



INSTANT
Short | Fast | Focused

Munin Plugin Starter

Write custom scripts to monitor, analyse, and optimize any device in your network

Bart ten Brinke

[PACKT]
PUBLISHING

www.allitebooks.com

Instant Munin Plugin Starter

Write custom scripts to monitor, analyze, and optimize any device in your network

Bart ten Brinke



BIRMINGHAM - MUMBAI

Instant Munin Plugin Starter

Copyright © 2013 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: February 2013

Production Reference: 1070213

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-84969-674-6

www.packtpub.com

Credits

Author

Bart ten Brinke

Project Coordinator

Michelle Quadros

Reviewer

Diego Elio Pettenò

Proofreader

Aaron Nash

Acquisition Editor

Martin Bell

Graphics

Aditi Gajjar

Commissioning Editor

Harsha Bharwani

Production Coordinator

Prachali Bhiwandkar

Technical Editor

Pooja Prakashan

Cover Work

Prachali Bhiwandkar

Copy Editors

Brandt D'Mello

Insiya Morbiwala

Cover Image

Sheetal Aute

About the Author

Bart ten Brinke is an experienced product developer; he has been building web applications for over six years, mainly focusing on Ruby on Rails. In 2012, he started his own company called Retrosync. Retrosync focuses on the three major pillars of web application development—security, scalability, and usability.

Prior to Retrosync, Bart worked at Nedap Healthcare as a product developer and security officer. Here he developed a web-based planning solution that is now used by a majority of the Dutch home care sector.

Bart holds a Masters title in Information Technology and a Minor in Biomedical Engineering, both from the University of Twente in the Netherlands. He is also a Certified Information Systems Security Professional (CISSP).

I would like to take this opportunity to thank Olaf van Zandwijk, Daniele Sluijters, Steve Schnepf, and Diego Elio Pettenò for taking the time to review this book. I would especially like to thank Nanda van de Weerd for always having faith in me.

About the Reviewer

Diego Elio Pettenò is a free software developer involved in projects based on Munin, Libav, and Gentoo Linux. He's been maintaining a general, but mostly technical, blog for the past 8 years and has contributed articles to www.linux.com and www.lwn.net. His specialties are ELF and Autotools.

www.packtpub.com

Support files, eBooks, discount offers and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

packtLib.packtpub.com

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- ◆ Fully searchable across every book published by Packt
- ◆ Copy and paste, print and bookmark content
- ◆ On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.



Table of Contents

Instant Munin Plugin Starter	1
So, what is Munin?	3
Installation	6
Step 1 – What do I need?	6
Step 2 – Installing Munin	6
Step 3 – Plotting graphs	7
And that's it	8
Quick start – Setting up Munin	10
Munin master configuration	10
Example configuration file for Munin	10
Munin node configuration	13
Example configuration file for munin-node	13
Managing plugins	15
Top 6 features you need to know about	17
Monitoring additional servers	17
Step 1 – Installing munin-node	17
Step 2 – Testing your munin-node installation	18
Step 3 – Installing additional plugins	19
Step 4 – Adding the new node to the master	20
Troubleshooting	21
Monitoring additional devices	22
Step 1 – Enabling the SNMP interface plugin	22
Step 2 – Testing the SNMP interface plugin	22
Step 3 – Configuring the Munin master	24
Installing sensor plugins	24
Monitoring sensors through lm-sensors	24
Monitoring sensors through IPMI	28
Writing a simple plugin	31
Step 1 – Gathering the information we need	31
Step 2 – Designing the graph	32
Step 3 – Writing the plugin	33

Table of Contents

Step 4 – Testing the plugin	33
Challenge yourself!	34
Writing a complex plugin	34
Step 1 – Rewriting our bash plugin to Perl	35
Step 2 – Documentation and markers	37
Step 3 – Supporting the autoconf argument	40
Step 4 – Adding support for the suggest argument	41
Advanced plugin options	41
dirtyconfig	41
Multigraph	43
Challenge yourself!	45
People and places you should get to know	46
Finding plugins	46
Windows support	46
Interesting websites	47

Instant Munin Plugin Starter

Welcome to the *Instant Munin Plugin Starter*. This book provides you with all the information that you need to set up a Munin cluster and expand it. We will also teach you how to write your own Munin plugins.

This book contains the following sections:

So, what is Munin? explains what Munin actually is, what you can do with it, and why it's so great.

Installation tells you how to install Munin with minimum fuss so that you can use it as soon as possible.

Quick start – Setting up Munin will walk you through all the important bits for the configuration of the Munin master, Munin node, and Munin plugin.

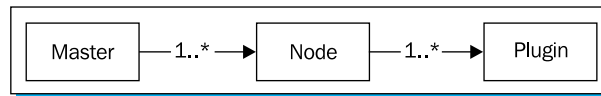
Top 6 features you need to know about explains how to monitor additional servers and devices. You will also learn how to write a Munin plugin using bash and Perl.

People and places you should get to know provides you with many useful links to websites and Twitter feeds where you can find more interesting information about Munin.

So, what is Munin?

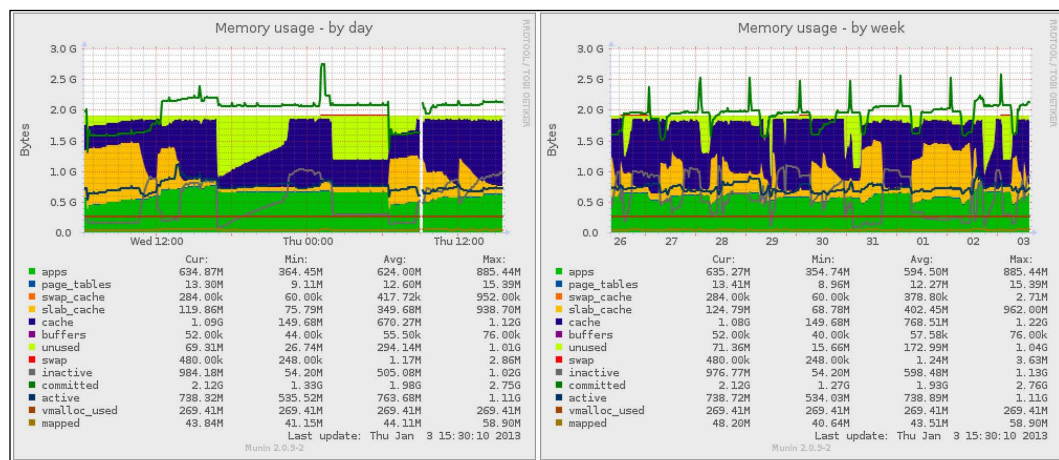
Munin is a set of tools that allows you to monitor computers, applications, and devices in a networked environment. Munin itself actually consists of three main components:

- ◆ Munin master – the server component
- ◆ Munin node – the client component
- ◆ Munin plugin – the program that collects data



The master typically connects to all its client nodes every 5 minutes and asks them for a status update. The node then looks at its configured plugins and calls them to produce the requested status information.

For example, a plugin might return the amount of free memory available in the system or the amount of free disk space remaining. The nodes send this data back to the master, which uses it to plot graphs of all the information provided. These graphs can easily be viewed using a web browser.



Graphs are created such that they display the information for the past 24 hours, week, month, and so on, up to the past year.

The real power of Munin is in the fact that this system of information gathering and graph plotting is set up in a very generic way. This makes a Munin setup easily extendable. When you configure a plugin to monitor something on a node, the Munin server will automatically start drawing neat graphs for you. The great thing is that while there are a lot of plugins provided with Munin, you can also find a lot of plugins on the Internet.

If you cannot find the exact plugin you need, you can write it yourself; it is very easy to do so. Further on, we will explain how you can write your own plugin from scratch.

So you might ask yourself why you would want to measure all this data. You might even already have some alerting services, such as Nagios or Icinga set up to warn you when a host is down or a service in your network is not responding. Even though Munin can actually send out alerts as well, it is definitely not its main focus. Munin is a very powerful tool to use when you are debugging complex performance problems. Munin can really help you out of a tight spot, especially if you have a problem spanning multiple servers.

When a performance issue arises, the problem is usually very simple to pinpoint, such as maybe your web server isn't handling as many requests per second as it should or your database server isn't processing as many queries as it used to.

The real challenge comes from trying to get to the root of a problem, say to find out what is causing this sudden drop in your performance. It is only when you find the root cause of the problem that you can actually start working on fixing it.

As Munin plots a lot of graphs for you about your systems, it makes it a lot easier for you to visually correlate the things that might be causing your problem. If you don't have this information at the ready when you encounter a problem, you would have to resort to frantically taking stabs in the dark.

Another great advantage of Munin is that the graphs help you with resource and capacity planning. For example, if you see the disk usage graph increase by 1 percent every day, you can be sure that you will have a big problem somewhere within the next 3 months.



As Munin is so flexible and easy to set up, it is very easy for an administrator to set up a custom dashboard of his or her network to help tackle any problems that may arise in their day-to-day business. Munin can monitor anything from web requests, mail queues, print queues, and network statistics to temperature, CPU load, the usage of the coffee machine, and everything in between.

Installation

In four easy steps, you can install Munin and have it up and running and monitoring your system.

Step 1 – What do I need?

Before you install Munin, you will need to check that you have all of the required elements listed as follows:

- ◆ **Disk space:** Munin requires 100 MB of free disk space. You will require more free space if you are going to monitor lots of devices.
- ◆ **Memory:** The minimum memory required is 128 MB, but 1 GB is recommended.
- ◆ **Munin master:** This requires a web server environment, such as Apache, and needs Perl 5 and RRDtool installed to function correctly.
- ◆ **Munin node:** This should run in any environment that runs Perl 5.
- ◆ **Munin plugins:** These can be platform-specific, but most should work on any Linux based system.

