Munin Documentation

Release 2.0.73-detached-2023-03-21-c1-g286dfb8f

Stig Sandbeck Mathisen <ssm@fnord.no>

Jun 14, 2023

Contents

Contents:

CHAPTER 1

Munin installation

This document explains how to get Munin onto your system, where to get help, and how to report bugs.

1.1 Prerequisites

In order for you to install Munin you must have the following:

1.1.1 Building munin

In order to build munin, you need:

- GNU Make — Please do not attempt to use any other make.
- A reasonable Perl 5 (Version 5.8 or newer)
- Perl modules: Module::Build

Developers / packagers need

- Test::MockModule
- Test::MockObject
- Test::Pod::Coverage
- Test::Perl::Critic 1.096 or later
- Test::Exception
- Directory::Scratch (err, wherefrom?)

In order to build the documentation, you need: * sphinx

1.1.2 Running munin

In order to run munin, you need:

- A reasonable perl 5 (Version 5.8 or newer)

The munin node needs:

- Perl modules
  - Net::Server
  - Net::Server::Fork
  - Time::HiRes
  - Net::SNMP (Optional, if you want to use SNMP plugins)
- Java JRE (Optional, if you want to use java plugins)
- Anything the separate plugins may need. These have diverse requirements, not documented here.

The munin master needs

- Perl modules:
  - CGI::Fast
  - Digest::MD5,
  - File::Copy::Recursive
  - Getopt::Long
  - HTML::Template
  - IO::Socket::INET6
  - Log::Log4perl 1.18 or later
  - Net::SSLeay (Optional, if you want to use SSL/TLS)
  - Params::Validate
  - Storable
  - Text::Balanced
  - Time::HiRes
  - TimeDate
- A web server capable of CGI or FastCGI

1.2 Installing Munin

With open source software, you can choose to install binary packages or install from source-code. To install a package or install from source is a matter of personal taste. If you don't know which method too choose read the whole document and choose the method you are most comfortable with.

1.2.1 Master and node

Munin is split into two distinct roles.

Node

The "munin node" is a daemon which runs on all servers being monitored.

Master

The “munin master” connects to all munin nodes, collects data, and stores it in RRD

You will need to install “munin-master” on the server which will collect data from all nodes, and graph the results. When starting with munin, it should be enough to install the munin master on one server.

On the munin master, you will need a web server capable of running CGI or FastCGI. Apache HTTD should be suitable. Also reported to be working is nginx and lighttpd.

1.2.2 Source or packages?

Installing Munin on most relevant operating systems can usually be done with with the systems package manager, typical examples being:

FreeBSD

From source:

```
cd /usr/ports/sysutils/munin-master && make install clean
cd /usr/ports/sysutils/munin-node && make install clean
```

Binary packages:

```
pkg_add -r munin-master
pkg_add -r munin-node
```

Debian/Ubuntu

Munin is distributed with both Debian and Ubuntu.

In order to get Munin up and running type

```
sudo apt-get install munin-node
```

on all nodes, and

```
sudo apt-get install munin
```

on the master.

Please note that this might not be the latest version of Munin. On Debian you have the option of enabling “backports”, which may give access to later versions of Munin.

RedHat / CentOS / Fedora

At time of writing, only the 1.x version of munin is available in EPEL.

If you want 2.x, your best option is probably to install from source.

Other systems

On other systems, you are probably best off compiling your own code. See *Installing Munin from source*.

1.2.3 Installing Munin from source

If there are no binary packages available for your system, or if you want to install Munin from source for other reasons, follow these steps:

We recommend downloading a release tarball, which you can find on sourceforge.net.

Alternatively, if you want to hack on Munin, you should clone our git repository by doing.

```
git clone git://github.com/munin-monitoring/munin
```

Please note that a git checkout will need some more build-dependencies than listed below, in particular the Python Docutils and Sphinx.

Build dependencies on Debian / Ubuntu

In order to build Munin from source you need a number of packages installed. On a Debian or Ubuntu system these are:

- perl
- htmldoc
- html2text
- default-jdk

Configuring and installing

Warning for NFS users

If you're using NFS please note that the "make install" process is slightly problematic in that it (Module::Build actually) writes files under $CWD. Since "make install" is usually run by root and root usually cannot write files on a NFS volume, this will fail. If you use NFS please install munin from /var/tmp, /tmp or some such to work around this.

Running make

There are make targets for node, master, documentation and man files. Generally you want to install everything on the master, and just the node and plugiuns on the nodes.

- Edit Makefile.config to suit your needs.
- Create the user "munin" with the primary group "munin".

  The user needs no shell and no privileges. On most Linux systems the munin user's shell is the nologin shell (it has different paths on different systems - but the user still needs to be able to run cron jobs.

Node

For the node, you need only the common parts, the node and the plugins.

```
make
make install-common-prime install-node-prime install-plugins-prime
```

Master

For the master, this will install everything.

```
make
make install
```

1.3 Initial configuration

1.3.1 Node

Plugins

Decide which plugins to use. The munin node runs all plugins present in CONFDIR/plugins/

The quick auto-plug-and-play solution:

```
munin-node-configure --shell --families=contrib,auto | sh -x
```

Access

The munin node listens on all interfaces by default, but has a restrictive access list. You need to add your master's IP address.

The "cidr_allow", "cidr_deny", "allow" and "deny" statements are used.

cidr_allow uses the following syntax (the /32 is not implicit, so for a single host, you need to add it):

> cidr_allow 127.0.0.0/8
> cidr_allow 192.0.2.1/32

allow uses regular expression matching against the client IP address.

> allow '^127.'
> allow '^192.0.2.1$'

For specific information about the syntax, see Net::Server. Please keep in mind that cidr_allow is a recent addition, and may not be available on all systems.

Startup

Start the node agent (as root) SBINDIR/munin-node. Restart it it it was already started. The node only discovers new plugins when it is restarted.

You probably want to use an init-script instead and you might find a good one under build/dists or in the build/resources directory (maybe you need to edit the init script, check the given paths in the script you might use).

1.3.2 Master

Add some nodes

Add some nodes to CONFDIR/munin.conf

[node.example.com] address 192.0.2.4

[node2.example.com] address node2.example.com

[node3.example.com] address 2001:db8::de:caf:bad

1.3.3 Configure web server

On the master, you need to configure a web server.

If you have installed "munin" through distribution packages, a webserver may have been configured for you already.

If you installed from source, there is a minimal configuration example in the "resources" directory in the source tarball.

For a more complex example, see *Apache virtualhost configuration*

1.4 Getting help

1.4.1 IRC Channel

The most immediate way to get hold of us is to join our IRC channel:

```
#munin on server irc.oftc.net
```

The main timezone of the channel is Europe+America.

If you can explain your problem in a few clear sentences, without too much copy&paste, IRC is a good way to try to get help. If you do need to paste log files, configuration snippets, scripts and so on, please use a pastebin.

If the channel is all quiet, try again some time later, we do have lives, families and jobs to deal with also.

You are more than welcome to just hang out, and while we don't mind the occasional intrusion of the real world into the flow, keep it mostly on topic, and do not paste random links unless they are *really* spectacular and intelligent.

1.5 Upgrading Munin from 1.x to 2.x

This is a compilation of items you need to pay attention to when upgrading from Munin 1.x to munin 2.x

1.5.1 FastCGI

Munin graphing is now done with FastCGI.

Munin HTML generation is optionally done with FastCGI.

1.5.2 Logging

The web server needs write access to the munin-cgi-html and munin-cgi-graph logs.

CHAPTER 2

The Munin master

2.1 Role

The munin master is responsible for gathering data from munin nodes. It stores this data in RRD, and graphs them on request.

2.2 Components

The following components are part of munin-master:

- *munin-cron* runs *munin-graph*, *munin-html*, *munin-limits* and *munin-update*.
- *munin-update* is run by *munin-cron*. It is the munin data collector, and it fetches data from *munin nodes*, which is then stored in RRD files.
- *munin-graph* is run by *munin-cron*. It generates graphs in PNG format from the RRD files. See also *munin-cgi-graph*.
- *munin-limits* is run by *munin-cron*. It notifies any configured contacts if a value moves between "ok", "warn" or "crit". Munin is commonly used in combination with Nagios, which is then configured as a contact.
- *munin-html* is run by *munin-cron*. It generates HTML pages. See also *munin-cgi-html*.
- *munin-cgi-graph* is run by a web server. If graph_strategy is set to "cgi", munin-cron will not run munin-graph, and assumes that the web server runs *munin-cgi-graph* instead.
- *munin-cgi-html* is run by a web server. If html_strategy is set to "cgi", munin-cron will not run munin-html, and assumes that the web server runs *munin-cgi-html* instead.

2.3 Configuration

The munin master has its primary configuration file at */etc/munin/munin.conf*.

2.4 Other documentation

2.4.1 Scaling the munin master with rrdcached

When the master grows big, and has a lot of nodes, there is a risk of disk IO becoming a bottleneck.

To reduce this disk IO, you can use the RRD Cache Daemon.

This will spool RRD changes in a queue, and flush changes on demand, and periodically. This will replace lots of random writes with a much smaller amount of sequential writes.

Configuring rrdcached

Parameters

RRDCached writes the spool data every 5 mintes by default. This is the same as the munin master. To have an effect, change the flushing intervals to allow more data to be spooled. Use the following parameters, and tune to your liking:

-w 1800	Wait 30 minutes before writing data
-z 1800	Delay writes by a random factor of up to 30 minutes (this should be equal to, or lower than, "-w")
-f 3600	Flush all data every hour

Example

Create a directory for the rrdcached journal, and have the "munin" user own it. (in this example: /var/lib/munin/rrdcached-journal).

