

F r o m T e c h n o l o g i e s t o S o l u t i o n s

Zabbix 1.8 Network Monitoring

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

Rihards Olups

PACKT
PUBLISHING

Zabbix 1.8 Network Monitoring

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

Rihards Olups



BIRMINGHAM - MUMBAI

Zabbix 1.8 Network Monitoring

Copyright © 2010 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: April 2010

Production Reference: 1220310

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 978-1-847197-68-9

www.packtpub.com

Cover Image by Vinayak Chittar (vinayak.chittar@gmail.com)

Credits

Author	Editorial Team Leader
Rihards Olups	Gagandeep Singh
Reviewers	Project Team Leader
Kris Buytaert	Lata Basantani
Renard Philippe	Project Coordinator
Acquisition Editor	Joel Goveya
Rashmi Phadnis	Proofreader
Development Editors	Lesley Harrison
Amey Kanse	Graphics
Rakesh Shejwal	Geetanjali Sawant
Technical Editor	Production Coordinator
Vinodhan Nair	Aparna Bhagat
Indexer	Cover Work
Rekha Nair	Aparna Bhagat

About the Author

Rihards Olups has over 10 years of experience in IT. He has had a chance to work with various systems, and most of that time has been spent with open source solutions. Exposure to Zabbix, one of the leading open source enterprise class monitoring solutions, was with the first public releases more than nine years ago, which has allowed to gain practical knowledge on the subject.

Previously employed by a government agency, Rihards was mostly involved in open source software deployments ranging from server to desktop grade software, with a big emphasis on Zabbix. More recently the author has joined Zabbix SIA, the company behind the software that this book is about, which has allowed him to gain even more experience with the subject.

Huge thanks to my mother, grandmother, and brother for being there, and to my Bumblebee for enduring through the process of writing the book.

Of course, thanks to the whole Zabbix team and community – there would be no subject of this book without them. Special kudos go to Alexei, who started this whole thing called Zabbix.

Thanks to the Packt team for their persistence and patience – it surely was hard to work with a chaotic person like me.

About the Reviewers

Kris Buytaert is a long time Linux and Open Source Consultant working on Linux and open source projects in Belgium, Europe, and the rest of the universe. He is currently working for Inuits.

Kris is the co-author of Virtualization with Xen, used to be the maintainer of the openMosix HOWTO, and is the author of different technical publications. He is a frequent speaker at different international conferences.

He spends most of his time working on Linux Clustering (both High Availability, Scalability, and HPC), Virtualization, and Large Infrastructure Management projects hence trying to build infrastructures that can survive the 10th floor test, better known today as "the cloud".

His blog titled "Everything is a Freaking DNS Problem" can be found at <http://www.krisbuytaert.be/blog/>.

Renard Philippe has traveled extensively during his youth, due to which he has had the opportunity to make his primary and secondary education in many countries (mostly in the Middle East, north Africa, and Europe).

The choice of higher education has brought him to the field of IT, for which he always had a passion.

Having obtained a degree in IT/ICT (with a specialization in network engineering), he has since had the opportunity to work on major projects including, among other ones, the implementation and deployment of a centralized monitoring system (for which Zabbix was chosen after analysis of multiple concurrent solutions).

Table of Contents

Preface	1
Chapter 1: Getting Started with Zabbix	7
First steps in monitoring	7
Zabbix features and architecture	9
Installation	10
Server and agent	12
Software requirements	13
Hardware requirements	13
Getting the source	14
Compilation	14
Initial configuration	15
Creating and populating the database	16
Starting up	17
SUSE Linux Enterprise Server	18
Slackware	27
Verifying the service's state	31
The Web frontend	32
Prerequisites and setting up the environment	32
Installation of the web frontend	33
Step 1 – Welcome	33
Step 2 – Licence	34
Step 3 – PHP prerequisites	34
Step 4 – Database access.	36
Step 5 – Zabbix server details	37
Step 6 – Summary	38
Step 7 – Writing the configuration file	38
Step 8 – Configuration file in place	40
Step 9 – Finishing the wizard	40
Step 10 – Logging in	41
Summary	43