The load impact on the nodes is typically very low. This might increase if you install a lot of plugins that perform heavy operations, such as log file analysis. This is something to look out for.

The load impact of the master is also very low with small clusters. If your cluster gets bigger, you should look into the fine-tuning of the graph plotting features as this is the most load-intensive work that the master does. When this happens, you should look at plotting graphs only when viewed or assigning more resources to the graphing processes.

Step 2 – Installing Munin

The easiest way to install Munin is to use the package manager provided by your operating system. This will ensure that you get the most current stable build.

For Debian or Ubuntu, enter the following command:

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

For Red Hat or Fedora, use:

```
sudo yum -y install munin munin-node
```

If you are trying to install Munin for another operating system, such as FreeBSD or OS X, please check whether they provide a native prepackaged Munin. If not, take a look at the wiki on the following Munin website for information on how to install Munin on your system:

<http://munin-monitoring.org/wiki/Documentation>

It is also possible to monitor Windows systems, but we will not go into that here.

Step 3 – Plotting graphs

In order to view the generated graphs, we need to configure a web server to actually serve the HTML generated by the master and munin-graph. If you don't have a web server installed, please install Apache2 and add the Munin host to the configuration.

For Debian or Ubuntu, use the following commands:

```
sudo apt-get install apache2
sudo ln -s /etc/munin/apache.conf /etc/apache2/sites-enabled/munin
sudo apache2ctl restart
```

For Red Hat or Fedora, use:

```
sudo yum install httpd
sudo ln -s /etc/munin/apache.conf /etc/httpd/conf.d/munin.conf
sudo apachectl restart
```

This will install Apache; symlink the Apache configuration provided by Munin to Apache, and then restart Apache to enable the new website.

By default, the Munin website is only available from localhost. If you want to change this, edit `/etc/munin/apache.conf` and add your IP address to the `Allow from` list. For example, if your IP address is `10.0.0.123`, change the configuration to look like this:

```
<Directory /var/cache/munin/www>
    Order allow,deny
    Allow from localhost 127.0.0.0/8 ::1 10.0.0.123
    Options None

    <IfModule mod_expires.c>
        ExpiresActive On
        ExpiresDefault M310
    </IfModule>

</Directory>
```

Please note that the list is space-separated and that you have to restart Apache before your changes take effect.

If you prefer to use basic authorization for authentication, you can also configure this here.




Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.PacktPub.com>. If you purchased this book elsewhere, you can visit <http://www.PacktPub.com/support> and register to have the files e-mailed directly to you.

And that's it

By this point, you should have a working installation of Munin running at <http://localhost/munin> or at http://your_munin_master/munin.

 **MUNIN** Overview

Problems Critical (0) Warning (0) Unknown (0)	<ul style="list-style-type: none">• localdomain<ul style="list-style-type: none">◦ localhost.localdomain [disk munin network processes system time]
Groups localdomain	
Categories disk [d w m y] munin [d w m y] network [d w m y] processes [d w m y] system [d w m y] time [d w m y]	

This page was generated by Munin version 2.0.9-2 at 2013-01-03 09:50:06+0100 (CET).

Go ahead and look around to discover more about Munin. Keep in mind that Munin generates a point in each graph every 5 minutes, so right now, your graphs will be pretty boring. Go and do something else for about an hour, and then come back and take a look at all the beautiful graphs that were generated for you!



Quick start – Setting up Munin

In this section, you'll learn the most important settings for Munin master and node and how to manage Munin plugins.

Munin master configuration

The configuration of the Munin master is located at `/etc/munin/munin.conf`. Use your favorite text editor to open that file and look at its contents. The configuration is pretty well documented, but we will walk you through it nonetheless.

Example configuration file for Munin

An example configuration file for Munin is given as follows:

```
# Example configuration file for Munin, generated by 'make build'

# The next three variables specify the location of the RRD
# databases, the HTML output, logs and the lock/pid files.  They
# all must be writable by the user running munin-cron.  They are all
# defaulted to the values you see here.
#
#dbdir      /var/lib/munin
#htmldir    /var/cache/munin/www
#logdir     /var/log/munin
#rundir     /var/run/munin

# Where to look for the HTML templates
#
#tmpdir     /etc/munin/templates

# Where to look for the static www files
#
#staticdir  /etc/munin/static

# temporary cgi files are here. note that they have to be writable by
# the cgi user (usually nobody or httpd).
#
#cgtmpdir   /var/lib/munin/cgi-tmp

# (Exactly one) directory to include all files from.
includedir  /etc/munin/munin-conf.d
```

```
# You can choose the time reference for "DERIVE" like graphs and
# show "per minute", "per hour" values instead of the default
#"per second" #graph_period second

# Graphics files are generated either via cron or by a CGI
#process.
# See http://munin-monitoring.org/wiki/CgiHowto2 for more
# documentation.
# Since 2.0, munin-graph has been rewritten to use the cgi code.
# It is single threaded *by design* now.
#
#graph_strategy cron

# munin-cgi-graph is invoked by the web server up to very many
# times at the same time.
# This is not optimal since it results in high CPU
# and memory consumption to the degree that the system can thrash.
# Again the default is 6. Most likely the optimal number for
# max_cgi_graph_jobs is the same as max_graph_jobs.
#
#munin_cgi_graph_jobs 6

# If the automatic CGI url is wrong for your system override
# it here:
#
#cgiurl_graph /munin-cgi/munin-cgi-graph

# max_size_x and max_size_y are the max size of images in pixel.
# Default is 4000. Do not make it too large otherwise RRD might
# use all RAM to generate the images.
#
#max_size_x 4000
#max_size_y 4000

# HTML files are normally generated by munin-html, no matter if
# the files are used or not. You can change this to on-demand
#generation
# by following the instructions in
# http://munin-monitoring.org/wiki/CgiHowto2
#
# Notes:
# - moving to CGI for HTML means you cannot have graphs generated
#   by cron.
# - cgi html has some bugs, mostly you still have to launch
```

Instant Munin Plugin Starter

```
# munin-html by hand
#
#html_strategy cron

# munin-update runs in parallel.
#
# The default max number of processes is 16, and is probably ok.
#
# If set too high, it might hit some process/ram/filedesc limits.
# If set too low, munin-update might take more than 5 min.
#
# If you want munin-update to not be parallel set it to 0.
#
#max_processes 16

# RRD updates are per default, performed directly on the rrd
# files.
# To reduce IO and enable the use of the rrdcached, uncomment it
# and set it to the location of the socket that rrdcached uses.
#
#rrdcached_socket /var/run/rrdcached.sock

# Drop somejuser@fnord.comm and anotheruser@blibb.comm an email
# everytime something changes (OK -> WARNING, CRITICAL -> OK, etc)
#contact.someuser.command mail -s "Munin" somejuser@fnord.comm
#contact.anotheruser.command mail -s "Munin" anotheruser@blibb.comm
#
# For those with Nagios, the following might come in handy.
# In addition, the services must be defined in the Nagios server
# as well.
#contact.nagios.command /usr/bin/send_nsca nagios.host.comm -c
/etc/nsca.conf

# a simple host tree
[localhost.localdomain]
    address 127.0.0.1
    use_node_name yes

## Then we want totals...
# [foo.com;Totals] #Force it into the "foo.com"-domain...
#     update no    # Turn off data-fetching for this "host".
#
# The graph "load1". We want to see the loads of both machines...
# "fii=fii.foo.com:load.load" means "label=machine:graph.field"
```

The first variables specify the location of all the files that the master needs. Important to note here are the location of the statistics database (`dbdir`) and the location of the HTML output (`htmldir`). All the data gathered by the Munin master is written to the database directory. The master then uses a tool called `munin-graph` to analyze the data and output graphs into the HTML output directory. If you want to change these later on, you can, but we will leave them at their defaults for now.

Further down are some settings for graphing. By default, updating of the graphs is triggered by cron, but in very large systems it would be smarter to generate them on demand through CGI. Just leave it at cron for now.

After that, we see the notification options. If you want, the master can trigger an alert when a graph goes from ok to warning or from warning to critical. Munin can either send an alert to an e-mail address or pass it on to Nagios. I recommend that you set up the mail notification at this point. This will notify you by e-mail if something is amiss. Add Nagios after you have completed your entire Munin installation and have had it up and running for about a month. This will prevent configuration errors in Munin from waking you up in the middle of the night.

A really interesting bit for now is the **host tree**. Here we can configure all the nodes we want our master to query for information. By default, it only looks at localhost, so effectively, it just monitors itself.

In addition to just hosts, your host tree can also include custom dashboard elements. The configuration file shows a couple of examples. Simple features that you can add are combining counters of multiple machines. In this example, the load of two machines is added into a new graph, which is not a very useful example. Features such as combining the web requests of all your web servers into one single graph, however, could be a very handy addition to your dashboard.

Munin node configuration

By default, the configuration of the Munin node is located at `/etc/munin/munin-node.conf`. Use your favorite text editor to open that file and look at its contents. It is also fairly well documented. However there are two settings that I would like to point out at this stage as you will need them in the future. These settings are highlighted in the following configuration file.