Set up a separate RRDCached instance, run by the munin user. The following command starts an RRDCached instance, and can be added to /etc/rc.local.

```
sudo -u munin /usr/bin/rrdcached \
  -p /run/munin/rrdcached.pid \
  -B -b /var/lib/munin/ \
  -F -j /var/lib/munin/rrdcached-journal/ \
  -m 0660 -l unix:/run/munin/rrdcached.sock \
  -w 1800 -z 1800 -f 3600
```

Note: While testing, add "-g" to the command line to prevent rrdcached from forking into the background.

The munin grapher also needs write access to this socket, in order for it to tell the RRDCached to flush data needed for graphing. If you run munin with CGI graphing, you will need to give the web server access. For a common setup, run the following command, as root, after starting rrdcached:

```
chgrp www-data /run/munin/rrdcached.sock
```

Recommended: If you have systemd installed, use a systemd service. If you have upstart installed, write a daemon job configuration file. If you use systemd, you can add "-g" to the rrdcached command line.

Configuring munin to use rrdcached

To enable rrdcached on the munin master, you will need to set the "rrdcached_socket" line in /etc/munin/munin.conf

```
rrdcached_socket /run/munin/rrdcached.sock
```

Is it working?

If all goes well, you should see the following:

Munin logging

There should be no messages regarding rrdcached in /var/log/munin/munin-update.log.

On failure to connect, there will be log lines like:

```
2012/06/26 18:56:12 [WARN] RRDCached feature ignored: rrdcached socket not writable
```

. . . and you should then check for permissions problems.

RRDCached spool

The rrdcached spool file should be in /var/lib/munin/rrdcached-journal/, and it should grow for each run of munin-update until it hits the flush time. The file looks like:

```
/var/lib/munin/rrdcached-journal/rrd.journal.1340869388.141124
```

For a munin master with 200 nodes, this could well grow to 100MiB, depending on the number of plugins, and the spool file time parameters.

CHAPTER 3

The Munin node

3.1 Role

The munin node is installed on all monitored servers. It accepts connections from the munin master, and runs plugins on demand.

By default, it is started at boot time, listens on port 4949/TCP, accepts connections from the *munin master*, and runs *munin plugins* on demand.

3.2 Configuration

The configuration file is *munin-node.conf*.

3.3 Other documentation

3.3.1 Asynchronous proxy node

The munin asynchronous proxy node (or "munin-async") connects to the local node periodically, and spools the results.

When the munin master connects, all the data is available instantly.

munin-asyncd

The Munin async daemon starts at boot, and connects to the local munin-node periodically, like a *munin master* would. The results are stored the results in a spool, tagged with timestamp.

You can also use munin-asyncd to connect to several munin nodes. You will need to use one spooldir for each node you connect to. This enables you to set up a "fanout" setup, with one privileged node per site, and site-to-site communication being protected by ssh.

munin-async

The Munin async client is invoked by the connecting master, and reads from the munin-async spool using the "spoolfetch" command.

Example configuration

On the munin master

We use ssh encapsulated connections with munin async. In the *the munin master* configuration you need to configure a host with a "ssh://" address.

```
[random.example.org]
  address ssh://munin-async@random.example.org
```

You will need to create an SSH key for the "munin" user, and distribute this to all nodes running munin-asyncd.

The ssh command and options can be customized in *munin.conf* with the ssh_command and ssh_options configuration options.

On the munin node

Configure your munin node to only listen on "127.0.0.1".

You will also need to add the public key of the munin user to the authorized_keys file for this user.

- You must add a "command=" parameter to the key to run the command specified instead of whatever command the connecting user tries to use.

```
command="/usr/share/munin/munin-async --spoolfetch" ssh-rsa AAAA[...] munin@master
```

The following options are recommended for security, but are strictly not necessary for the munin-async connection to work

- You should add a "from=" parameter to the key to restrict where it can be used from.
- You should add hardening options. At the time of writing, these are "no-X11-forwarding", "no-agent-forwarding", "no-port-forwarding", "no-pty" and "no-user-rc".

  Some of these may also be set globally in /etc/ssh/sshd_config.

```
no-port-forwarding,no-X11-forwarding,no-agent-forwarding,no-pty,no-user-rc,from=
↪"192.0.2.0/24",command="/usr/share/munin/munin-async --spoolfetch" ssh-rsa AAAA[.
↪..] munin@master
```

See the sshd_config (5) and authorized_keys(5) man pages for more information.

CHAPTER 4

The Munin plugin

4.1 Role

The munin plugin is a simple executable, which role is to gather one set of facts about the local server.

The plugin is called with the argument "config" to get metadata, and with no arguments to get the values.

4.2 Other documentation

4.2.1 Using munin plugins

Installing

The default plugin directory is /etc/munin/plugins/.

To install a plugin, place it in the plugin directory, and make it executable.

You can also place the plugin elsewhere, and install a symbolic link in the plugin directory. All the plugins provided with munin are installed in this way.

Configuring

The plugin configuration directory is /etc/munin/plugin-conf.d/. The syntax is:

user <username> The user the plugin will run as.

Default: munin

group <groupname> The group the plugin will run as

Default: munin

env.variablename <variable content> Defines and exports an environment variable called "variablename" with the content set to <variable content>.

There is no need to quote the variable content.

Note: When configuring a munin plugin, add the least amount of extra privileges needed to run the plugin. For instance, do not run a plugin with "user root" to read syslogs, when it may be sufficient to set "group adm" instead.

Example:

```
[pluginname]
user              username
group             groupname
env.variablename  some content for the variable
env.critical      92
env.warning       95
```

Plugin configuration is optional.

Testing

To test if the plugin works when executed by munin, you can use the *munin-run* command.

```
# munin-run myplugin config

# munin-run myplugin
```

Download munin plugins

The munin project maintains a set of core plugins that are distributed in munin's releases. Additionally the munin project maintains the contrib repository. It contains more than a thousand plugins contributed by a wide range of people. In order to use these plugins they can either be downloaded manually or managed via the *munin-get* plugin tool.

Additionally the munin plugins in the contrib repository can be browsed via the Munin Plugin Gallery.

4.2.2 Writing a munin plugin

A munin plugin is a small executable. Usually, it is written in some interpreted language.

In its simplest form, when the plugin is executed with the argument "config", it outputs metadata needed for generating the graph. If it is called with no arguments, it outputs the data which is to be collected, and graphed later.

Plugin output

The minimum plugin output when called with "config" it must output the graph title.

It should also output a label for at least one datasource.

```
graph_title Some title for our plugin
something.label Foobar per second
```

When the plugin is executed with no arguments, it should output a value for the datasource labelled in "config". It must not output values for which there are no matching labels in the configuration output.

```
something.value 42
```

For a complete description of the available fields, see the *Plugin reference*.

Example shell plugin

The base of a plugin is a small option parser, ensuring the plugin is called with the correct argument, if any.

Two main functions are defined: One for printing the configuration to the standard output, and one for printing the data. In addition, we have defined a function to generate the data itself, just to keep the plugin readable.

The "output_usage" function is there just to be polite, it serves no other function. :)

```sh
#!/bin/sh

output_config() {
    echo "graph_title Example graph"
    echo "plugins.label Number of plugins"
}

output_values() {
    printf "plugins.value %d\n" $(number_of_plugins)
}

number_of_plugins() {
    find /etc/munin/plugins -type l | wc -l
}

output_usage() {
    printf >&2 "%s - munin plugin to graph an example value\n" ${0##*/}
    printf >&2 "Usage: %s [config]\n" ${0##*/}
}

case $# in
    0)
        output_values
        ;;
    1)
        case $1 in
            config)
                output_config
                ;;
            *)
                output_usage
                exit 1
                ;;
        esac
        ;;
    *)
        output_usage
        exit 1
        ;;
esac
```

Activating the plugin

Place the plugin in the /etc/munin/plugins/ directory, and make it executable.

Then, restart the munin-node.

Debugging the plugin

To see how the plugin works, as the munin node would run it, you can use the command "munin-run".

If the plugin is called "example", you can run "munin-run example config" to see the plugin configuration, and "munin-run example" to see the data.

If you do not get the output you expect, check if your munin plugin needs more privileges. Normally, it is run as the "munin" user, but gathering some data may need more access.

If the munin plugin emits errors, they will be visible in /var/log/munin/munin-node.log

4.2.3 Supersampling

Every monitoring software has a polling rate. It is usually 5 min, because it's the sweet spot that enables frequent updates yet still having a low overhead.

Munin is not different in that respect: it's data fetching routines have to be launched every 5 min, otherwise you'll face data loss. And this 5 min period is deeply grained in the code. So changing it is possible, but very tedious and error prone.

But sometimes we need a very fine sampling rate. Every 10 seconds enables us to track fast changing metrics that would be averaged out otherwise. Changing the whole polling process to cope with a 10s period is very hard on hardware, since now every update has to finish in these 10 seconds.

This triggered an extension in the plugin protocol, commonly known as "supersampling".

Overview

The basic idea is that fine precision should only be for selected plugins only. It also cannot be triggered from the master, since the overhead would be way too big.

So, we just let the plugin sample itself the values at a rate it feels adequate. Then each polling round, the master fetches all the samples since last poll.

This enables various constructions, mostly around "streaming" plugins to achieve highly detailed sampling with a very small overhead.

Notes

This protocol is currently completely transparent to *munin-node*, and therefore it means that it can be used even on older (1.x) nodes. Only a 2.0 *master* is required.

Protocol details

The protocol itself is derived from the spoolfetch extension.

Config

A new plugin directive is used, update_rate. It enables the master to create the rrd with an adequate step.

Omitting it would lead to rrd averaging the supersampled values onto the default 5 min rate. This means **data loss**.

Note: Heartbeat

The heartbeat has always a 2 step size, so failure to send all the samples will result with unknown values, as expected.

Note: Data size

The RRD file size is always the same in the default config, as all the RRA are configured proportionally to the update_rate. This means that, since you'll keep as much data as with the default, you keep it for a shorter time.