Table of Contents

Chapter 2: Getting Your First Notification	45
Exploring the frontend	45
Monitoring quickstart	48
Creating a host	50
Creating an item	52
Introducing simple graphs	54
Creating triggers	58
Configuring e-mail parameters	60
Creating an action	62
Information flow in Zabbix	63
Let's create some load	64
Basic item configuration	66
Monitoring categories	67
Availability	67
Performance	67
Security	67
Management	68
Efficiency	68
Item types	68
How items can be monitored	70
Summary	72
Chapter 3: Monitoring with Zabbix Agents and Basic Protocols	73
Using Zabbix agent	73
Passive items	75
Cloning items	81
Active items	82
Supported items	93
Simple checks	93
Setting up ICMP checks	95
Tying it all together	97
Positional parameters for item descriptions	97
Using mass update	98
Value mapping	100
Copying items	102
Summary	106
Chapter 4: Monitoring SNMP and IPMI Devices	107
Simple Network Management Protocol	107
Using Net-SNMP	108
Using SNMPv3 with Net-SNMP	112
Adding new MIBs	113
Working with SNMP items in Zabbix	115
Translating SNMP OIDs	119

Table of Contents

Dynamic indexes	119
Receiving SNMP traps	123
Trap handling schemes	129
Intelligent Platform Management Interface	136
Dell Remote Access Controller	136
Preparing Zabbix for IPMI querying	136
Configuring DRAC IPMI access	137
Setting up IPMI items	138
Card attached to one of the already monitored hosts	139
Card attached to a different host	139
Creating IPMI item	140
Summary	142
Chapter 5: Managing Hosts, Users, and Permissions	143
Host and host groups	143
Users, user groups, and permissions	149
Authentication methods	149
Creating a user	150
Creating user groups	156
Summary	164
Chapter 6: Acting Upon Monitored Conditions	165
Triggers	165
Trigger dependencies	168
Constructing trigger expressions	173
Triggers that time out	177
Human-readable constants	177
Event details	177
Event generation and hysteresis	178
Actions	180
Limiting conditions when actions are sent	180
Additional action conditions	182
Dependencies and actions	182
Per media limits	183
Sending out notifications	184
Using macros	185
Escalating things	187
Integration with issue management systems	196
Bugzilla	196
CA Unicenter Service Desk	197
Using scripts as media	197
Remote commands	199
Summary	201

Table of Contents

Chapter 7: Simplifying Complex Configuration with Templates	203
Identifying template candidates	203
Creating a template	204
Linking templates to hosts	206
Changing configuration in template	211
Macro usage	212
Using multiple templates	214
Unlinking templates from hosts	216
Nested templates	217
Summary	220
Chapter 8: Visualizing the Data	221
Visualize what?	221
Single elements	222
Graphs	222
Simple graphs	222
Custom graphs	223
Maps	237
Creating a map	238
Linking map elements	241
Further map customization	245
Compound elements	250
Screens	250
Dynamic screens	253
Slide shows	256
Showing data on a big display	257
Challenges	257
Non-interactive display	257
Information overload	258
Displaying a specific section automatically	258
Recent change flashing	259
Summary	260
Chapter 9: Creating Reports	261
Simple reports	261
Status of Zabbix	261
Availability report	263
Most busy triggers top 100	264
Bar reports	265
Distribution of values for multiple periods	266
Distribution of values for multiple items	269
Comparing values for multiple periods	273
Summary	277