Example configuration file for munin-node

An example configuration file for munin-node is as follows:

```
## Example config-file for munin-node
#

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

```
background 1
setuid 1

user root
group root

# Regexprs for files to ignore
ignore_file [#~]$
ignore_file DEADJOE$
ignore_file \.bak$
ignore_file %$
ignore_file \.dpkg-(tmp|new|old|dist)$
ignore_file \.rpm(save|new)$
ignore_file \.pod$

# Set this if the client doesn't report the correct hostname when
# telnetting to localhost, port 4949
#
#host_name localhost.localdomain

# A list of addresses that are allowed to connect. This must be a
# regular expression, since Net::Server does not understand
# CIDR-style network notation unless the perl module Net::CIDR is
# installed. You may repeat the allow line as many times as you'd
# like

allow ^127\.0\.0\.1$
allow ^::1$

# If you have installed the Net::CIDR perl module, you can use one
# or more cidr_allow and cidr_deny address/mask patterns.
# A connecting client must match any cidr_allow, and not match any
# cidr_deny. Note that a netmask must be provided, even if it's /32
#
# Example:
#
# cidr_allow 127.0.0.1/32
# cidr_allow 192.0.2.0/24
# cidr_deny 192.0.2.42/32

# Which address to bind to;
```

```

host *
# host 127.0.0.1

# And which port
port 4949

```

Firstly, there is the `host_name` setting. This is the name the machine responds with. By default this is the hostname, but in a lot of internal network environments, the configured hostname of a host differs from its network name. If this is the case, you can enter the hostname your master expects here.

Secondly, there is the `allow` section. The master connects to its nodes by connecting them through port 4949, defined at the bottom of the configuration file. The node uses the `allow` option to block information requests not coming from the master. For now, `localhost` is fine, since we are just monitoring ourselves, but if you are running `munin-node` on an external host, you need to enter the IP address or range of your master here. If you want to know more about this, you can skip forward to the *Monitoring additional servers* feature.

Managing plugins

All the active plugins can be found at `/etc/munin/plugins`. Go to this directory and take a look at its contents. As you can see, the plugins that are currently installed are all symbolic links to default Munin plugins at `/usr/share/munin/plugins/`. Munin-node created these symbolic links during its installation as it detected that these plugins were useful to have on your system. If you ever add a plugin to your system, you can place them here.

```

admin@server:/etc/munin/plugins# ls -la
cpu -> /usr/share/munin/plugins/cpu
df -> /usr/share/munin/plugins/df
df_inode -> /usr/share/munin/plugins/df_inode
diskstats -> /usr/share/munin/plugins/diskstats
entropy -> /usr/share/munin/plugins/entropy
forks -> /usr/share/munin/plugins/forks
fw_packets -> /usr/share/munin/plugins/fw_packets
http_loadtime -> /usr/share/munin/plugins/http_loadtime
if_err_eth0 -> /usr/share/munin/plugins/if_err_
if_eth0 -> /usr/share/munin/plugins/if_
interrupts -> /usr/share/munin/plugins/interrupts
iostat -> /usr/share/munin/plugins/iostat
iostat_ios -> /usr/share/munin/plugins/iostat_ios
irqstats -> /usr/share/munin/plugins/irqstats
load -> /usr/share/munin/plugins/load
memory -> /usr/share/munin/plugins/memory
munin_stats -> /usr/share/munin/plugins/munin_stats
ntp_kernel_err -> /usr/share/munin/plugins/ntp_kernel_err

```

Instant Munin Plugin Starter

```
ntp_kernel_pll_freq -> /usr/share/munin/plugins/ntp_kernel_pll_freq
ntp_kernel_pll_off -> /usr/share/munin/plugins/ntp_kernel_pll_off
ntp_offset -> /usr/share/munin/plugins/ntp_offset
open_files -> /usr/share/munin/plugins/open_files
open_inodes -> /usr/share/munin/plugins/open_inodes
processes -> /usr/share/munin/plugins/processes
proc_pri -> /usr/share/munin/plugins/proc_pri
swap -> /usr/share/munin/plugins/swap
threads -> /usr/share/munin/plugins/threads
uptime -> /usr/share/munin/plugins/uptime
users -> /usr/share/munin/plugins/users
vmstat -> /usr/share/munin/plugins/vmstat
```

Please note that some distributions might put their shared plugins at a different location. The configuration file of the plugins themselves is located in `/etc/munin/plugin-conf.d/munin-node`. The convention here is quite straightforward. The name of the plugin is listed at the top of each section in `[accolades]`. Below that are the specific options for each plugin. Open up the file and take a look at its contents.

The most important are the user and group settings. These dictate which user and group the plugin will run as. By default, this will be `nobody:munin`, and a lot of plugins can work just fine with this, but some plugins need more permissions to be able to get the information they need. Always try to keep the permissions here as tight as possible.

Most of the other settings in this file are environment settings. These are used to pass additional parameters to the plugins. Most plugins are configured by their symlink, but additional options, such as locations of files or ports to monitor, can be set here. For example, if we open up the `munin-node` configuration and look at the `dhcpd3` section, you should see something like the following :

```
[dhcpd3]
user root
env.leasefile /var/lib/dhcp3/dhcpd.leases
env.configfile /etc/dhcp3/dhcpd.conf
```

The `dhcpd3` plugin should be run as `root` because, otherwise, it cannot read the information it needs. The lease and config files are passed as environment variables to the plugin as they can be different per distribution.

It is also possible to use wildcards in the section description. If you scroll further down, you will see a section labeled `[if_*]`. This matches interface plugins such as `if_eth0` and `if_eth1`.

Top 6 features you need to know about

Adding the same host that runs the master as the first node was just the first step in connecting the rest of your network. In this section, you will learn how to add additional nodes to your master so that you can monitor additional servers and devices.

After that, we will look into plugins. First you will learn how to install the commonly used plugins to monitor sensors. Next, you will build a simple plugin using bash, which we will expand to a full-fledged Munin plugin in Perl. Finally, we will be looking at some of the more advanced features you can build into your plugin.

Monitoring additional servers

The first step in expanding your munin cluster will be monitoring another server. Once you know how to add one server, you will be able to add all of them! We will do this in four simple steps.

Step 1 – Installing munin-node

First we need to connect to the server we want to monitor and install munin-node. In our examples, we will be using the name `muninnode` as the name of our additional server. Your server will probably have a different name, so every time you see `muninnode` in an example, you should replace that with the name of the server you are using. Examples will also use the term `username`, which you should replace with your username. But first, let's install munin-node.

For Debian or Ubuntu, use the following commands:

```
ssh username@muninnode
sudo apt-get install munin-node
```

For Red Hat or Fedora, use:

```
ssh username@muninnode
sudo yum install munin-node
```

Next, we will take a look at the generated configuration file. It is located at `/etc/munin/munin-node.conf`. Please open it up in your favorite editor.

The first thing we have to take care of is the fact that we want our master to be able to connect to this node. For security reasons, munin-node defaults to allowing only connections from the localhost to query its data. So, let's scroll down to the `allow` section and add a line beneath it.

If your master has a static IP address, please enter it in the `allow` section in the following format:

```
allow 10.0.0.200
```

This will grant the master at `10.0.0.200` access to the data of this node.

Instant Munin Plugin Starter

If your server has a dynamic IP or you want to trust your entire network range, you can either add a single line for every possible IP addresses or use a `cidr_allow` section. Please note that you can only use this if you have the Net::CIDR Perl module installed. Most systems will have this by default, but if you are having problems, you should check that.

```
cidr_allow 10.0.0.0/24
```

This will grant anyone connecting from any IP from 10.0.0.0 to 10.0.0.255 to fetch all the information available in this node.

After you have done this, you need to save the file and restart the munin-node daemon.

For older versions of Debian or Ubuntu, use the following command:

```
sudo invoke-rc.d munin-node restart
```

For Debian or Ubuntu and Red Hat or Fedora, use:

```
sudo service munin-node restart
```

Step 2 – Testing your munin-node installation

Now that we have installed the node, it is a good idea to check if it functions correctly. We will do this by connecting to the node and fetching some information.

```
ssh username@muninnode
telnet localhost 4949
version
list
quit
```

You should get the following output:

```
ssh username@muninnode

Welcome to muninnode

username@muninnode:~$ telnet muninnode 4949
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

# munin node at muninnode.
version
```

```
munins node on muninnode. version: 2.0.9-2
```

```
list
cpu df df_inode entropy forks fw_packets http_loadtime if_err_eth0
if_eth0 interrupts iostat iostat_ios irqstats load memory
munin_stats ntp_kernel_err ntp_kernel_pll_freq ntp_kernel_pll_off
ntp_offset open_files open_inodes proc_pri processes sensors swap
threads uptime users vmstat
quit
Connection closed by foreign host.
```

```
username@muninnode:~$
```

Please note that the node might be a bit impatient with you. If you connect to it using Telnet and then give no further instructions for a few seconds, munin-node will automatically disconnect you, thinking you are just wasting its time. If this happens, just go ahead and try again.

Now that we know that munin-node is running, we want to make sure it is functioning correctly. Munin-node keeps its log file at `/var/log/munin/munin-node.log`. Let's take a look at that.