Fetch

When spoolfetching, the epoch is also sent in front of the value. Supersampling is then just a matter of sending multiple epoch/value lines, with monotonically increasing epoch.

Note: Note that since the epoch is an integer value for rrdtool, the smallest granularity is 1 second. For the time being, the protocol itself does also mandates integers. We can easily imagine that with another database as backend, an extension could be hacked together.

Compatibility with 1.4

On older 1.4 masters, only the last sampled value gets into the RRD.

Sample implementation

The canonical sample implementation is multicpu1sec, a contrib plugin on github. It is also a so-called streaming plugin.

Streaming plugins

These plugins fork a background process when called that streams a system tool into a spool file. In multicpu1sec, it is the mpstat tool with a period of 1 second.

Undersampling

Some plugins are on the opposite side of the spectrum, as they only need a lower precision.

It makes sense when :

- data should be kept for a *very* long time
- data is *very* expensive to generate and it varies only slowly.

CHAPTER 5

Documenting Munin

This document is rather meta, it explains how to document Munin.

5.1 Nomenclature

To be able to use Munin, to understand the documentation, and - not to be neglected - to be able to write documentation that is consistent with Munin behaviour, we need a common nomenclature.

5.1.1 Common terms

Term	Explanation	Also referred to as as
Munin Master	The central host / server where Munin gathers all data. The machine runs munin-cron	master, server, munin server
Munin Node	The daemon / network service running on each host to be contacted by the	In SNMP terms it may be called an agent.
Plugin	Each munin node handles one or more plugins to monitor stuff on hosts	service
Host	A machine monitored by Munin, maybe by proxy on a munin node, or via a SNMP plugin	
Field	Each plugin presents data from one or more data sources. Each found, read or calculated value corresponds to a field.attribute tuple.	Data source
Attribute	Description found in output from plugins, both general (global) to the plugin, and also specific for each Field.	
Environment variable	Set up by munin node, used to control plugin behaviour. Found in the plugin configuration directory. (/etc/munin/plugin-conf.d/)	
Global (plugin) attribute	Used in the global context in the configuration output from a plugin. (Note: The attribute is considered "global" only to the plugin (and the node), and only when executed.	
Datasource specific plugin attribute	Used in the datasource-specific context in the output of a plugin	
Global directive	Used in munin.conf	
Node level directive	Used in munin.conf	
Group level directive	Used in munin.conf	
Field level directive	Used in munin.conf	

5.1.2 Examples

To shed some light on the nomenclature, consider the examples below:

Global plugin attribute

Global plugin attributes are in the plugins output when run with the config argument. The full list of these attributes is found on the protocol config page. This output does not configure the plugin, it configures the plugins graph.

```
graph_title Load average
----------- ------------
     |           `------ value
     `------------------ attribute
```

Datasource specific plugin attribute

These are found both in the config output of a plugin and in the normal readings of a plugin. A plugin may provide data from one or more data sources. Each data source needs its own set of field.attribute tuples to define how the data source should be presented.

```
load.warning 100
---- ------- ---
  |     |     `- value
  |     `------- one of several attributes used in config output
  `------------- field

load.value 54
---- ----- --
  |    |    `- value
  |    `------ only attribute when getting values from a plugin
  `----------- field
```

Configuration files

This one is from the global section of munin.conf:

```
dbdir       /var/lib/munin/
-----       ---------------
  |                `--------- value
  `-------------------------- global directive
```

And then one from the node level section:

```
[foo.example.org]
  address localhost
  ------- ---------
     |        `----- value
     `-------------- node level directive
```

CHAPTER 6

Reference

This section contains man pages and other reference material

6.1 Man pages

6.1.1 munin-async

DESCRIPTION

The munin async clients reads from a spool directory written by *munin-asyncd*.

It can optionally request a cleanup of this directory.

OPTIONS

--spooldir `| -s <spooldir>`
Directory for spooled data [/var/lib/munin/spool]

--hostname `<hostname>`
Overrides the hostname [The local hostname]

This is used to override the hostname used in the greeting banner. This is used when using munin-async from the munin master, and the data fetched is from another node.

--cleanup
Clean up the spooldir after interactive session completes

--cleanupandexit
Clean up the spooldir and exit (non-interactive)

--spoolfetch
Enables the "spool" capability [no]

--vectorfetch
Enables the "vectorized" fetching capability [no]

Note that without this flag, the "fetch" command is disabled.

--verbose | -v
Be verbose

--help | -h
View this message

EXAMPLES

```
munin-async --spoolfetch
```

This starts an interactive munin node session, enabling the "spoolfetch" command. This does not connect to the local munin node. Everything happens within munin-async, which reads from the spool directory instead of connecting to the node.

SEE ALSO

See also *Asynchronous proxy node* for more information and examples of how to configure munin-async.

6.1.2 munin-asyncd

DESCRIPTION

The munin async daemon connects to a *munin node* periodically, and requests plugin configuration and data.

This is stored in a spool directory, which is read by *munin-async*.

OPTIONS

--spool | -s <spooldir>
Directory for spooled data [/var/lib/munin/spool]

--host <hostname:port>
Connect a munin node running on this host name and port [localhost:4949]

--interval <seconds>
Set default interval size [86400 (one day)]

--retain <count>
Number of interval files to retai [7]

--nocleanup
Disable automated spool dir cleanup

--fork
Fork one thread per plugin available on the node. [no forking]

--verbose | -v
Be verbose

--help | -h
View this message

SEE ALSO

See also *Asynchronous proxy node* for more information and examples of how to configure munin-asyncd.

6.1.3 munin-cgi-graph

DESCRIPTION

The munin-cgi-graph program is intended to be run from a web server. It can either run as CGI, or as FastCGI.

OPTIONS

munin-cgi-graph is controlled using environment variables. See environment variables *PATH_INFO* and *QUERY_STRING*.

Note: The munin-cgi-graph script may be called with the command line options of *munin-graph*. However, the existence of this should not be relied upon.

ENVIRONMENT VARIABLES

The following environment variables are used to control the output of munin-cgi-graph:

PATH_INFO
This is the remaining part of the URI, after the path to the munin-cgi-graph script has been removed.

The group, host, service and timeperiod values are extracted from this variable. The group may be nested.

CGI_DEBUG
If this variable is set, debug information is logged to STDERR, and to /var/log/munin/munin-cgi-graph.log

QUERY_STRING
A list of key=value parameters to control munin-cgi-graph. If QUERY_STRING is set, even to an empty value, a no_cache header is returned.

HTTP_CACHE_CONTROL
If this variable is set, and includes the string "no_cache", a no_cache header is returned.

HTTP_IF_MODIFIED_SINCE
Returns 304 if the graph is not changed since the timestamp in the HTTP_IF_MODIFIED_SINCE variable.

EXAMPLES

When given an URI like the following:

http://munin/munin-cgi/munin-cgi-graph/example.org/client.example.org/cpu-week.png

munin-cgi-graph will be called with the following environment:

PATH_INFO=/example.org/client.example.org/cpu-week.png

To verify that munin is indeed graphing as it should, you can use the following command line:

```
sudo -u www-data \
PATH_INFO=/example.org/client.example.org/irqstats-day.png \
/usr/lib/munin/cgi/munin-cgi-graph | less
```

The "less" is strictly not needed, but is recommended since munin-cgi-graph will output binary data to your terminal.

You can add the *CGI_DEBUG* variable, to get more log information. Content and debug information is logged to STDOUT and STDERR, respectively. If you only want to see the debug information, and not the HTTP headers or the content, you can redirect the file descriptors:

```
sudo -u www-data \
CGI_DEBUG=yes \
PATH_INFO=/example.org/client.example.org/irqstats-day.png \
/usr/lib/munin/cgi/munin-cgi-graph 2>&1 >/dev/null | less
```

6.1.4 munin-cgi-html

DESCRIPTION

The **munin-cgi-html** program is intended to be run from a web server. It can either run as CGI, or as FastCGI.

OPTIONS

munin-cgi-html takes no options. It is controlled using environment variables.

ENVIRONMENT VARIABLES

The following environment variables are used to control the output of munin-cgi-html:

PATH_INFO
This is the remaining part of the URI, after the path to the munin-cgi-html script has been removed.

The group, host, service and timeperiod values are extracted from this variable. The group may be nested.

EXAMPLES

PATH_INFO

"/" refers to the top page.

"/example.com/" refers to the group page for "example.com" hosts.

"/example.com/client.example.com/" refers to the host page for "client.example.com" in the "example.com" group

COMMAND-LINE

When given an URI like the following: http://munin.example.org/munin-cgi/munin-cgi-html/example.org

munin-cgi-html will be called with the following environment:

PATH_INFO=/example.org

To verify that munin is able to create HTML pages, you can use the following command line:

```
sudo -u www-data \
PATH_INFO=/example.org \
/usr/lib/munin/cgi/munin-cgi-html
```

SEE ALSO

munin-cgi-graph.

6.1.5 munin-check

DESCRIPTION

munin-check is a utility that fixes the permissions of the munin directories and files.

Note: munin-check needs superuser rights.

Note: Please don't use this script if you are using 'graph_strategy cgi'. It doesn't care about the right permissions for www-data yet.

OPTIONS

`--fix-permissions` `| -f`
Fix the permissions of the munin files and directories.

`--help` `| -h`
Display usage information

6.1.6 munin-cron

DESCRIPTION

Munin-cron is a part of the package Munin, which is used in combination with *munin-node*.

Munin is a group of programs to gather data from Munin's nodes, graph them, create html-pages, and optionally warn Nagios about any off-limit values.

"munin-cron" runs the following programs, in the given order:

1. *munin-update*
2. *munin-limits*
3. *munin-graph* (unless configured to run from CGI)
4. *munin-html* (unless configured to run from CGI)

Unless the munin master is configured otherwise, "munin-cron" should run every 5 minutes.