Table of Contents

Chapter 10: Advanced Item Monitoring	279
Aggregate items	279
External checks	282
User parameters	287
Just getting it to work	287
Querying data that Zabbix agent does not support	288
Flexible user parameters	289
Level of the details monitored	291
Environment trap	293
Things to remember about user parameters	296
Wrapper scripts	296
Other methods to gather data	297
Sending in the data	297
Using custom agents	300
Summary	301
Chapter 11: Monitoring Windows and Web Pages	303
Monitoring web pages	303
Creating web monitoring scenario	303
Windows-specific monitoring	310
Installing Zabbix agent for Windows	310
Querying performance counters	314
Using numeric references to performance counters	315
Using aliases for performance counters	318
Monitoring Windows services	318
Checking whether an automatic service has stopped	320
Summary	321
Chapter 12: Using Proxies to Monitor Remote Locations	323
When proxies are useful	323
Setting up the proxy	325
Monitoring a host through a proxy	327
Proxy benefits	329
Proxy reliability	331
Tweaking proxy configuration	333
Summary	334
Chapter 13: Working Closely with Data	335
Getting raw data	335
Extracting from the frontend	335
Querying the database	337
Using data in a remote site	340

Table of Contents

Diving further in the database	342
Managing users	342
Converting a host to a template	345
Changing existing data	346
Finding out "when"	347
"When" in computer language	347
Finding out what	347
Performing the change	347
Using XML import/export for configuration	348
Exporting initial configuration	348
Modifying configuration	348
XML export format	349
Script around the export	350
Importing modified configuration	351
Summary	352
Chapter 14: Upgrading Zabbix	353
General policy	353
Zabbix versions	353
Version upgrades	354
Upgrading Zabbix	354
Change level upgrade	354
Adding the indexes	355
Replacing frontend files	356
Minor or major level upgrades	357
Patching the database	358
Frontend configuration file	361
Compatibility	361
Summary	362
Chapter 15: Taking Care of Zabbix	363
Internal items	363
Performance considerations	368
Reducing the query count	369
Increasing write performance	370
Who did that?	372
Real men make no backups	374
Backing up the database	374
Restoring from backup	376
Separating configuration and data backups	377
Summary	378

Table of Contents

Appendix A: Troubleshooting	381
Installation	381
Compilation	381
Frontend	383
Starting services	383
Frontend	383
Locked out of the frontend	385
Problems with monitoring	386
General monitoring	386
Monitoring with Zabbix agent	386
User parameters	388
Problems with SNMP devices	388
Problems with IPMI monitoring	389
Problems with ICMP checks	389
General issues	389
Triggers	390
Actions	390
Appendix B: Being Part of the Community	391
Community and support	391
Using the Zabbix forum	392
Editing the wiki	392
Chatting on IRC	393
Filing issues on the tracker	394
Following the development	394
Getting the source	395
Daily snapshots	395
Accessing the version control system	396
Commercial support options	400
Summary	401
Index	403

Preface

Imagine you're celebrating the start of the weekend with Friday-night drinks with a few friends. And then suddenly your phone rings – one of the servers you administer has gone down, and it needs to be back up before tomorrow morning. So you drag yourself back to the office, only to discover that some logfiles have been growing more than usual over the past few weeks and have filled up the hard drive.

While the scenario above is very simplistic, something similar has happened to most IT workers at one or another point in their careers. To avoid such situations this book will teach you to monitor your network's hardware, servers, and web performance using Zabbix - an open source system monitoring and reporting solution.

What this book covers

In *Chapter 1, Getting Started with Zabbix*, we'll cover Zabbix installation from scratch, including the initial database, server and agent daemons, and web frontend, all running on the same machine and configure the Zabbix web frontend, using PHP to access the database.

Chapter 2, Getting Your First Notification, will cover configuring Zabbix using the frontend to set up data gathering, triggering upon specified conditions, and informing us by sending an e-mail for a single data source.

In *Chapter 3, Monitoring with Zabbix Agents and Basic Protocols*, we'll set up the most widely used and basic data gathering methods – Zabbix agents and simple checks such as ICMP ping and direct TCP service checking.

In *Chapter 4, Monitoring SNMP and IPMI Devices*, we'll learn how to set up industry standard monitoring protocols, SNMP and IPMI, for both polling by Zabbix and receiving SNMP traps, which will allow us to monitor a large portion of devices, including printers, switches, UPSes, routers, and others.

Chapter 5, Managing Hosts, Users, and Permissions, will cover hosts, users, and permissions, including host and user group functionality and their impact on permissions.