```
ssh username@muninnode
tail /var/log/munin/munin.log
```

You should be able to see your connection attempt in the log; it should look something like the following:

```
2013/01/01-12:30:10 CONNECT TCP Peer: "127.0.0.1:44363" Local: \
"127.0.0.1:4949"
```

If you have a node that is experiencing problems with connections or a plugin, make sure to look at this log file for exceptions or error messages.

Step 3 – Installing additional plugins

When munin-node was installed, it ran its autodetect script to enable plugins from its standard library if they were applicable to your system. If you have installed new software on this machine, you can easily re-run this script to see if Munin can help you monitor the new software. If you, for example, have installed MySQL or PostgreSQL, then this is what you do:

```
ssh username@muninnode
sudo munin-node-configure --suggest
sudo munin-node-configure --shell
```

The first command will show you all the plugins your munin-node has out of the box and whether they apply to your system. The second command will display the commands you will have to execute to create the symbolic links in order to enable those suggestions. Please note that not all plugins support this, and therefore, not all applicable plugins will automatically be enabled.



Munin-node has to be restarted after you've added new plugins; otherwise, these changes will not take effect.

Step 4 – Adding the new node to the master

Now that we've completely configured the node and tested to see if it works, we are ready to add the node to our master. To do this, we have to go to our master and test whether we can connect back to our munin-node.

```
ssh username@muninmaster
telnet 4949 muninnode
version
list
quit
```

This should display the version and the capabilities of the munin-node running on the muninnode server. If this does not work, make sure you have started the munin-node on the muninnode server and also check whether firewalls allow you to connect to it on port 4949. Also go ahead and recheck the allowed IP addresses in the munin-node configuration as mentioned in step 2.

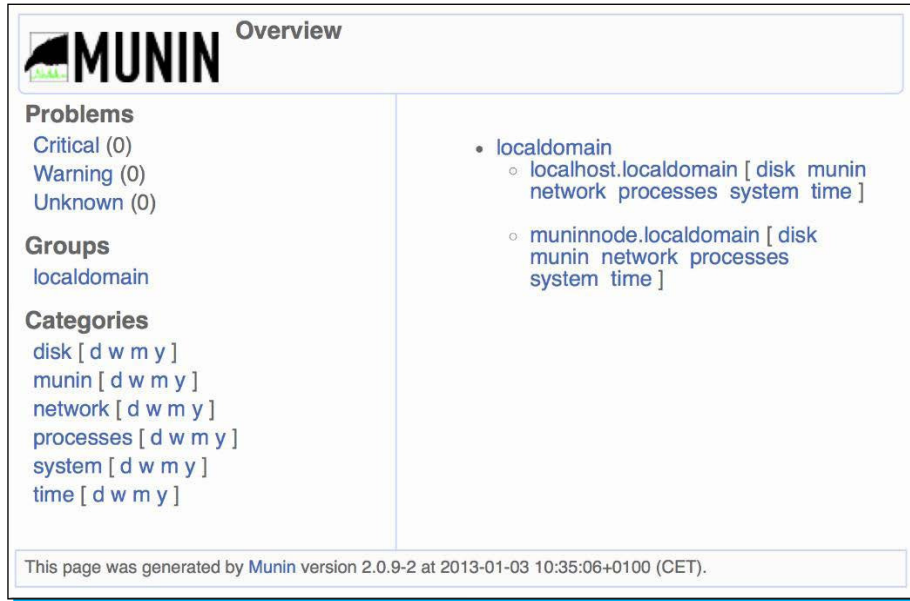
If this is working correctly, go ahead and open up the file at `/etc/munin/munin.conf`. Here, we scroll down until we see the following host tree:

```
# a simple host tree
[localhost.localdomain]
    address 127.0.0.1
    use_node_name yes
```

We need to add our new munin-node to this host tree as follows:

```
# the host tree of our local network
[localhost.localdomain]
    address 127.0.0.1
    use_node_name yes
[muninnode.localdomain]
    address 10.0.0.200
    use_node_name yes
```

Now, we'll have to wait at least 10 minutes before we will be able to see our new node on the Munin master's website. Go ahead and point your browser to your Munin master at <http://localhost/munin> or at http://your_munin_master/munin; you should see something like the following screenshot:



After a couple of minutes, you should be able to see graphs for your node and even compare the nodes of your cluster side by side.

Troubleshooting

Now it could very well be possible that it isn't working for you. Here are the few steps you should check first:

- ◆ Check the Munin master log at `/var/log/munin/munin.log` for errors.
- ◆ Check the Munin node log at `/var/log/munin/munin-node.log` on the munin server for access calls and errors.
- ◆ Try to connect from your Munin master to your node using Telnet 4949.
- ◆ If you can connect, type `nodes` and check whether the name of your node is there.
- ◆ Still in Telnet, type `list muninnode.localdomain` and check whether you get a list of plugins. If not, add your hostname to `/etc/munin/munin-node.conf` (see the *Munin node configuration* section).

Monitoring additional devices

Not all things you want to monitor will be able to run munin-node themselves. If you want to monitor routers, switches, or printers, you will have to set up a munin-node to do that for you. Of course, the device will need to be networked and have either SNMP or another interface exposed that can be used to monitor it. In this section, we will be going through the steps of doing just that. If you understand how you can add a single networked device, you can go ahead and start monitoring all the devices in your network.

We will be setting up monitoring of a networked router through SNMP. Note that, while most routers can be monitored through SNMP, some can't. So this example might not work for your setup.

Step 1 – Enabling the SNMP interface plugin

Munin-node comes with a handy plugin that can monitor the traffic of most switches and routers through SNMP. In this example, we will be monitoring a core router named `router`, which has an IP address of `10.0.0.1`.

We will start by enabling the plugins, as follows:

```
ln -s /usr/share/munin/plugins/snmp__if_  
/etc/munin/plugins/snmp_router_if_1  
  
ln -s /usr/share/munin/plugins/snmp__if_err_  
/etc/munin/plugins/snmp_router_if_err_1
```



The hostname of our router is part of the symlink.

Step 2 – Testing the SNMP interface plugin

We start by testing the plugin.

```
cd /etc/munin/plugins  
munin-run snmp_router_if_1
```

This should give you the following output:

```
recv.value 2928754283  
send.value 562680241
```

If you get `recv.value noSuchObject`, your switch isn't supported by this SNMP plugin and you will have to look for a specific one for your switch or router (see the *People and places you should get to know* section).

If you get Unable to resolve UDP/IPv4 address 'router', the hostname of your router could not be resolved. You can easily fix this by changing the hostname or by adding the hostname and the IP address of your router to the hosts file of your server in `/etc/hosts`.

Now that we have the plugin working, we have to make sure that the node publishes our router correctly. Start by restarting munin-node so it picks up our changes to its plugins.

Next, we connect to the munin-node through Telnet and we query it for nodes. If everything is correct, you should see our new node, `router`. If we list the plugins for router through `list router`, we get the plugins for this node; they are `snmp_router_if_1` and `snmp_router_if_err_1`. To test them we can call a fetch on them using `fetch snmp_router_if_1`.

```
$ sudo service munin-node restart
```

```
Stopping Munin-Node: done.
```

```
Starting Munin-Node: done.
```

```
$ telnet localhost 4949
```

```
Trying 127.0.0.1...
```

```
Connected to localhost.
```

```
Escape character is '^['.
```

```
# munin node at muninmaster
```

```
$ nodes
```

```
muninmaster
```

```
router
```

```
.
```

```
$ list router
```

```
snmp_router_if_1
```

```
snmp_router_if_err_1
```

```
$ fetch snmp_router_if_1
```

```
.
```

```
recv.value 2928754283
```

```
send.value 562680241
```

```
.
```

```
$ quit
```

Step 3 – Configuring the Munin master

We go back to opening the master configuration file at `/etc/munin.conf` on the Munin master and then go to our host tree.

Here we will add the following section:

```
[router]
    use_node_name no
    address 127.0.0.1
```

Even though `router` is the name of the node, the master will not use that name. By default, the Munin master will try to get information about our router by calling `list localhost.localdomain`. This is, of course, not what we want. To make sure the master queries using `list router`, we have to set `use_node_name` to `no`.

After you've restarted the Munin master, you should see the traffic from your router in the web interface.



If you want to find out whether one of the default SNMP plugins provided by Munin is applicable for a specific host, run `munin-node-configure --shell --snmp hostname` and try the suggestions Munin gives you.

Installing sensor plugins

Some of the most frequently used plugins with `munin-node` are the plugins for monitoring sensors. This allows you to view temperatures, fan speeds, and voltages of your system.

There are two main sensor plugins. First, there is the `sensors` plugin that is shipped with Munin. This depends on the `lm-sensors` package and uses kernel modules to read out information. This method works on a lot of systems, but most modern servers these days use IPMI to expose their sensors. In this section, we will first try to use the `sensors` package. If this fails, please try to use the IPMI variant, which can be found in the next section.

Monitoring sensors through lm-sensors

To use it, we must first install the `lm-sensors` package.



The autodetection of sensors used by the `lm-sensors` package might cause instability of your server. Please make sure you do this either during a maintenance window or on a server that can be rebooted at any time. I have personally never seen it go wrong, but it is better to be safe than sorry.

For Debian or Ubuntu, use the following commands:

```
ssh username@muninnode
sudo apt-get install lm-sensors
sudo sensors-detect
sensors
```

For Red Hat or Fedora, use:

```
ssh username@muninnode
sudo yum install lm-sensors
sudo sensors-detect
sensors
```

After this, please answer all the questions with yes.