OPTIONS

`--service` `<service>`
Limit services to <service>. Multiple –service options may be supplied. [unset]

`--host` `<host>`
Limit hosts to <host>. Multiple –host options may be supplied. [unset]

`--config` `<file>`
Use <file> as configuration file. [/etc/munin/munin.conf]

SEE ALSO

munin-update, *munin-graph*, *munin-limits*, *munin-html*, *munin.conf*,

6.1.7 munin-get

Note: The tool "munin-get" is available since Munin v2.0.52.

Description

The munin plugin helper allows to search, download and use munin plugins from external repositories easily.

A common source of munin plugins is the contrib repository (maintained by the munin project).

Example

Download and enable a plugin (by default: from the contrib repository):

```
munin-get update
munin-get install traffic
munin-get enable traffic
service munin-node restart   # for systemd: systemctl restart munin-node
```

Add a n external repository:

```
munin-get add-repository foo http://example.org/foo.git
munin-get update
munin-get list
```

6.1.8 munin-graph

DESCRIPTION

The munin-graph script is run by munin-cron, and creates graphs from all RRD files in the munin database directory.

OPTIONS

Some options can be negated by prefixing them with "no". Example: –fork and –nofork

--fork
By default munin-graph forks subprocesses for drawing graphs to utilize available cores and I/O bandwidth. Can be negated with –nofork [–fork]

--n `<processes>`
Max number of concurrent processes [6]

--force
Force drawing of graphs that are not usually drawn due to options in the config file. Can be negated with –noforce [–noforce]

--lazy
Only redraw graphs when needed. Can be negated with –nolazy [–lazy]

--help
View this message.

--version
View version information.

--debug
View debug messages.

--cron
Behave as expected when run from cron. (Used internally in Munin.) Can be negated with –nocron

--host <host>
Limit graphed hosts to <host>. Multiple –host options may be supplied.

--only-fqn <FQN>
For internal use with CGI graphing. Graph only a single fully qualified named graph,

For instance: –only-fqn root/Backend/dafnes.example.com/diskstats_iops

Always use with the correct –host option.

--config <file>
Use <file> as configuration file. [/etc/munin/munin.conf]

--list-images
List the filenames of the images created. Can be negated with –nolist-images. [–nolist-images]

--output-file | -o
Output graph file. (used for CGI graphing)

--log-file | -l
Output log file. (used for CGI graphing)

--day
Create day-graphs. Can be negated with –noday. [–day]

--week
Create week-graphs. Can be negated with –noweek. [–week]

--month
Create month-graphs. Can be negated with –nomonth. [–month]

--year
Create year-graphs. Can be negated with –noyear. [–year]

--sumweek
Create summarised week-graphs. Can be negated with –nosumweek. [–summweek]

--sumyear
Create summarised year-graphs. Can be negated with –nosumyear. [–sumyear]

--pinpoint <start,stop>
Create custom-graphs. <start,stop> is the time in the standard unix Epoch format. [not active]

--size_x <pixels>
Sets the X size of the graph in pixels [175]

--size_y <pixels>
Sets the Y size of the graph in pixels [400]

--lower_limit <lim>
Sets the lower limit of the graph

--upper_limit <lim>
Sets the upper limit of the graph

Note: *--pinpoint* and *--only-fqn* must not be combined with any of *--day*, *--week*, *--month* or *--year* (or their negating forms). The result of doing that is undefined.

SEE ALSO

munin-cron, *munin-cgi-graph*

6.1.9 munin-html

DESCRIPTION

munin-html is one of the munin master components run from the *munin-cron* script.

This script is responsible for generating static HTML pages.

If “html_strategy cgi” is set in munin.conf, munin-html will assume HTML pages are generated by munin-cgi-html, and exit silently.

OPTIONS

munin-html has one significant option, which configuration file to use.

Several other options are recognized and ignored as “compatibility options”, since *munin-cron* passes all options through to the underlying components, of which munin-html is one.

--config <file>
    Use <file> as configuration file. [/etc/munin/munin.conf]

--help
    View this message.

--debug
    View debug messages.

--version
    View version information.

--nofork
    Compatibility. No effect.

--service <service>
    Compatibility. No effect.

--host <host>
    Compatibility. No effect.

SEE ALSO

munin-cron, *munin-cgi-html*

6.1.10 munin-limits

DESCRIPTION

munin-limits is one of the processes regularly run from the *munin-cron* script.

It reads the current and the previous collected values for each plugin, and compares them to the plugin’s warning and critical values, if it has any.

If the limits are breached, for instance, if a value moves from “ok” to “warning”, or from “critical” to “ok”, it sends an event to any configured contacts.

A common configured contact is “nagios”, which can use events from munin-limits as a source of passive service check results.

OPTIONS

`--config` `<file>`
Use <file> as configuration file. [/etc/munin/munin.conf]

`--contact` `<contact>`
Limit contacts to those of <contact<gt>. Multiple –contact options may be supplied. [unset]

`--host` `<host>`
Limit hosts to those of <host<gt>. Multiple –host options may be supplied. [unset]

`--service` `<service>`
Limit services to those of <service>. Multiple –service options may be supplied. [unset]

`--always-send` `<severity list>`
Force sending of messages even if you normally wouldn't.

The <severity list> can be a whitespace or comma separated list of the values "ok", "warning", "critical" or "unknown".

This option may be specified several times, to add more values.

Use of "–always-send" overrides the "always_send" value in munin.conf for configured contacts. See also –force.

`--force`
Alias for "–always-send ok,warning,critical,unknown"

`--force-run-as-root`
munin-limits will normally prevent you from running as root. Use this option to override this.

The use of this option is not recommended. You may have to clean up file permissions in order for munin to run normally afterwards.

`--help`
View help message.

`--debug`
If set, view debug messages. Can be negated with –nodebug. [–nodebug]

FILES

/etc/munin/munin.conf

*/var/lib/munin/**

*/var/run/munin/**

SEE ALSO

munin.conf

6.1.11 munin-node

DESCRIPTION

munin-node is a daemon for reporting statistics on system performance.

By default, it is started at boot time, listens on port 4949/TCP, accepts connections from the *munin master*, and runs *munin plugins* on demand.

OPTIONS

--config <configfile>
Use <file> as configuration file. [/etc/munin/munin-node.conf]

--paranoia
Only run plugins owned by root. Check permissions as well. Can be negated with –noparanoia [–noparanoia]

--help
View this help message.

--debug
View debug messages.

Note: This can be very verbose.

--pidebug
Plugin debug. Sets the environment variable `MUNIN_DEBUG` to 1 so that plugins may enable debugging.

CONFIGURATION

The configuration file is *munin-node.conf*.

FILES

/etc/munin/munin-node.conf

*/etc/munin/plugins/**

*/etc/munin/plugin-conf.d/**

/var/run/munin/munin-node.pid

/var/log/munin/munin-node.log

SEE ALSO

munin-node.conf

Example configuration

```
# /etc/munin/munin-node.conf - config-file for munin-node
#

host_name random.example.org
log_level 4
log_file /var/log/munin/munin-node.log
pid_file /var/run/munin/munin-node.pid
background 1
setsid 1

# Which port to bind to;

host [::]
port 4949
user root
group root
```

(continues on next page)

(continued from previous page)

```
# Regexps for files to ignore

ignore_file ~$
ignore_file \.bak$
ignore_file %$
ignore_file \.dpkg-(tmp|new|old|dist)$
ignore_file \.rpm(save|new)$
ignore_file \.puppet-bak$

# Hosts to allow

cidr_allow 127.0.0.0/8
cidr_allow 192.0.2.129/32
```

6.1.12 munin-run

DESCRIPTION

munin-run is a script to run Munin plugins from the command-line.

It is primarily used to debug plugins; munin-run runs these plugins in the same conditions as they are under *munin-node*.

OPTIONS

--config `<configfile>`
    Use <file> as configuration file. [/etc/munin/munin-node.conf]

--servicedir `<dir>`
    Use <dir> as plugin dir. [/etc/munin/plugins/]

--sconfdir `<dir>`
    Use <dir> as plugin configuration dir. [/etc/munin/plugin-conf.d/]

--sconffile `<file>`
    Use <file> as plugin configuration. Overrides sconfdir. [undefined]

--paranoia
    Only run plugins owned by root and check permissions. [disabled]

--help
    View this help message.

--debug
    Print debug messages.

    Debug messages are sent to STDOUT and are prefixed with "#" (this makes it easier for other parts of munin to use munin-run and still have –debug on). Only errors go to STDERR.

--pidebug
    Enable debug output from plugins. Sets the environment variable `MUNIN_DEBUG` to 1 so that plugins may enable debugging. [disabled]

--version
    Show version information.

FILES

/etc/munin/munin-node.conf

*/etc/munin/plugins/**

*/etc/munin/plugin-conf.d/**

/var/run/munin/munin-node.pid

/var/log/munin/munin-node.log

6.1.13 munin-update

DESCRIPTION

munin-update is the primary Munin component. It is run from the *munin-cron* script.

This script is responsible for contacting all the agents (munin-nodes) and collecting their data. Upon fetching the data, munin-update stores everything in RRD files - one RRD files for each field in each plugin.

Running munin-update with the –debug flag will often give plenty of hints on what might be wrong.

munin-update is a component in the Munin server.

OPTIONS

`--config_file` `<file>`
Use <file> as the configuration file. [/etc/munin/munin.conf]

`--debug`
If set, log debug messages. Can be negated with –nodebug [–nodebug]

`--fork`
If set, will fork off one process for each host. Can be negated with –nofork [–fork]

`--host` `<host>`
Limit fetched data to those from <host<gt>. Multiple –host options may be supplied. [unset]

`--service` `<service>`
Limit fetched data to those of <service>. Multiple –service options may be supplied. [unset]

`--timeout` `<seconds>`
Set the network timeout to <seconds>. [180]

`--help`
Print the help message then exit.

`--version`
Print version information then exit.

