain]
      Add machine.dom.ain to group dom.ain. (A short form of
      [dom.ain;machine.dom.ain].)
```

To add a new node, just put in a new section and add the `address` option. Group-level options

```
node_order
      Changes order of nodes in a group. (Default is alphabetically
      sorted.)
local_address
      Set the local address to be used for connecting to the nodes in the
      group.
compare
      Generate node comparisons for the nodes in this group.
contacts
      Set which contact entries to use for nodes in this group. Default is
      all contact entries existing under "contact" tree.
```

Node-level options

```
address
      Set the node address
local_address
      Set the local address to be used for connecting to the node.
port
      Set node port number (default 4949)
use_node_name
```

```
      Set to "yes" or "y" to force getting all the default plugins from a
      node. Good for hosts which changes hostname (e.g. laptops).
use_default_name
      Set to "yes" or "y" to force getting all the default plugins from a
      node. Good for hosts which changes hostname (e.g. laptops). NOTE:
      Deprecated. Use use_node_name instaed.
contacts
      Set which contact entries to use for this node. Default is all
      contact entries existing under "contact" tree.
```

Field-level options

```
sum
      Summarise other fields. See the FAQ for how to use this.
stack
      Stack other fields. See the FAQ for how to use this.
+++
      Check the node configuration (further up) for everything else.
```

munin-update

Munin-update reads munin.conf, searches for nodes, and connect to the munin-nodes using the address-field. When connected it will run the list-command to fetch available scripts, then it will run config for each script. This configuration will expand in the `datafile` and rdd-databases will be created. Already expanded configuration will be skipped. Then munin-update runs through it's newly modified configuration file and runs `fetch` on all scripts.

munin-graph

Munin-graph reads /etc/munin/munin.conf and graphs all services unless `[service].graph no`. The following options are available in the configuration

limited to 19 characters

[service].graph_title
: The title of the graph

[service].graph_order
: Which order to graph the lines.

[service].graph_args
: Extra arguments to the graph

[field].label
: REQUIRED, the name of the value to be graphed,

[field].type
: Type of value. COUNTER, GAUGE, defaults to GAUGE. NOTE: When GAUGE is used, only "snapshots" of every 5 minutes are recorded. Peaks in-between updates will not be graphed. When you use COUNTER, the numbers are averaged out over the past 5 minutes, so short peaks will show up as substancially lower than they were.

munin-html

Munin-html creates the html-pages for the graphs.

Usefull configuration in the server.conf file is:

node_order [node1] [node2] ....

In which order the nodes should be listed, defaults to sorted. This is a domain-level option.

domain_order [domain1] [domain2] ....

In which order the domains should be listed, defaults to sorted. This is a top-level option.

category_order_order [category1] [category2] ....

In which order the categories should be listed, defaults to sorted. This is a node-level option.

service_order_order [service1] [service2] ....

In which order the services should be listed, defaults to sorted. This is a node-level option.

munin-limits

Munin-limits is a script to send an alert to a set of contacts. Munin-limits operates with three states -- ok, warning and critical.

The quick and easy way

For most people, the following line will do all the work:

```
contact.email.command mail -s "Munin-notification for ${var:group} :: ${var:host}" your@email.add
```

This entry will use the default `text` entry, which should probably suite most people. If you also use Nagios, try swapping all the `nsca*` parameters for:

```
contact.nagios.command /usr/bin/send_nsca -H your.nagios-host.here -c /etc/send_nsca.cfg
```

Defining contacts

There are some top-level options available in munin.conf:

contact.<contact>.command <command>
: Define which command to run. Mandatory for each contact. The command can start with "> " (greater than space) to create/empty a file and write to it, or ">> " (greater than greater than space) to append to a file.

contact.<contact>.text <text>
: Text to pipe into the command. Default is the text in `contact.default.text`, which is hardcoded (but can be overridden). `contact.nagios.text` also has a short hardcoded default suitable for transmission via nsca to Nagios.

contact.<contact>.max_messages <num>
: Close (and reopen) command after <num> messages.

contact.<contact>.always_send <states>
: Always send messages with a state that is mentioned in <states>. This only works for "warning" and "critical" states. <states> is a space delimited list.

contacts <contact-list>
: A list of the available contacts to use by default. Defaults to all contacts with a `command` definition. Can be set on every level -- top-level, domain-level, node-level and service-level.

Command and text definitions

When defining the `command` and `text` entries, a number of variables are available for expansion.

${var:<variable>}
: For example `${var:graph_title}`. All variables from the plugin are available, as well as the following:

  numofields
  : Number of OKs in the service.

  numfofields
  : Number of new OKs (which were not OK on the last run) in teh service.

  numwfields
  : Number of warnings in the service.

  numcfields
  : Number of criticals in the service.

  ofields
  : Fields in the service with an OK state.

  fofields
  : Fields in the service which just went to OK state.

  wfields
  : Fields in the service with a state of warning.

  cfields
  : Fields in the service with a state of critical.

  fields
  : All fields in the service.

  worst
  : The worst state of all the fields in the plugin.

  worstid
  : 0 for ok, 1 for warning, 2 for dcritical.

  wrange
  : The warning range of the field.

  crange
  : The critical range of the field.

  host
  : The hostname.

  group
  : The name of the group/domain.

  plugin
  : The name of the plugin.

  value
  : Current value of the field.

${if:[!]<field> <text>}
: Include <text> only if <field> is not 0 or a zero length string. Reverse meaning if the "!" is included.

${loop[<sep>]:<list> <text>}
: Print <text> once for each element of the space separated <list>. Separate the <text> additions with the contents of [sep] if it exists (note that the < and > should be included). Example: ${loop<, >:fields ${var:label} is ${var:state}}.

Limit definitions in the plugin or munin.conf

A contact is only contacted if a value falls outside the .warning or .critical fields in your configuration or plugin scripts. The value for these field can be a single maxvalue or a colonseperated range

```
processes.warning 10:300
processes.critical 5:500
```

A value lower than 10 or higher then 300 will result in a warning to nagios, a value lower than 5 or higher than 500 will result in a critical.

Other usefull ranges:

```
[field].warning :400
```

is equal to:

```
[field].warning 400
```

Only warn if lower than 300:

```
[field].warning 300:
```

When a service contains .critical or .warning it will chech it's status agains the last fetched value. Any change in the state (ok, warning, critical) will cause a notification to be sent.

munin-nagios

NOTE: As of version 1.1.5, munin-nagios is replaced by munin-limits. Munin-nagios is a optional script to send a passive alert to a nagios-server. For this to work, you need a nagios-nsca server, a working send_nsca configuration and the following configuration in /etc/munin/munin.conf:

```
nsca          /usr/bin/send_nsca
nsca_config   /etc/nagios/send_nsca.cfg
nsca_server   [nsca-server]
```

Then add .warning and .critical fields in your configuration or directly into you plugin scripts. The value for these field can be a single maxvalue or a colonseperated range

```
processes.warning 10:300
processes.critical 5:500
```

A value lower than 10 or higher then 300 will result in a warning to nagios, a value lower than 5 or higher than 500 will result in a critical to nagios

Other usefull ranges:

```
[service].warning :400
```

is equal to:

```
[service].warning 400
```

Only warn if lower than 300:

```
[service].warning 300:
```

When a service contains .critical or .warning it will chech it's status agains the last fetched value. If it's ok, a "{service}.ok" file will be created in the $dbdir/$domain directory. If the value is not ok. This file will be removed and munin-nagios will update nagios every 5 minutes untill the value is ok and a new ".ok" file will be created.

File locations

According to FHS, this is where you should place the files.

System package (Debian, RedHat, maybe others)

CONFDIR
: /etc/munin/

SBINDIR
: /usr/sbin/

LIBDIR
: /usr/share/munin/

STATEDIR
: /var/run/munin/

LOGDIR
: /var/log/munin/

DBDIR
: /var/lib/munin/

Independent install (tarball)

CONFDIR
: /etc/opt/munin/

SBINDIR
: /opt/munin/sbin/

LIBDIR
: /opt/munin/lib/

STATEDIR
: /var/run/munin/

LOGDIR
: /var/log/munin/

DBDIR
: /var/opt/munin/