```
ssh username@muninnode
```

```
Welcome to muninnode
```

```
username@muninnode:~$ sudo sensors-detect
# sensors-detect revision 5984 (2011-07-10 21:22:53 +0200)
# System: Unknow Unknow
```

```
This program will help you determine which kernel modules you need
to load to use lm_sensors most effectively. It is generally safe
and recommended to accept the default answers to all questions,
unless you know what you're doing.
```

```
# OUTPUT SHORTENED FOR READABILITY
```

```
Now follows a summary of the probes I have just done.
Just press ENTER to continue:
```

```
Driver 'it87':
  * ISA bus, address 0x228
    Chip 'ITE IT8712F Super IO Sensors' (confidence: 9)
```

```
Driver 'k8temp' (autoloaded):
  * Chip 'AMD K8 thermal sensors' (confidence: 9)
```

Instant Munin Plugin Starter

To load everything that is needed, add this to `/etc/modules`:

```
#----cut here----  
# Chip drivers  
it87  
k8temp  
#----cut here----
```

Do you want to add these lines automatically to `/etc/modules`? (yes/NO)
yes

Successful!

The `lm-sensors` package automatically installs the required kernel modules for us. We can check this by looking at the contents of `/etc/modules`.

```
# /etc/modules: kernel modules to load at boot time.  
#  
# This file contains the names of kernel modules that should be  
# loaded at boot time, one per line. Lines beginning with "#" are  
# ignored.  
  
loop  
lp  
  
# Generated by sensors-detect  
# Chip drivers  
it87  
k8temp
```

If there is nothing there, your system is probably not supported by `lm-sensors`, and you should advance to the section about monitoring through IPMI. If you do have modules in place here, we can use them to look at the sensor values of our server in the terminal:

```
username@muninnode:~$ sensors  
acpitz-virtual-0  
Adapter: Virtual device  
temp1:          +40.0°C (crit = +75.0°C)  
  
k8temp-pci-00c3  
Adapter: PCI adapter  
Core0 Temp:     +23.0°C  
Core0 Temp:     +23.0°C  
Core1 Temp:     +21.0°C
```

```
Corel Temp:    +20.0°C
```

```
it8712-isa-0228
```

```
Adapter: ISA adapter
```

```
in0:           +1.18 V (min = +1.01 V, max = +4.08 V)
in1:           +3.01 V (min = +0.00 V, max = +3.95 V)
in2:           +3.42 V (min = +0.00 V, max = +4.08 V)
in3:           +2.94 V (min = +2.50 V, max = +4.08 V)
in4:           +2.98 V (min = +0.00 V, max = +4.08 V)
in5:           +3.25 V (min = +0.00 V, max = +4.08 V)
in6:           +2.98 V (min = +0.00 V, max = +4.08 V)
in7:           +2.94 V (min = +0.00 V, max = +4.08 V)
Vbat:          +2.83 V
fan1:          2689 RPM (min = 2657 RPM, div = 2)
temp1:         +57.0°C (low= +127.0°C, high = +75.0°C)
temp2:         +56.0°C (low = +127.0°C, high = +75.0°C)
temp3:         +51.0°C (low = +127.0°C, high = +75.0°C)
cpu0_vid:      +0.000 V
```

Your output will differ from mine as your system will have a different chipset and `lm-sensors` will detect different sensors.

Our next step will be to enable the plugin that parses the output of the sensor's command and outputs something that Munin understands. This will in turn generate graphs of this data. We do this by running the sensors plugin, as follows:

```
$ ssh username@muninnode
username@muninnode:~$ cd /usr/share/munin/plugins
username@muninnode:/usr/share/munin/plugins$ ./sensors_ suggest
fan
volt
temp
```

In my case, you can see that the Munin sensors plugin discovers three groups of sensors it can read out: `fan`, `volt`, and `temp`. To enable these, we have to create symbolic links for them:

```
ssh username@muninnode
$ cd /etc/munin/plugins
$ sudo ln -s /usr/share/munin/plugins/sensors_
```

Instant Munin Plugin Starter

```
/etc/munin/plugins/sensors_fan
$ munin-run sensors_fan
```

```
fan1.value 2678
```

You will see that the output from the Munin sensors plugin is identical to the `sensors` command we had given earlier, so this means it works! Go ahead and enable any other suggestions that the Munin sensors autosuggest has given you by creating a symbolic link for them as well.

Now restart munin-node and let's make sure that our Munin node exposes these sensors correctly:

```
ssh username@muninnode
telnet localhost 4949
fetch sensors_fan
```

They should give you the following output:

```
username@muninnode:~$ telnet localhost 4949
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
# munin node at muninnode
fetch sensors_fan
fan1.value 2689
.
quit
Connection closed by foreign host.
username@muninnode:~$
```

You will see that when we ask the node to fetch `sensors_fan`, it executes the plugin and returns this information over a network connection.

You have also learned that plugins can take part of their configuration from the name of their symlink. `sensors_volt` and `sensors_fan` call the same `sensors_script` function but use the name of the symlink to filter out different results. This keeps down the number of duplicate plugins, thus keeping the configuration as simple as possible.

Monitoring sensors through IPMI

To use it, we must first install the `freeipmi-tools` package as the `freeipmi` plugin depends on it to read out the sensor values.

For Debian or Ubuntu, use the following commands:

```
ssh username@muninnode
sudo apt-get install freeipmi-tools
sudo ipmi-sensors
```

For Red Hat or Fedora, use:

```
ssh username@muninnode
sudo yum install free-ipmi-tools
sudo ipmi-sensors
```

If your system supports IPMI, this should output a large array of available sensors:

ID	Name	Type	Reading	Units	Event
7	Ambient Temp	Temperature	23.00	C	'OK'
9	CMOS Battery	Battery	N/A	N/A	'OK'
24	FAN MOD 1A RPM	Fan	5040.00	RPM	'OK'
25	FAN MOD 1B RPM	Fan	3480.00	RPM	'OK'
26	FAN MOD 2A RPM	Fan	5040.00	RPM	'OK'
27	FAN MOD 2B RPM	Fan	3480.00	RPM	'OK'
28	FAN MOD 3A RPM	Fan	5040.00	RPM	'OK'
29	FAN MOD 3B RPM	Fan	3480.00	RPM	'OK'
30	FAN MOD 4A RPM	Fan	5040.00	RPM	'OK'
31	FAN MOD 4B RPM	Fan	3480.00	RPM	'OK'
32	FAN MOD 5A RPM	Fan	4920.00	RPM	'OK'
33	FAN MOD 5B RPM	Fan	3480.00	RPM	'OK'
48	OS Watchdog	Watchdog 2	N/A	N/A	'OK'
50	Intrusion	Physical	N/A	N/A	'OK'
51	PS Redundancy	Power Supply	N/A	N/A	'Redundant'
52	Fan Redundancy	Fan	N/A	N/A	'Redundant'
56	Current 1	Current	0.20	A	'OK'
57	Current 2	Current	0.20	A	'OK'
58	Voltage 1	Voltage	228.00	V	'OK'
59	Voltage 2	Voltage	228.00	V	'OK'
60	System Level	Current	91.00	W	'OK'

Instant Munin Plugin Starter

Next, we need to get the `freeipmi` plugin. This plugin is not available by default, so we need to download it from the GitHub repository.

```
username@muninnode:~$ cd /etc/munin/plugins
username@muninnode:/etc/munin/plugins$ wget
'https://raw.githubusercontent.com/
munin-monitoring/contrib/master/plugins/sensors/freeipmi'
username@muninnode:/etc/munin/plugins$ chmod +x freeipmi
```

Next, we need to run `freeipmi` as root by adding the following to the bottom of the `/etc/munin/plugin-conf.d/munin-node` file:

```
[freeipmi]
user root
```

And now, to test it, use the following commands:

```
username@muninnode:~$ cd /etc/munin/plugins
username@muninnode:/etc/munin/plugins$ munin-run freeipmi
```

You should get output similar to the following:

```
multigraph freeipmi_voltage
ipmi58.value 228.00
ipmi59.value 228.00
multigraph freeipmi_fan
ipmi24.value 5040.00
ipmi25.value 3480.00
ipmi26.value 5040.00
multigraph freeipmi_temp
ipmi7.value 23.00
multigraph freeipmi_current
ipmi56.value 0.20
ipmi57.value 0.20
multigraph freeipmi_power
ipmi60.value 98.00
```

If you get the preceding output, everything is working. Quickly bring up the web interface of your Munin master and take a look at all your sensory data.

Writing a simple plugin

In the previous sections, we have mainly been focusing on how you can enable and configure existing plugins for a node. One of the great advantages of Munin, however, is the fact that it is really easy to write your own plugins.

Basically, a plugin is a simple executable or script that conforms to the plugin API. There are just two things a minimal Munin plugin must do in order to conform to this API:

- ◆ Provide the layout of the graph that the Munin master has to draw
- ◆ Output a list of labels and measurements on execution

If we take a look at a simple plugin, such as the load plugin, we can see how it does this:

```
/etc/munin/plugins$ munin-run load config
graph_title Load average
graph_vlabel load
graph_scale no
graph_category system
load.label load
graph_info The load average of the machine describes how many
processes are in the run-queue (scheduled to run "immediately").
load.info 5 minute load average

/etc/munin/plugins$ sudo munin-run load
load.value 1.44
```

So, the current load of this system is 1.44, which is the value that the master uses to draw a point in its graph. The `config` output specifies the rest of the graph, the vertical label on the graph should read `load`, and the graph should not have a scale. It should be placed in the `system` category. `graph_info` contains the information shown when you hover over the graph in the web interface.

As an example, we will be writing a similar plugin that monitors the amount of disk space used by a specific user in four simple steps.

Step 1 – Gathering the information we need

Systems are packed with handy tools that make it very easy to get the size of a specific directory, so we will try not to reinvent the wheel. We will be using the disk usage utility, `du`, as it is available on most Unix-based systems and does almost everything that we want.