SEE ALSO

munin-cron

6.1.14 munin.conf

DESCRIPTION

This is the configuration file for the munin master. It is used by *munin-update*, *munin-graph*, *munin-limits*. *munin-html*, *munin-cgi-graph* and *munin-cgi-html*.

GLOBAL DIRECTIVES

Global directives affect all munin master components unless specified otherwise.

dbdir `<path>`
The directory where munin stores its database files. Default: /var/lib/munin

logdir `<path>`
The directory where munin stores its logfiles. Default: /var/log/munin

htmldir `<path>`
The directory where *munin-html* stores generated HTML pages, and where *munin-graph* stores graphs. Default: /var/cache/munin/www

rundir `<path>`
Directory for files tracking munin's current running state. Default: /var/run/munin

tmpldir `<path>`
Directories for templates used by *munin-html* and *munin-cgi-html* to generate HTML pages. Default /etc/munin/templates

fork `<yes|no>`
This directive determines whether *munin-update* fork when gathering information from nodes. Default is "yes".

If you set it to "no" munin-update will collect data from the nodes in sequence. This will take more time, but use less resources. Not recommended unless you have only a handful of nodes.

Affects: *munin-update*

palette `<default|old>`
The palette used by *munin-graph* and *munin-cgi-graph* to colour the graphs. The "default" palette has more colours and better contrast than the "old" palette.

Affects: *munin-graph*

graph_data_size `<normal|huge>`
This directive sets the resolution of the RRD files that are created by *munin-graph* and *munin-cgi-graph*.

Default is "normal".

"huge" saves the complete data with 5 minute resolution for 400 days.

Changing this directive has no effect on existing graphs

Affects: *munin-graph*

graph_strategy `<cgi|cron>`
If set to "cron", *munin-graph* will graph all services on all nodes every run interval.

If set to "cgi", *munin-graph* will do nothing. To generate graphs you must then configure a web server to run *munin-cgi-graph* instead.

Affects: *munin-graph*

html_strategy `<strategy>`
Valid strategies are "cgi" and "cron". Default is "cgi".

If set to "cron", *munin-html* will recreate all html pages every run interval.

If set to "cgi", *munin-html* will do nothing. To generate html pages you must configure a web server to run *munin-cgi-html* instead.

ssh_command `<command>`
The name of the secure shell command to use. Can be fully qualified or looked up in $PATH.

Defaults to "ssh".

ssh_options <options>
The options for the secure shell command.

Defaults are "-o ChallengeResponseAuthentication=no -o StrictHostKeyChecking=no". Please adjust this according to your desired security level.

With the defaults, the master will accept and store the node ssh host keys with the first connection. If a host ever changes its ssh host keys, you will need to manually remove the old host key from the ssh known hosts file. (with: ssh-keygen -R <node-hostname>, as well as ssh-keygen -R <node-ip-address>)

You can remove "StrictHostKeyChecking=no" to increase security, but you will have to manually manage the known hosts file. Do so by running "ssh <node-hostname>" manually as the munin user, for each node, and accept the ssh host keys.

If you would like the master to accept all node host keys, even when they change, use the options "-o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no -o PreferredAuthentications=publickey".

EXAMPLE

A minimal configuration file

```
[client.example.com]
  address client.example.com
```

6.1.15 munin-node.conf

DESCRIPTION

This is the configuration file for *munin-node* and *munin-run*.

The directives "host_name", "paranoia" and "ignore_file" are munin node specific.

All other directives in munin-node.conf are passed through to the Perl module Net::Server. Depending on the version installed, you may have different settings available.

DIRECTIVES

Native

host_name
The hostname used by munin-node to present itself to the munin master. Use this if the local node name differs from the name configured in the munin master.

ignore_file
Files to ignore when locating installed plugins. May be repeated.

paranoia
If set to a true value, *munin-node* will only run plugins owned by root.

Inherited

These are the most common Net::Server options used in *munin-node*.

log_level
Ranges from 0-4. Specifies what level of error will be logged. "0" means no logigng, while "4" means very verbose. These levels correlate to syslog levels as defined by the following key/value pairs. 0=err, 1=warning, 2=notice, 3=info, 4=debug.

Default: 2

`log_file`
Where the munin node logs its activity. If the value is Sys::Syslog, logging is sent to syslog

Default: undef (STDERR)

`port`
The TCP port the munin node listens on

Default: 4949

`pid_file`
The pid file of the process

Default: undef (none)

`background`
To run munin node in background set this to "1". If you want munin-node to run as a foreground process, comment this line out and set "setsid" to "0".

`host`
The IP address the munin node process listens on

Default: * (All interfaces)

`user`
The user munin-node runs as

Default: root

`group`
The group munin-node runs as

Default: root

`setsid`
If set to "1", the server forks after binding to release itself from the command line, and runs the POSIX::setsid() command to daemonize.

Default: undef

`ignore_file`
Files to ignore when locating installed plugins. May be repeated.

`host_name`
The hostname used by munin-node to present itself to the munin master. Use this if the local node name differs from the name configured in the munin master.

`allow`
A regular expression defining which hosts may connect to the munin node.

Note: Use cidr_allow if available.

`cidr_allow`
Allowed hosts given in CIDR notation (192.0.2.1/32). Replaces or complements "allow". Requires the presence of Net::Server, but is not supported by old versions of this module.

`cidr_deny`
Like cidr_allow, but used for denying host access

`timeout`
Number of seconds after the last activity by the master until the node will close the connection.

If plugins take longer to run, this may disconnect the master.

Default: 20 seconds

EXAMPLE

A pretty normal configuration file:

```
host *
port 4949

cidr_allow 127.0.0.0/8
cidr_allow 192.0.2.0/24

user       root
group      root
background 1
setsid     1

log_level 4
log_file  /var/log/munin/munin-node.log
pid_file  /var/run/munin-node.pid

ignore_file \.bak$
ignore_file ^README$
ignore_file \.dpkg-(old|new)$
ignore_file \.rpm(save|new)$
ignore_file \.puppet-new$
```

SEE ALSO

munin-node, *munin-run*

6.2 Other reference material

6.2.1 Directories

dbdir

This directory is used to store the munin master database.

It contains one subdirectory with RRD files per group of hosts, as well as other variable state the munin master would need.

plugindir

This directory contains all the plugins the *munin node* should run.

pluginconfdir

This directory contains plugin configuration.

rundir

This directory contains files needed to track the munin run state. PID files, lock files, and possibly sockets.

logdir

Contains the log files for each munin program.

6.2.2 Plugin reference

Fields

On a configuration run, the plugin is called with the argument "config". The following fields are used.

Field	Value	type	Description	See also	Default
graph_title	string	required	Sets the title of the graph		
graph_args	string	optional	Arguments for the rrd grapher. This is used to control how the generated graph looks, and how values are interpreted or presented.	rrdgraph	
graph_vlabel	string	optional	Label for the vertical axis of the graph		
graph_category	lower case string, no whitespace	optional	Category used to sort the graph on the generated index web page.		misc
graph_info	html text	optional	Additional text for the generated graph web page		
graph_scale	yes|no	optional	If "yes", the generated graph will be scaled to the upper and lower values of the datapoints within the graph.		no
graph_order	space separated list of graph.datapoints	optional	Ensures that the listed datapoints are displayed in order. Any additional datapoints are added in the order of appearance after datapoitns appearing on this list. This field is also used for "borrowing", which is the practice of taking datapoints from other graphs.		
update_rate	integer (seconds)	optional	Sets the update_rate used by the munin master when it creates the RRD file. The update rate is the interval at which the RRD file expects to have data. This field requires a munin master version of at least 2.0.0		
datapoint.label	lower case string, no whitespace	required	The label used in the graph for this field		
datapoint.info	html text	optional	Additional html text for the generated graph web page, used in the field description table		
datapoint.warning	integer, or integer:integer (signed)	optional	This field defines a threshold value or range. If the field value above the defined warning value, or outside the range, the service is considered to be in a "warning" state.		
datapoint.critical	integer, or integer:integer (signed)	optional	This field defines a threshold value or range. If the field value is above the defined critical value, or outside the range, the service is considered to be in a "critical" state.		
datapoint.graph	yes|no	optional	Determines if this datapoint should be visible in the generated graph.		yes
datapoint.cdef	CDEF statement	optional	A CDEF statement is a Reverse Polish Notation statement used to construct a datapoint from other datapoints. This is commonly used to calculate percentages.	cdef-tutorial	
datapoint.draw	AREA, LINE, LINE[n], STACK, AREASTACK, LINESTACK, LINE[n]STACK		Determines how the graph datapoints are displayed in the graph. The "LINE" takes an optional width suffix, commonly "LINE1", "LINE2", etc… The *STACK values are specific to munin and makes the first a LINE, LINE[n] or AREA datasource, and the rest as STACK.	rrdgraph	LINE

On a data fetch run, the plugin is called with no arguments. the following fields are used.

Field	Value	type	Description	See also	Default
datapoint.value	integer, scientific notation, or "U" (may be signed)	required	The value to be graphed.		No default

Example

This is an example of the plugin fields used with the "df" plugin. The "munin-run" command is used to run the plugin from the command line.

Configuration run

```
# munin-run df config
graph_title Filesystem usage (in %)
graph_args --upper-limit 100 -l 0
graph_vlabel %
graph_category disk
graph_info This graph shows disk usage on the machine.
_dev_hda1.label /
_dev_hda1.info / (ext3) -> /dev/hda1
_dev_hda1.warning 92
_dev_hda1.critical 98
```

Data fetch run

```
# munin-run df
_dev_hda1.value 83
```

CHAPTER 7

Examples

Examples of munin and related configuration are gathered here.

7.1 Apache virtualhost configuration

This example describes how to set up munin on a separate apache httpd virtual host. It uses FastCGI if this is available, and falls back to CGI if it is not.