In *Chapter 6, Acting Upon Monitored Conditions*, we'll look at ways to define which conditions are noteworthy by configuring triggers and how to react to such conditions by sending e-mail, launching an external script, opening a report in a separate bug tracker, or even restarting a faulty service. We will also learn to configure escalations in Zabbix and figure out how hysteresis works.

In *Chapter 7, Simplifying Complex Configuration with Templates*, we'll learn that we did it all wrong before and improve our configuration by using templates that allow us to apply uniform configuration to a bunch of hosts. We'll also explore template nesting which allows creating very flexible configuration in a large and mixed environment.

In *Chapter 8, Visualizing the Data*, we'll create visual elements to display the gathered data, including several types of graphs, interactive network maps, screens that collect various types of elements to display, and slideshows that allow cycling through several screens in an automated fashion.

In *Chapter 9, Creating Reports*, we'll use the built-in reporting capabilities of Zabbix such as status of Zabbix, availability reports, most often happening problems reports, and the heavily configurable bar reports.

In *Chapter 10, Advanced Item Monitoring*, we'll find out about more advanced ways to gather information by using external, aggregate, and custom item types to retrieve basically any information.

In *Chapter 11, Monitoring Windows and Web Pages*, we'll set up some Windows monitoring by installing Zabbix agent and using performance counters, as well as get to monitoring accessibility, performance, and availability of web pages.

In *Chapter 12, Using Proxies to Monitor Remote Locations*, we'll explore usage of proxies that collect the data on behalf of the Zabbix server and then transmit it back to the server, which helps with remote locations that can't be accessed directly because of firewall concerns and also reduces load on the Zabbix server.

In *Chapter 13, Working Closely with Data*, we'll figure out some details on how data is stored in the Zabbix database and how we can interact with it directly, as well as use Zabbix's native XML import and export functionality to more easily create large amounts of configuration.

In *Chapter 14, Upgrading Zabbix*, we'll learn about the Zabbix upgrade procedure, how different components of various versions can interact and what database patching between versions involves.

In *Chapter 15, Taking Care of Zabbix*, we'll look in more detail at the Zabbix setup itself and check out what internal health and performance metrics we can use, what simple first steps we can take to improve performance, and what internal logging and auditing options are available.

In *Appendix A, Troubleshooting*, we'll look at common pitfalls with installation, connectivity, configuration, and other areas.

In *Appendix B, Being Part of the Community*, we'll find out that we are not alone and there's a community around the Zabbix monitoring solution, which we can reach via forums, IRC, and the wiki.

Who this book is for

This book assumes no experience with Zabbix and minimal experience with Linux. The knowledge provided by this book, will be useful if:

- You are responsible for managing in-house IT infrastructure such as network hardware, servers, and web pages
- You are responsible for managing a non-IT infrastructure that provides data such as temperature, flow, and other readings
- You have clients with strict accessibility requirements and want to monitor the hardware that provides services to them
- You are a system administrator who wants to monitor their network hardware, servers, and web performance

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "If you see a file and directory listing instead of the installation wizard, make sure you have added `index.php` to `DirectoryIndex` directive."

A block of code will be set as follows

```
zagent_start() {
    if [ -x $BINLOCATION/zabbix_agentd ]; then
        if processcheck zabbix_agentd; then
            echo "Zabbix agent daemon already running"
        else
            echo "Starting zabbix agent daemon: $BINLOCATION/zabbix_agentd"
            $BINLOCATION/zabbix_agentd
        fi
    else
        echo "Executable $BINLOCATION/zabbix_agentd not present"
        fi
}
```