```
~$ du -s -b /home/testuser
1323302912 /home/testuser
```

Instant Munin Plugin Starter

`du -s` gives us a neat total of the test user's directory, while the `-b` flag makes sure we are consistently getting bytes, which is needed to get a good, consistent graph. We don't need the `/home/testuser` bit, so we need to chop that part off:

```
~$ du -s -b /home/testuser | awk '{print $1}'  
1323302912
```

To achieve this, we pipe the original output of `du` into `awk`. `awk` then cuts the line at the first space into two lines and prints the first part, leaving just 1323302912.



`du` can take a very long time if the user you are testing has a lot of files in his directory. If it takes longer than 5 minutes, munin-node will assume something is wrong with the plugin and interrupt its execution. Please choose a small user directory to try this example as it will save you a lot of time. If you don't have a small user on your system, create a user named `testuser`, and place a few files in its home directory.

Step 2 – Designing the graph

The Munin master needs some context information about the value that we are returning. It needs to know what base it is, what the maximum and minimum values are, and so on.

```
graph_title Diskusage of testuser  
graph_args --base 1024 --lower-limit 0  
graph_vlabel bytes  
graph_category system  
userdiskusage.label Average disk usage the last five minutes.  
userdiskusage.warning 2147483648  
userdiskusage.critical 5368709120  
graph_info The disk usage of testuser in bytes.
```

As disk space is measured in blocks of 1,024 bits, we have to set the graph base to 1,024. If you are monitoring something like a network speed or temperature, you need to change the base to 1,000.

As we cannot have negative disk usage, we set the lower limit of the graph to zero. We also add a warning for if the user goes over 2 GB and a critical message for if the user goes over 5 GB.

These `graph_args` arguments are passed directly to the graphing engine of munin, which is called **RRDGraph**. If you want to know more about the graphing possibilities, you can find the documentation for the RRDtool at <http://oss.oetiker.ch/rrdtool/doc/rrdtool.en.html>.

Step 3 – Writing the plugin

Now, we need to combine steps 1 and 2 into a plugin. For this simple plugin, we will be doing this with some bash code:

```
#!/bin/bash
case $1 in
  config)
    echo "graph_title Diskusage of testuser"
    echo "graph_args --base 1024 --lower-limit 0"
    echo "graph_vlabel bytes"
    echo "graph_category system"
    echo "testuserdiskusage.label Average disk usage."
    echo "testuserdiskusage.warning 2147483648"
    echo "testuserdiskusage.critical 5368709120"
    echo "graph_info The disk usage of testuser in bytes."
    exit 0;
  esac

  echo -n "testuserdiskusage.value "
  du -s -b /home/testuser | awk '{print $1}'
```

So, if `config` gets passed as the first argument, we print out the graph options that we need and exit. If nothing gets passed, we print `testuserdiskusage.value` and the value that `du` is giving us.

Step 4 – Testing the plugin

Now, to test the plugin, we run it:

```
/etc/munin/plugins$ ./testuserdiskusage
testuserdiskusage.value 1323302912
```

It works! So next, we try to run it using `munin-run`:

```
munin-run testuserdiskusage
testuserdiskusage.value du: cannot read directory '/home/testuser':
Permission denied
0
```

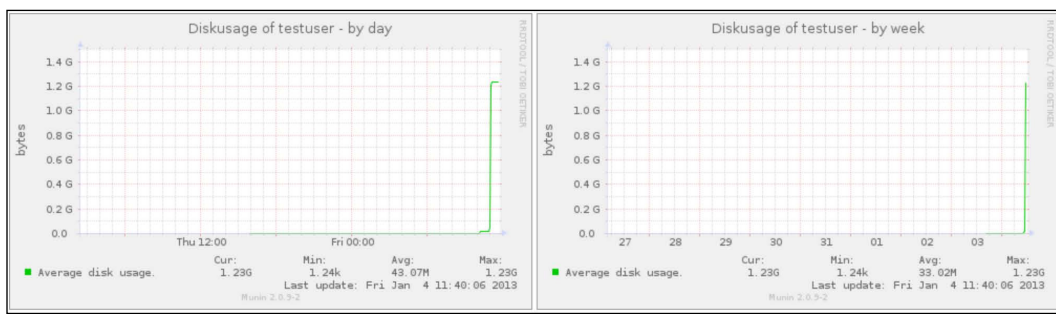
Oh! What is going on here? Well, the first time the script gets executed as your user, and your user can probably look at the contents of `/home/testuser` to calculate its total size. However, by default, `munin-node` runs plugins as the user `nobody` that, in this case, does not have enough permissions to view the contents of `/home/testuser`. To fix this, we need to open `/etc/munin/plugin-conf.d` in your favorite editor and add the following section:

```
[testuserdiskusage]
user testuser
```

Instant Munin Plugin Starter

This will make munin-run run the plugin under the user `testuser`, which is the safest option in this case. Just re-run the plugin with `munin-run` to verify that it works and then take the following steps:

1. Restart `munin-node` (`service munin-node restart`).
2. Verify that the plugin has been picked up by `munin-node` using Telnet on port 4949.
3. Fetch the value of `testuserdiskusage` as a test.
4. Verify the plugin works through Telnet by connecting from the Munin server to the Munin node.
5. Open up the Munin web interface and see the results of your work.



Challenge yourself!

And now to test your own abilities! The three following assignments get increasingly difficult:

- ◆ Change the plugin such that you can set the username in the environment so that it can monitor other users as well.
- ◆ Rename the plugin to `userdiskusage_` and create a symlink to it, called `userdiskusage_testuser`. Modify the plugin so that it fetches the username from the symlink. This way you can monitor any user by just creating a symlink.
- ◆ Make sure that the script is executed with only the privileges of the user it is monitoring.

Writing a complex plugin

So you've written your first plugin! The plugin you've built is a very basic one, but those are usually good enough to monitor something in your network for years and years. If you want to go one step further and provide your plugin to other people or even to Munin core, you have to apply a bit more polish to your plugin.

The Munin community calls plugins that check all the boxes *vetted plugins*, and we will be writing one together. A vetted plugin needs to provide the following:

- ◆ Graph information on the `config` argument
- ◆ Data on the `fetch` argument
- ◆ Correct documentation stating author and functionality
- ◆ Magic markers that state the correct capabilities of the plugin
- ◆ Autodetect its usefulness on the system through the `autoconf` argument
- ◆ Suggest possible use cases when the `suggest` argument is passed

We will be rewriting our plugin so that it complies with all of these requirements.

Step 1 – Rewriting our bash plugin to Perl

We will start by wrapping our bash script with just enough Perl to make the generation of our graph a lot more flexible.

```
#!/usr/bin/perl
# -*- cperl -*-

use warnings;
use strict;
use utf8;

use Munin::Plugin;

# Process and setup global variables
my $warn_level = 2147483648;
$warn_level = $ENV{'warning'} if defined($ENV{'warning'});
my $crit_level = 5368709120;
$crit_level = $ENV{'critical'} if defined($ENV{'critical'});
my $username = $Munin::Plugin::me;
$username =~ s/^user_diskusage_//g;
my $home = "/home";
$home = $ENV{'alt_home'} if defined($ENV{'alt_home'});
my $user_directory = "$home/$username";
my $du = defined $ENV{'du'} ? $ENV{'du'} : "/usr/bin/du";

sub Config {
    print "graph_title Diskusage of $username\n";
    print "graph_args --base 1024 --lower-limit 0\n";
    print "graph_vlabel bytes\n";
```

```
print "graph_category system\n";
print "user_diskusage.label Average disk usage.\n";
print "user_diskusage.warning $warn_level\n";
print "user_diskusage.critical $crit_level\n";
print "graph_info The disk usage of $username in bytes.\n";
}

sub Fetch {
    my $du_output = '$du -s -b $user_directory';
    my @du_items = split(/\s+/, $du_output);
    print "user_diskusage.value $du_items[0]\n";
}

sub Execute {
    if (defined $ARGV[0] && $ARGV[0] ne '') {
        my $command = $ARGV[0];
        if ($command eq 'config') { Config(); }
        elsif ($command eq 'autoconf') { AutoConf(); }
        elsif ($command eq 'suggest') { Suggest(); }
    } else {
        Fetch();
    }
}

Execute(); # Run it!
exit(0);
```

As not everyone is probably familiar with Perl, I'll step through the file to highlight the specific points of interest.

At the top, we declare that this is a Perl file and that we want to make use of `Munin::Plugin`. The `Munin::Plugin` class is supplied with Munin and provides us with some convenient methods. We use `$Munin::Plugin::me` to fetch the name of the plugin.

After that, we process all our global variables. As we want this plugin to be as versatile as possible, we swap all of the hardcoded values from our bash script for configurable values.

We set the warning and critical levels to the defaults or fetch them from the munin environment. We use a regular expression to get the username from the symlink, and we use that to build the `$user_directory` variable. Because somebody might use a different location as the value `/home`, we make that configurable as well.

The script also needs to know where `du` is located so it can call it. Because these might not always be in `/usr/bin/`, this is made configurable as well.

The `Config` and `Fetch` methods are exact copies of the output of our bash script earlier, except that we now do the splitting in Perl. We inserted the variables we've just defined into their correct positions.

Lastly, the `Execute` method wraps it all together. It looks at the first argument that has been passed. If this is the config, we run `Config` to get the config output; otherwise, `Fetch` is run.

Now try out your new plugin using `munin-run`.

Step 2 – Documentation and markers

Documentation is always vital to software, especially if it is a piece of software that a lot of people might use in the future. Munin plugins should have decent documentation, so we'll provide just that.