7.1.1 Munin configuration

This example assumes the following configuration in /etc/munin/munin.conf

```
graph_strategy cgi
html_strategy  cgi
```

7.1.2 Virtualhost configuration

Add a new virtualhost, using the following example:

```
<VirtualHost *:80>
    ServerName munin.example.org
    ServerAlias munin

    ServerAdmin  info@example.org

    DocumentRoot /srv/www/munin.example.org

    ErrorLog  /var/log/apache2/munin.example.org-error.log
    CustomLog /var/log/apache2/munin.example.org-access.log combined

    # Rewrites
    RewriteEngine On

    # Static content in /static
    RewriteRule ^/favicon.ico /etc/munin/static/favicon.ico [L]
```

(continues on next page)

(continued from previous page)

```
    RewriteRule ^/static/(.*) /etc/munin/static/$1          [L]

    # HTML
    RewriteCond %{REQUEST_URI} .html$ [or]
    RewriteCond %{REQUEST_URI} =/
    RewriteRule ^/(.*)          /usr/lib/munin/cgi/munin-cgi-html/$1 [L]

    # Images
    RewriteRule ^/munin-cgi/munin-cgi-graph/(.*) /usr/lib/munin/cgi/munin-cgi-
↪graph/$1 [L]

    # Ensure we can run (fast)cgi scripts
    <Directory "/usr/lib/munin/cgi">
        Options +ExecCGI
        <IfModule mod_fcgid.c>
            SetHandler fcgid-script
        </IfModule>
        <IfModule !mod_fcgid.c>
            SetHandler cgi-script
        </IfModule>
    </Directory>
</VirtualHost>
```

7.2 lighttpd configuration

This example describes how to set up munin on lighttpd. It spawns two lighttpd processes, one for the graph rendering, and one for the html generation.

You need to enable the “mod_rewrite” module in the main lighttpd configuration.

7.2.1 Munin configuration

This example assumes the following configuration in /etc/munin/munin.conf

```
# Use cgi rendering for graph and html
graph_strategy cgi
html_strategy cgi
```

7.2.2 Webserver configuration

```
alias.url += ( "/munin-static" => "/etc/munin/static" )
alias.url += ( "/munin"        => "/var/cache/munin/www/" )

fastcgi.server += ("/munin-cgi/munin-cgi-graph" =>
                   (( "socket"      => "/var/run/lighttpd/munin-cgi-graph.sock",
                      "bin-path"    => "/usr/lib/munin/cgi/munin-cgi-graph",
                      "check-local" => "disable",
                   )),
                  "/munin-cgi/munin-cgi-html" =>
                   (( "socket"      => "/var/run/lighttpd/munin-cgi-html.sock",
                      "bin-path"    => "/usr/lib/munin/cgi/munin-cgi-html",
                      "check-local" => "disable",
                   ))
                 )
```

(continues on next page)

(continued from previous page)

```
url.rewrite-repeat-if-not-file += (
                   "/munin/(.*)" => "/munin-cgi/munin-cgi-html/$1",
                   "/munin-cgi/munin-cgi-html$" => "/munin-cgi/munin-cgi-html/",
                   )
```

7.3 nginx configuration

This example describes how to set up munin on nginx.

nginx does not spawn FastCGI processes by itself, but comes with an external "spawn-fcgi" program.

We need one process for the graph rendering, and one for the html generation.

7.3.1 Munin configuration

This example assumes the following configuration in /etc/munin/munin.conf

```
# graph_strategy should be commented out, if present
html_strategy cgi
```

7.3.2 FastCGI configuration

This will spawn two FastCGI processes trees. One for munin cgi graphing and one for HTML generation. It will create a socket owned by www-data, and run the processes as the "munin" user.

```
spawn-fcgi -s /var/run/munin/fastcgi-graph.sock -U www-data \
  -u munin -g munin /usr/lib/munin/cgi/munin-cgi-graph

spawn-fcgi -s /var/run/munin/fastcgi-html.sock  -U www-data \
  -u munin -g munin /usr/lib/munin/cgi/munin-cgi-html
```

Note: Depending on your installation method, the "munin-*-graph" programs may be in another directory. Check Makefile.config if you installed from source, or your package manager if you used that to install.

Note: If you installed using the package manager on Debian or Ubuntu, the /var/log/munin/munin-cgi-*.log files may be owned by the "www-data" user. This example runs the processes as the "munin" user, so you need to chown the log files, and edit /etc/logrotate.d/munin.

7.3.3 Webserver configuration

```
location ^~ /munin-cgi/munin-cgi-graph/ {
    fastcgi_split_path_info ^(/munin-cgi/munin-cgi-graph)(.*);
    fastcgi_param PATH_INFO $fastcgi_path_info;
    fastcgi_pass unix:/var/run/munin/fastcgi-graph.sock;
    include fastcgi_params;
}

location /munin/static/ {
    alias /etc/munin/static/;
}

location /munin/ {
    fastcgi_split_path_info ^(/munin)(.*);
    fastcgi_param PATH_INFO $fastcgi_path_info;
    fastcgi_pass unix:/var/run/munin/fastcgi-html.sock;
```

(continues on next page)

(continued from previous page)

```
    include fastcgi_params;
}
```

7.3.4 Authentication and group access

If you have munin statistics, and need to allow some user (ie: customers) to access only graphs for a subset of nodes, the easiest way might be to use groups, and authentication with the exact same name as the node-group name.

Here is an example of how to redirect the users to the group that matches their name, and prevent any access to other groups. It also has allow an admin user to see it all.

Warning: If you don't want users to get any information about the other group names, you should also change the templates accordingly, and remove any navigation part that might.

```
# Here, the whole vhost has auth requirements.
# You can duplicate it to the graph and html locations if you have
# something else that doesn't need auth.
auth_basic            "Restricted stats";
auth_basic_user_file  /some/path/to/.htpasswd;

location ^~ /cgi-bin/munin-cgi-graph/ {
    # not authenticated => no rewrite (back to auth)
    if ($remote_user ~ ^$) { break; }

   # is on the right subtree ?
    set $ok "no";
    # admin can see it all
    if ($remote_user = 'admin') { set $ok "yes"; }
    # only allow given path
    if ($uri ~ /cgi-bin/munin-cgi-graph/([^/]*)) { set $path $1; }
    if ($path = $remote_user) { set $ok "yes"; }

    # not allowed here ? redirect them where they should land
    if ($ok != "yes") {
        # redirect to where they should be
        rewrite / /cgi-bin/munin-cgi-graph/$remote_user/ redirect;
    }

    fastcgi_split_path_info ^(/cgi-bin/munin-cgi-graph)(.*);
    fastcgi_param PATH_INFO $fastcgi_path_info;
    fastcgi_pass unix:/var/run/munin/fastcgi-graph.sock;
    include fastcgi_params;
}

location /munin/static/ {
    alias /etc/munin/static/;
}

location /munin/ {
    # not authenticated => no rewrite (back to auth)
    if ($remote_user ~ ^$) { break; }

   # is on the right subtree ?
    set $ok "no";
    # admin can see it all
    if ($remote_user = 'admin') { set $ok "yes"; }
    # only allow given path
    if ($uri ~ /munin/([^/]*)) { set $path $1; }
    if ($path = $remote_user) { set $ok "yes"; }
```

(continues on next page)

(continued from previous page)

```
    # not allowed here ? redirect them where they should land
    if ($ok != "yes") {
        # redirect to where they should be
        rewrite / /munin/$remote_user/ redirect;
    }

    fastcgi_split_path_info ^(/munin)(.*);
    fastcgi_param PATH_INFO $fastcgi_path_info;
    fastcgi_pass unix:/var/run/munin/fastcgi-html.sock;
    include fastcgi_params;
}
```

7.4 Graph aggregation by example

This example covers creating aggregate graphs. The configuration reads the current and power from two UPSes (i.e. two hosts with two plugins each) and then creates one virtual host with two virtual plugins; one for current and one for power.

7.4.1 Plugins involved

The example uses a plugin for monitoring UPSes through SNMP, where the UPS address and the different aspects are defined through symlinks. The two UPSes, called "ups-5a" and "ups-5b", are monitored with respect to "current" and "power". Thus, the affected plugins are called as:

```
snmp_ups_ups-5a_current
snmp_ups_ups-5b_current
snmp_ups_ups-5a_power
snmp_ups_ups-5b_power
```

The original plugin name is actually "snmp_ups__" - note the "two" underscores at the end. The plugin is then symlinked to the given host name(s) (e.g. ups-5a) and what we want to monitor (e.g. power). Let's just take one closer look at one of them:

```
snmp_ups_ups-5a_power
-------- ------ -----
    |       |      |
    |       |      `--- The function we want to monitor
    |       `--------- The node name of the UPS
    `----------------- The plugin
```

7.4.2 Extract from munin.conf

The following extract from /etc/munin/munin.conf is explained in detail, step by step, below the configuration.

```
[UPS;ups-5a]
       address 127.0.0.1 # localhost fetches data

[UPS;ups-5b]
       address 127.0.0.1 # localhost fetches data

[UPS;Aggregated]
        update no
        contacts no

```

(continues on next page)

(continued from previous page)

```
11          snmp_ups_current.update no
12          snmp_ups_current.graph_args --base 1000 -l 0
13          snmp_ups_current.graph_category UPS
14          snmp_ups_current.graph_title Aggregated input/output current
15          snmp_ups_current.graph_vlabel Ampere
16          snmp_ups_current.inputtotal.label Input current
17          snmp_ups_current.outputtotal.label Output current
18          snmp_ups_current.graph_order inputtotal outputtotal
19          snmp_ups_current.inputtotal.sum \
20                          ups-5a:snmp_ups_ups-5a_current.inputcurrent \
21                          ups-5b:snmp_ups_ups-5b_current.inputcurrent
22          snmp_ups_current.outputtotal.sum \
23                          ups-5a:snmp_ups_ups-5a_current.outputcurrent \
24                          ups-5b:snmp_ups_ups-5b_current.outputcurrent
25
26          snmp_ups_power.update no
27          snmp_ups_power.graph_args --base 1000 -l 0
28          snmp_ups_power.graph_category UPS
29          snmp_ups_power.graph_title Aggregated output power
30          snmp_ups_power.graph_vlabel Watts
31          snmp_ups_power.output.label Output power
32          snmp_ups_power.graph_order output
33          snmp_ups_power.output.sum \
34                          ups-5a:snmp_ups_ups-5a_power.outputpower \
35                          ups-5b:snmp_ups_ups-5b_power.outputpower
```

7.4.3 Explanations, per line

- 1 - 2: The SNMP-based plugin for the UPS known as "ups-5a" is defined. The group name is "UPS" and the node name is "ups-5a". The plugin is run from localhost.
- 4 - 5: The SNMP-based plugin for the UPS known as "ups-5b" is defined. The group name is "UPS" and the node name is "ups-5b". The plugin is run from localhost.
- 7: The group and "virtual node name" for the aggregated graphs are defined. The group name is "UPS" and the virtual node name is "Aggregated".
- 8: Make sure that Munin (specifically, "munin-update") does not try to actively gather information for this node.
- 9: Tell "munin-limits" not to send alerts if any limit is breached.