Any command-line input and output is written as follows:

```
# useradd -m -s /bin/bash zabbix
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "clicking the **Next** button moves you to the next screen".



Warnings or important notes appear in a box like this.



Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book – what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on www.packtpub.com or e-mail suggest@packtpub.com.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book on, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code for the book



Visit http://www.packtpub.com/files/code/7689_Code.zip to directly download the example code.

The downloadable files contain instructions on how to use them.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or the code – we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/support>, selecting your book, clicking on the **let us know** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1

Getting Started with Zabbix

It's Friday night, and you are at a party outside the city with old friends. After a few beers it looks like this is going to be a great party, when suddenly your phone rings. A customer can't access some critical server that absolutely has to be available as soon as possible. You try to ssh in the server, only to discover that customer is right – it can't be accessed.

As driving after those few beers would quite likely lead to inoperable server for quite some time, you get a taxi (expensive because of the distance. While many modern systems have out-of-bands management cards installed that might have helped a bit in such a situation, our hypothetical administrator does not have one available). After arriving at the server room, you find out that some logfiles have been growing more than usual over the past few weeks and have filled up the hard drive.

While the scenario above is very simplistic, something similar has probably happened to most IT workers at one or another point in their careers. Most implemented a simple system monitoring and reporting solution soon after that.

We will learn to set up and configure one such monitoring system – Zabbix.

First steps in monitoring

Situations similar to the one described above, are actually more common than desired. A system fault that had no symptoms visible before is relatively rare. Probably a subsection of Unix Administration Horror Stories (visit <http://www-uxsup.csx.cam.ac.uk/misc/horror.txt>) could be easily compiled that contained only stories about faults that were not noticed on time.

As experience shows, problems tend to happen when we are least equipped to solve them. To work with them on our terms we turn to a class of software, commonly referred to as **network monitoring** software. Such software usually allows us to constantly monitor things happening in a computer network using one or more methods and notify the persons responsible if some metric passes a defined threshold.

One of the first monitoring solutions most administrators implement is a simple shell script, invoked from crontab, that checks some basic parameters like disk usage or some service state, like Apache server. As the server and monitored parameter count grows, a neat and clean script systems starts to grow into a performance-hogging script hairball that costs more time in upkeep than it saves. While do-it-yourself crowds claim that nobody needs dedicated software for most tasks (monitoring included), most administrators will disagree as soon as they have to add switches, UPSes, routers, IP cameras, and a myriad of other devices to the swarm of monitored objects.

So what basic functionality can one expect from a monitoring solution? They are as follows:

- **Data gathering:** This is where everything starts. Usually data will be gathered using various methods, including SNMP, agents, IPMI, and others.
- **Alerting:** Gathered data can be compared to thresholds and alerts sent out when needed using different channels, like e-mail or SMS.
- **Data storage:** Once we have gathered the data it doesn't make sense to throw it away, so we will often want to store it for later analysis.
- **Visualization:** Humans are better at distinguishing visualized data than raw numbers, especially when there are huge amounts of them. As we have data already gathered and stored, it is trivial to generate simple graphs from it.