```
#!/usr/bin/perl
# -*- cperl -*-
=head1 NAME

user_diskusage_ - Plugin to monitor diskusage of specific users.

=head1 APPLICABLE SYSTEMS

Any system that has du support and a home directory.

=head1 CONFIGURATION
This script is used to generate diskusage data for specific users.
To generate data, you need to create a symbolic link named:
user_disk_usage_<USERNAME> to the user_disk_usage_ script.

In addition you need to specify a user that has read rights to the
<USERNAME>'s home directory in your munin-node.conf:
    [user_diskusage_*]
    user root

Or if you want more specific privileges:
    [user_diskusage_john]
    user john

Warning and critical values can be set via the environment variables.
References to an alternative home directory or alternative du
implementation can also be optionally specified.
    [user_diskusage_john]
    user home
    env.alt_home '/alternative_home'
```

Instant Munin Plugin Starter

```
env.du '/full_path_to/du'
env.warning 2147483648
env.critical 5368709120
```

The default warning and critical levels are 2048MB and 5120MB, respectively.

=head1 DEPENDENCIES

This script depends on Perl and a working du application.

=head1 AUTHOR

Bart ten Brinke <info@retrosync.com>, Retrosync

=head1 LICENSE.

Copyright (c) 2012 Bart ten Brinke

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

=head1 VERSION

Version 1.0

=head1 MAGIC MARKERS

```
### family=auto
### capabilities=autoconf suggest

=cut
```

The name and applicable systems and version are self-explanatory, so we won't go into these. I will explain all the other headers one by one, going through them from top to bottom:

- ◆ **Configuration:** The configuration section is very important. Here, you should explain what your plugin does and how it does it. It is recommended that you provide at least two configuration examples. First, a minimal configuration example that contains everything that is needed to get this plugin running. Second, a maximal configuration example that shows all the options that are available and how they might be useful.
- ◆ **Dependencies:** These are again what you expect. As we depend on a functional `du` to be present, we state that here.
- ◆ **Author:** This is you, so add your name here and an e-mail address where you can be reached. If anybody has questions about your plugin, it is always handy for them to be able to contact you.
- ◆ **License:** The license is a vital part of any piece of software that you are building. I have chosen to license my plugin under the MIT license. This license allows anybody to do pretty much anything with my plugin without restrictions. "Why add a license then?" you might ask. Well by including it, we make this fact very clear for the rest of the world. If somebody wants to make our plugin part of the default Munin plugins, this license makes it absolutely clear that this will not cause any legal problems for the Munin project. When no license is explicitly given, people would have to assume that all rights are reserved, which means that it can't be distributed, modified, or used as a basis for other plugins.
- ◆ **Magic markers:** The magic markers are the most puzzling. These are there to tell munin-configure and munin-node what the capabilities are of this plugin. The plugin can belong to any of the following families:
 - `auto`: This plugin can be installed fully automatically by munin-configure
 - `contrib`: This plugin comes from <https://github.com/munin-monitoring/contrib>, a collection of plugins that probably work but are not actively checked and are probably not vetted
 - `example`: A nonfunctional example plugin to show how to build plugins in a specific language or style
 - `manual`: This plugin needs to be installed manually
 - `snmpauto`: This is the same as `auto`, except for `snmp` plugins, configured through `munin-configure --snmp`
 - `test`: These are plugins that are used to test munin-node.

Currently there are only two capability options for Munin plugins:

- ◆ `autoconf`: This plugin supports automatic configuration by `munin-node-configure`
- ◆ `suggest`: This plugin is a wildcard plugin and can suggest options for the plugin

Our plugin currently does not support `autoconf` and `suggest` even though our documentation says that we do. Let's fix this in the next step.

Step 3 – Supporting the `autoconf` argument

`autoconf` is the ability of a plugin to detect whether it will work out of the box for this system.

```
sub AutoConf {
    unless (-e $du) {
        print "no (Cannot find du at $du)\n";
        exit(1);
    }

    my $dir;
    if (!opendir($dir, $home)) {
        print "no (no access to $home)\n";
        exit(1);
    } else {
        closedir($dir);
    }

    print "yes\n";
}
```

We need to be testing as much as possible here. We start by testing our dependencies; `du` should exist. If it doesn't exist, `autoconf` will result in an error. We will also test whether the home directory exists and whether we have access to it.

Try out `autoconf` by running it through `munin-run`.

```
$ munin-run user_diskusage_testuser autoconf
yes
```

To test whether our `autoconf` argument works correctly, try changing the location of `du` in your `/etc/munin/plugin-conf.d/munin-node` file to something incorrect and see if it fails.

```
[user_diskusage_*]
user root
env.du /does/not/exist

$ munin-run user_diskusage_testuser autoconf
no (Cannot find du at /does/not/exist)
```

Make sure to change it back though!

Step 4 – Adding support for the suggest argument

`suggest` gives a list of possible options for a wildcard plugin. For example, the `if_` plugin will tell you which interfaces are available for monitoring. Our plugin should suggest a list of available user directories it can monitor, so let's build that:

```
sub Suggest {
    my $dir;
    opendir($dir, $home);
    while (my $file = readdir($dir)) {
        next unless (-d "$home/$file");
        next if ($file =~ m/^\./);
        print "$file\n";
    }
    closedir($dir);
}
```

We open the home directory (by default, `/home`) and we walk through all the files there. If the file is a directory and doesn't equal `.` or `..`, we suggest this directory to the user.

Try it out by running `munin-run user_disk_usage_ suggest`.

Advanced plugin options

With the latest releases of Munin, some additional options were made available for advanced plugin writers. The most noteworthy of these are `dirtyconfig` and `multigraph`. Both have the potential to speed up complex plugin execution considerably. We will explain each of these through an example.

dirtyconfig

As we explained in the previous section, a plugin gets called with the `config` parameter the first time and without it the second time. For our example plugin, this is fine; during the `config` run we just output our graph options, and in the `fetch` run we do the actual work. There are, however, some plugins for which the only way to find out what the output will look like is to actually generate the output. These plugins will basically be doing the same work for the `config` option as for the `fetch` option.

As an example, we will take a look at the `freeipmi` sensors plugin we used in the previous section at <https://github.com/munin-monitoring/contrib/blob/master/plugins/sensors/freeipmi>.

Instant Munin Plugin Starter

This plugin uses the `ipmi-sensors` command to read out all the sensors that are available and their value. Because each system has a different set of sensors, this means that `ipmi-sensors` would first be called to gather the available labels and then run again to gather the available data. As this is not very efficient, Munin 2.0 has introduced `dirtyconfig`.

When the Munin master connects to the node, it signals the node that it supports `dirtyconfig` by sending a `cap dirtyconfig` command. The node passes this on to its plugins by setting the `MUNIN_CAP_DIRTYCONFIG` environment variable to 1. When a plugin detects this, it may send the config and the fetched result back in a single reply.

```
$ munin-run freeipmi config

multigraph freeipmi_voltage
graph_title Voltages by IPMI
graph_vlabel Volt
graph_args --base 1000 --logarithmic
graph_category sensors
ipmi58.label Voltage 1
ipmi59.label Voltage 2

$ MUNIN_CAP_DIRTYCONFIG=1 munin-run freeipmi config

multigraph freeipmi_voltage
graph_title Voltages by IPMI
graph_vlabel Volt
graph_args --base 1000 --logarithmic
graph_category sensors
ipmi58.label Voltage 1
ipmi59.label Voltage 2

multigraph freeipmi_voltage
ipmi58.value 228.00
ipmi59.value 228.00
```

Updating our disk usage plugin to support this is actually really simple:

```
sub Execute {
    if (defined $ARGV[0] && $ARGV[0] ne '') {
        my $command = $ARGV[0];
        if ($command eq 'config') {
            Config();
            Fetch() if (defined($ENV{MUNIN_CAP_DIRTYCONFIG}));
        }
        elsif ($command eq 'autoconf') { AutoConf(); }
        elsif ($command eq 'suggest') { Suggest(); }
    }
}
```

```

    } else {
        Fetch();
    }
}

```

What you also might have noticed is the fact that the `freeipmi` plugin uses another new feature called multigraph. We will explain that in the next section.

Multigraph

Since Version 1.4, Munin has added support for multigraph plugins. This feature extends the normal graphing options and provides two additional options:

- ◆ Yield multiple graphs instead of one graph
- ◆ Support graph hierarchy

For most situations, the normal style plugin is preferred as it keeps things nice, simple, and contained in a single plugin. However there are situations that are more complex than monitoring the disk usage of a single user. Down-drilling network traffic on a switch or slow queries on a database server are where multigraph plugins can really shine.

So how can I make one myself? Well that's actually not very complicated. Just like with `dirtyconf`, the master sends a `cap multigraph` command to the nodes if it has multigraph support. The node passes this on to its plugins by setting the `MUNIN_CAP_MULTIGRAPH` environment variable.

When we want our plugin to provide the total disk usage of all users as well as a drill-down to specific users, we need to output the config for multiple graphs. This is done by separating each graph config with a multigraph line:

```

multigraph user_diskusage
graph_title Total diskusage of all home users
graph_args --base 1024 --lower-limit 0
graph_vlabel Bytes
graph_category system
user_diskusage.label Average disk usage the last 5 minutes.
user_diskusage.warning 322122547200
user_diskusage.critical 429496729600
graph_info Total diskusage in bytes.