The above lines (1 - 9) have now established the fundament for three different graph pages; one for each of the two UPSes and one for the aggregate graphs.

- 11 - 15: Define the basic information for the virtual plugin for aggregated current. Note that "snmp_ups_current" is the virtual plugin's name.
- 16 - 17: Simultaneously define and label "two" values to be graphed in the virtual plugin: "inputtotal" and "outputtotal".
- 18: Order the values.
- 19 - 21: Calculate the value for "inputtotal" by reading the "inputcurrent" values from each of the two UPSes.

Let's take a closer look at the components

```
snmp_ups_current.inputtotal.sum \
---------------- ---------- ---
        |             |       |
        |             |       `-- The sum mechanism
```

(continues on next page)

(continued from previous page)

```
        |            `--------- One of this virtual plugin's values
        `----------------------- The name of the virtual plugin
```

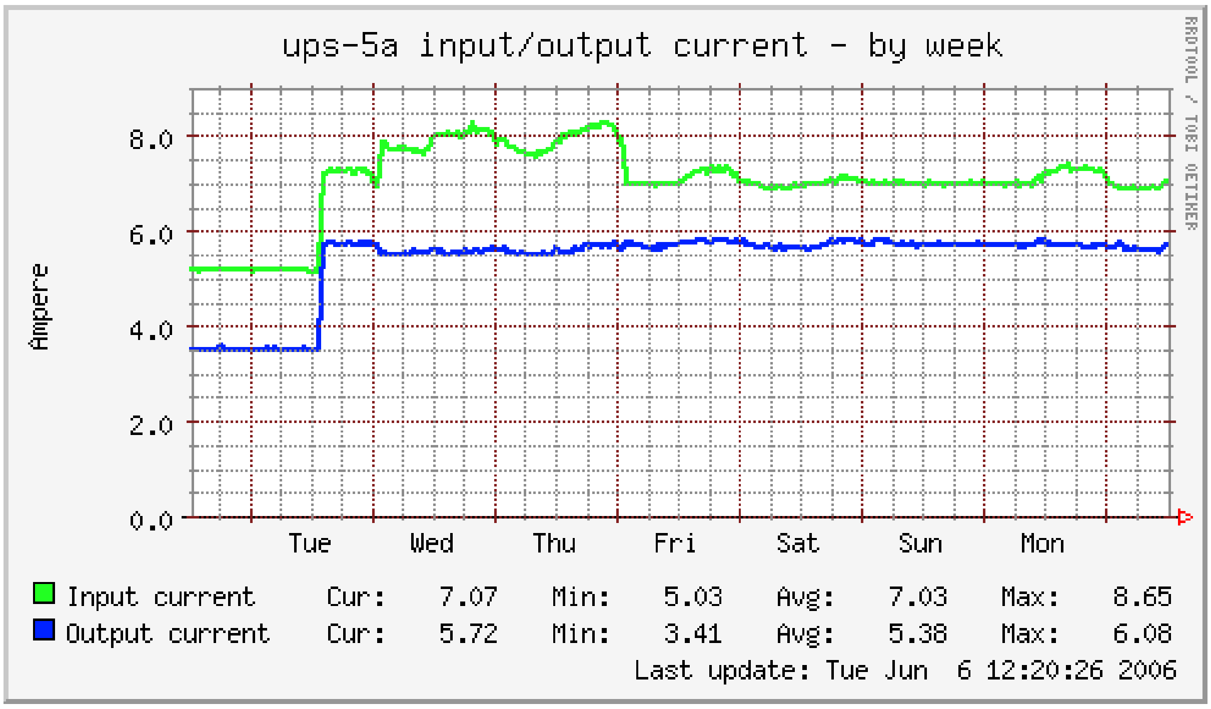
```
ups-5a:snmp_ups_ups-5a_current.inputcurrent \
ups-5b:snmp_ups_ups-5b_current.inputcurrent
------ ----------------------- ------------
   |              |                 |
   |              |                 `------ The "inputcurrent" value from the␣
↪real plugin
   |              `------------------------ The real plugin's name (symlink)
   `--------------------------------------- The host name from which to seek␣
↪information
```

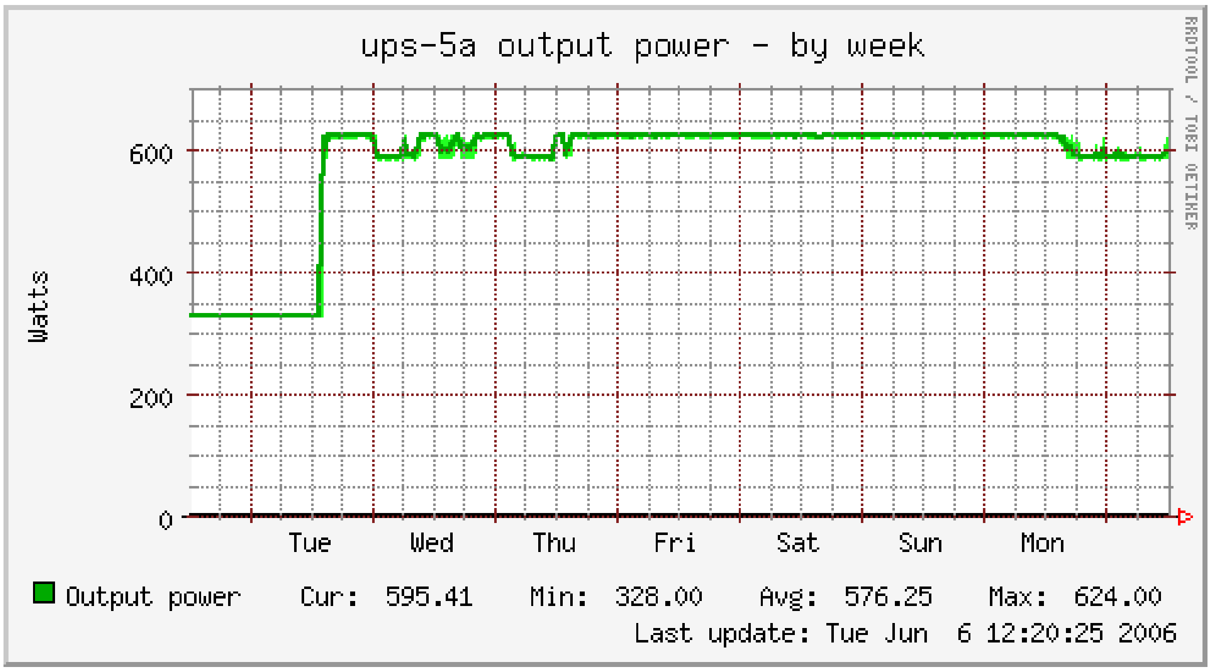
- 22 - 24: Similarly for “outputtotal”.
- 26 - 35: Like the above, but for power instead. Note that this virtual plugin graphs only “one” value, and as such, only “one” “sum” mechanism is used.

7.4.4 Result graphs

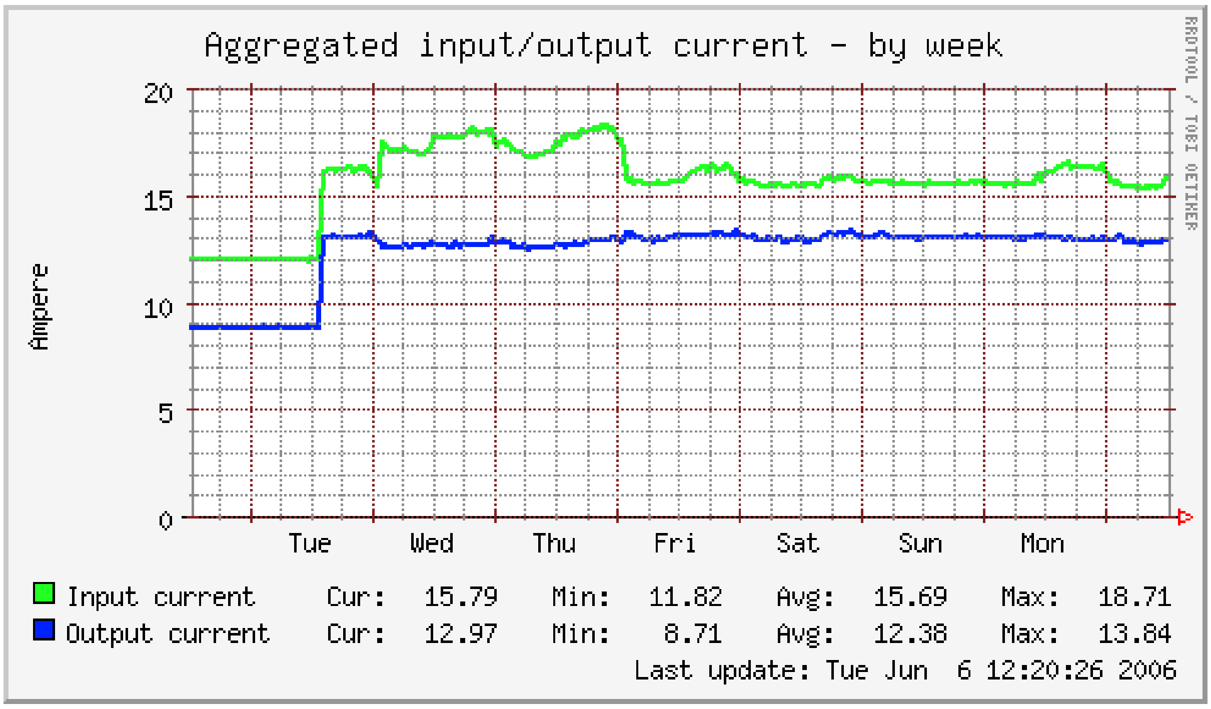
The graphs below show one of the UPSes, and the aggregated values. The graphs used are by week, because they had a nice dip in the beginning of the graphing period :-)

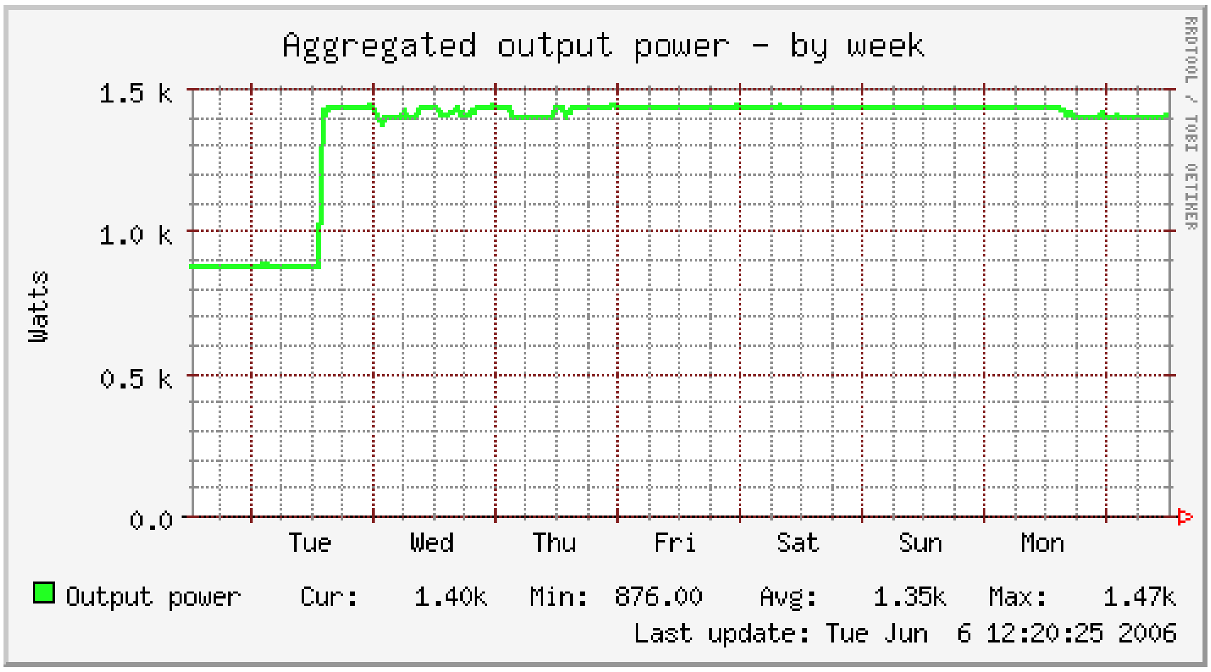
Source graphs for one of the UPSes:





Aggregate graphs:





7.4.5 Summary

We have now, in addition to the two real UPS nodes "ups-5a" and "ups-5b" (lines 1 - 5), created one virtual host named "Aggregated" (line 7) with two virtual plugins: "snmp_ups_current" (lines 11 - 24) and "snmp_ups_power" (lines 26 - 35).

The "snmp_ups_current" virtual plugin outputs two field names: "inputtotal" (lines 16 and 19 - 21) and "outputtotal" (lines 17 and 22 - 24), while the "snmp_ups_power" virtual plugin outputs only one field name, namely "output" (lines 31 - 35).

7.4.6 Further reading

- [wiki:Using_SNMP_plugins Using SNMP plugins]
- [wiki:munin.conf munin.conf] directives explained

7.5 multiple master data aggregation

This example describes a way to have multiple master collecting different information, and show all the data in a single presentation.

When you reach some size (probably several hundreds of nodes, several tousands plugins), 5 minutes is not enough for your single master to connect and gather data from all hosts, and you end up having holes in your graph.

7.5.1 Requirements

This example requires a shared nfs space for the munin data between the nodes.

Before going that road, you should make sure to check other options first, like changing the number of update threads, and having rrdcached.

An other option you might consider, is using munin-async. It requires modifications on all nodes, so it might not be an option, but I felt compeled to mention it. If you can't easily have shared nfs, or if you might have connectivity issues between master and some node, async would probably be a better approach.

Because there is some rrd path merge required, it is highly recommended to have **all** nodes in groups.

7.5.2 Overview

Munin-Master runs different scripts via the cron script (munin-cron).

`munin-update` is the only part actually connecting to the nodes. It gathers information and updates the rrd (you'll probably need rrdcached, especially via nfs).

`munin-limits` checks what was collected, compared to the limits and places warning and criticals.

`munin-html` takes the information gathered by update and limits, and generates the actual html files (if don't have cgi-html). It currently still generates some data needed by the cgi.

`munin-graph` generate the graphs. If you are thinking about getting many masters, you probably have a lot of graph, and don't want to generate them every 5 minutes, but you would rather use cgi-graph.

The trick about having multiple master running to update is :

- run `munin-update` on different masters (called update-masters there after), having `dbdir` on nfs
- run `munin-limits` on either each of the update-masters, or the html-master (see next line)
- run `munin-html` on a single master (html-master), after merging some data generated by the update processes
- have graph (cgi) and html (from file or cgi) served by either html-master, or specific presentation hosts.

Of course, all hosts must have access to the shared nfs directory.

Exemples will consider the shared folder /nfs/munin.

7.5.3 Running munin-update

Change the `munin-cron` to only run `munin-update` (and `munin-limits`, if you have alerts you want to be managed directly on those masters). The cron should NOT launch munin-html or munin-graph.

Change your `munin.conf` to use a dbdir within the shared nfs, (ie: `/nfs/munin/db/<hostname>`).

To make it easier to see the configuration, you can also update the configuration with an `includedir` on nfs, and declare all your nodes there (ie: `/nfs/munin/etc/<hostname>.d/`).

If you configured at least one node, you should have `/nfs/munin/db/<hostname>` that starts getting populated with subdirectories (groups), and a few files, including `datafile`, and `datafile.storable` (and `limits` if you also have munin-limits running here).

7.5.4 Merging data

All our update-masters generate update their dbdir including:

- `datafile` and `datafile.storable` which contain information about the collected plugins, and graphs to generate.
- directory tree with the rrd files

In order to have munin-html to run correctly, we need to merge those dbdir into one.

Merging files

`datafile` is just plain text with lines of `key value`, so concatenating all the files is enough.

`datafile.storable` is a binary representation of the data as loaded by munin. It requires some munin internal structures knowledge to merge them.

If you have `munin-limits` also running on update-masters, it generate a `limits` files, those are also plain text.

In order to make that part easier, a `munin-mergedb.pl` is provided in contrib.

Merging rrd tree

The main trick is about rrd. As we are using a shared nfs, we can use symlinks to get them to point to one an other, and not have to duplicate them. (Would be hell to keep in sync, that's why we really need shared nfs storage.)

As we deal with groups, we could just link top level groups to a common rrd tree.

Example, if you have two updaters (update1 and update2), and 4 groups (customer1, customer2, customer3, customer4), you could make something like that:

```
/nfs/munin/db/shared-rrd/customer1/
/nfs/munin/db/shared-rrd/customer2/
/nfs/munin/db/shared-rrd/customer3/
/nfs/munin/db/shared-rrd/customer4/
/nfs/munin/db/update1/customer1 -> ../shared-rrd/customer1
/nfs/munin/db/update1/customer2 -> ../shared-rrd/customer2
/nfs/munin/db/update1/customer3 -> ../shared-rrd/customer3
/nfs/munin/db/update1/customer4 -> ../shared-rrd/customer4
/nfs/munin/db/update2/customer1 -> ../shared-rrd/customer1
/nfs/munin/db/update2/customer2 -> ../shared-rrd/customer2
/nfs/munin/db/update2/customer3 -> ../shared-rrd/customer3
/nfs/munin/db/update2/customer4 -> ../shared-rrd/customer4
/nfs/munin/db/html/customer1 -> ../shared-rrd/customer1
/nfs/munin/db/html/customer2 -> ../shared-rrd/customer2
/nfs/munin/db/html/customer3 -> ../shared-rrd/customer3
/nfs/munin/db/html/customer4 -> ../shared-rrd/customer4
```

At some point, an option to get the rrd tree separated from the dbdir, and should avoid the need of such links.

7.5.5 Running munin-html

Once you have your update-masters running, and a merge ready to go, you should place a cron on a html-master to :

- merge data as requested
- launch `munin-limits`, if not launched on update-masters and merged
- launch `munin-html` (required, even if you use cgi)
- launch `munin-graph` unless you use cgi-graph

CHAPTER 8

Indices and tables

- genindex
- search

Index