multigraph user_diskusage.testuser
graph_title Diskusage of testuser
graph_args --base 1024 --lower-limit 0
graph_vlabel Bytes
graph_category system

```

Instant Munin Plugin Starter

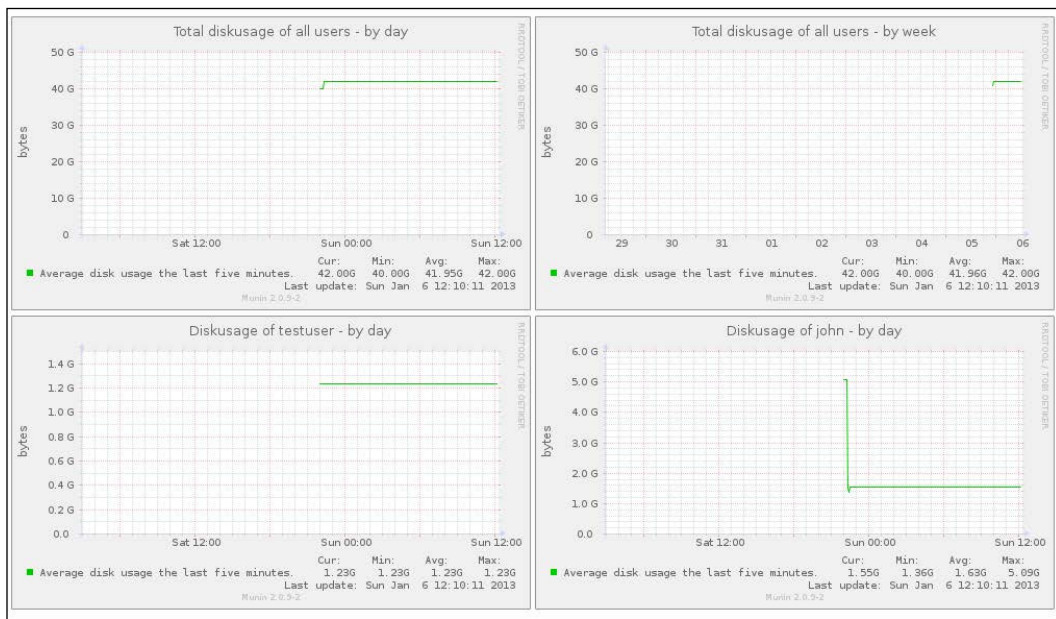
```
user_diskusage.label Average disk usage the last five minutes.  
user_diskusage.warning 2147483648  
user_diskusage.critical 5368709120  
graph_info The disk usage of testuser in bytes.  
  
multigraph user_diskusage.john  
...
```

As you might have guessed, the `user_diskusage` keyword represents the top of the hierarchy and `user_diskusage.testuser` is a down-drill of this.

The `fetch` argument still provides the same output that it normally would, but they are separated by the `multigraph` lines:

```
multigraph user_diskusage  
user_diskusage.value 42949672960  
multigraph user_diskusage.testuser  
user_diskusage.value 1323302912  
multigraph user_diskusage.john  
...
```

When the Munin master picks this up, you will get a total disk usage of all users on your dashboard. When you click it, you will be able to see the disk usage of individual users.



Challenge yourself!

Once again it is time to test your own abilities. The three following assignments get increasingly difficult:

- ◆ Try to change the `Config` subroutine so that it prints out a multigraph output like in the preceding example. Use the `autosuggest` subroutine as a starting point.
- ◆ Change the output of the `fetch` subroutine so that it prints out the same multigraph separated values, just like in the preceding example.
- ◆ `du` will probably be too slow to monitor the disk usage of all your users. Can you find and implement a replacement?

If you want to see more multigraph examples, take a look at the PyMunin project at <http://aouyar.github.com/PyMunin>. Here you can find a few very nice multigraph plugins to down-drill, PostgreSQL, Memcache, Redis, and many other systems, all written in Python.

People and places you should get to know

The official website for Munin is located at <http://munin-monitoring.org>. Here you can find a wiki with a lot of good documentation, especially on installing munin-node on diverse systems, and tips and tricks on writing your own plugins. If you want to read more about this, take a look at the following wiki pages:

- ◆ <http://munin-monitoring.org/wiki/HowToWritePlugins>
- ◆ <http://munin-monitoring.org/wiki/HowToWriteSNMPPlugins>
- ◆ http://munin-monitoring.org/wiki/Debugging_Munin_plugins
- ◆ <http://munin-monitoring.org/wiki/MultigraphSampleOutput>

Finding plugins

There used to be a munin-plugin-exchange, much like the Nagios Exchange, which helped people with finding plugins. This has been discontinued and the Munin codebase has been moved to GitHub. All of the munin code can be found on GitHub at <https://github.com/munin-monitoring>.

If you are looking for plugins for a specific piece of software that you would like to monitor, you should look at the contrib project. This project contains a lot of third-party plugins for a wide variety of applications.

If you still cannot find the plugin you need, try searching for `munin plugin` and `your application` on GitHub or Google. A lot of people write simple plugins to solve a problem they had and post it on their blog or on GitHub.

Windows support

At GitHub, you can also find a munin-node implementation for Windows. This application gathers some counters through the Win32 API and makes them available through the standard munin-node interface. This can be very handy if you need to monitor an odd Windows machine in your network. It is also possible to monitor Windows machines through SNMP.

Interesting websites

If you don't want to run a Munin master, take a look at <http://hostedmunin.com>; they provide Munin master as a service.

A lot of people are using Munin to monitor their home or office (myself included). Take a look at the Arduino power monitor at <https://github.com/hessu/arduino-powermunin> or the coffeyleft plugin at <https://github.com/ways/coffeeleft>.

Twitter has a lot of tweets, if you search for #munin, about things people are doing with Munin. There is an official Munin twitter account at @MuninMonitoring, but it mostly just tracks GitHub. Steve Schnepf, one of the main contributors, has interesting Munin tidbits here at @steve_schnepf.



Thank you for buying
Instant Munin Plugin Starter

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

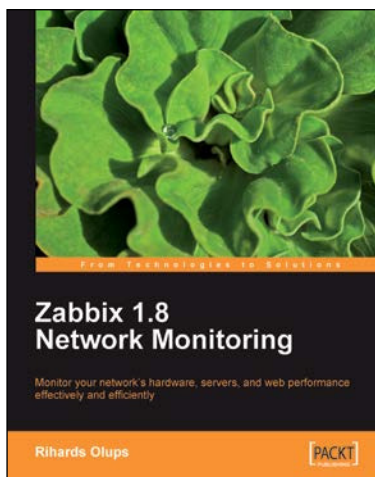
Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.packtpub.com.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

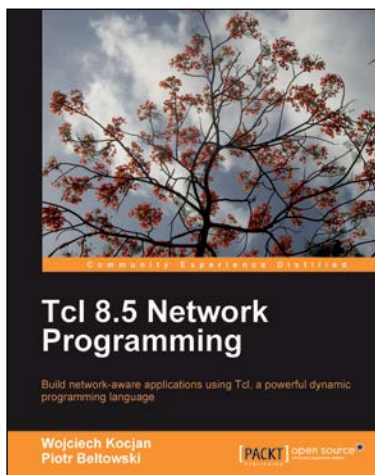


Zabbix 1.8 Network Monitoring

ISBN: 978-1-84719-768-9 Paperback: 428 pages

Monitor your network hardware, servers, and web performance effectively and efficiently

1. Start with the very basics of Zabbix, an enterprise-class open source network monitoring solution, and move up to more advanced tasks later
2. Efficiently manage your hosts, users, and permissions
3. Get alerts and react to changes in monitored parameters by sending out e-mails, SMSs, or even execute commands on remote machines
4. In-depth coverage for both beginners and advanced users with plenty of practical, working examples and clear explanations



Tcl 8.5 Network Programming

ISBN: 978-1-84951-096-7 Paperback: 588 pages

Build network-aware applications using Tcl, a powerful dynamic programming language

1. Develop network-aware applications with Tcl
2. Implement the most important network protocols in Tcl
3. Packed with hands-on-examples, case studies, and clear explanations for better understanding

Please check www.PacktPub.com for information on our titles

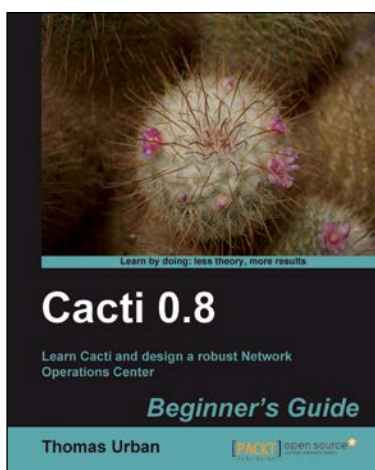


Cacti 0.8 Network Monitoring

ISBN: 978-1-84719-596-8 Paperback: 132 pages

Monitor your network with ease!

1. Install and setup Cacti to monitor your network and assign permissions to this setup in no time at all
2. Create, edit, test, and host a graph template to customize your output graph
3. Create new data input methods, SNMP, and Script XML data query
4. Full of screenshots and step-by-step instructions to monitor your network with Cacti



Cacti 0.8 Beginner's Guide

ISBN: 978-1-84951-392-0 Paperback: 348 pages

Learn Cacti and design a robust Network Operations Center

1. A complete Cacti book that focuses on the basics as well as the advanced concepts you need to know for implementing a Network Operations Center
2. A step-by-step Beginner's Guide with detailed instructions on how to create and implement custom plugins
3. Real-world examples, which you can explore and make modifications to as you go
4. Written by Thomas Urban – creator of the "Network Management Inventory Database" plugins for Cacti

Please check www.PacktPub.com for information on our titles