Sounds simple? That's because it is. But then we start to desire more features like easy and efficient configuration, escalations, permission delegation, and so on. If we sit down and start listing the things we want to keep an eye out for, it may turn out that area of interest extends beyond the network – for example, a hard drive that has SMART errors logged, an application that has too many threads, or a UPS that has one phase overloaded. It is much easier to manage monitoring of all these different problem categories from a single configuration point.

In the quest for a manageable monitoring system wondrous adventurers stumbled upon collections of scripts much like the way they implemented themselves, obscure and not so obscure workstation-level software, and heavy, expensive monitoring systems from big vendors.

Another group is open source monitoring systems that have various sophistication levels, one of which is Zabbix.

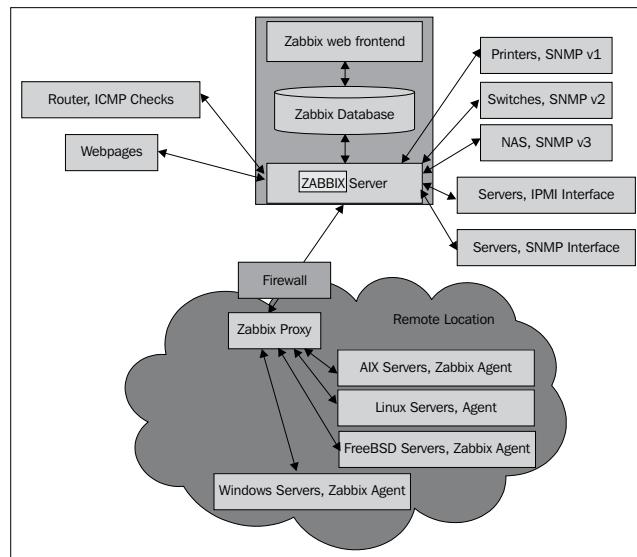
Zabbix features and architecture

Zabbix provides many ways to monitor different aspects of your IT infrastructure and indeed, almost anything one might want to hook to it. It can be characterized as a semi-distributed monitoring system with centralized management. While many installations have a single central database, it is possible to use distributed monitoring with nodes and proxies, and most installations will use Zabbix agents.

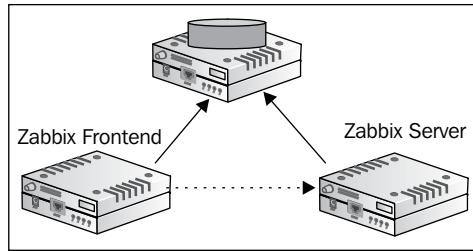
So what features does Zabbix provide? They are:

- Centralized, easy to use web interface
- Server that runs on most Unix-like operating systems, including Linux, AIX, FreeBSD, OpenBSD, and Solaris
- Native agents for most Unix-like operating systems and Microsoft Windows versions
- Ability to directly monitor SNMP (v1, 2, and 3) and IPMI devices
- Built-in graphing and other visualization capabilities
- Notifications that allow for easy integration with other systems
- Flexible configuration, including templating
- And a lot of other features that would allow you to implement a sophisticated monitoring solution

If we look at a simplified network from the Zabbix perspective, placing Zabbix server at the center, the communication of the various monitoring aspects matters. The following image depicts a relatively simple Zabbix setup with several of the monitoring capabilities used and different device categories connected.



Our central object is the **Zabbix database**, with several backends supported. **Zabbix server**, written in C, and web frontend written in PHP, can both reside on the same machine or on another server. When running each component on a separate machine, both the **Zabbix server** and the frontend need access to the database, and frontend optionally needs access to **Zabbix server** to show server status. Required connection directions are depicted by arrows in the following image.



Zabbix server directly monitors multiple devices, but a remote location is separated by a firewall, so it gathers data through a Zabbix proxy. Zabbix proxy and agents, just like the server, are written in C.

While it is perfectly fine to run all three server components on a single machine, there might be good reasons to separate them, like taking advantage of an existing high performance database or web server.

In general, monitored devices have little control over what is monitored – most of the configuration is centralized. Such an approach seriously reduces the capabilities of single misconfigured system to bring down the whole monitoring setup.

Installation

Alright, enough with the dry-talk, what will we get? Let's look at dashboard screen of Zabbix web frontend, showing only a very basic configuration.

The screenshot shows the Zabbix 1.8.1 Personal Dashboard. At the top, there's a navigation bar with links for Monitoring, Inventory, Reports, Configuration, Administration, Help, Get support, Print, Profile, and Logout. Below the navigation is a breadcrumb trail: History > History > Overview > Dashboard > Hosts > Dashboard. The main area is titled "PERSONAL DASHBOARD". It features several sections:

- Status of Zabbix:** A table showing various metrics like hosts running, items monitored, triggers enabled, and users online.
- System status:** A table showing the status of host groups (e.g., Linux servers, Zabbix Servers) across categories: Disaster, High, Average, Warning, Information, and Not classified.
- Host status:** A table showing the count of hosts without problems, with problems, and total for Linux servers and Zabbix Servers.
- Last 20 issues:** A table listing recent problems (issues) with columns for host, issue, last change, age, ack status, and actions.
- Web monitoring:** A table showing the status of host groups across categories: Ok, Failed, In progress, and Unknown.

At the bottom, it says "Zabbix 1.8.1 Copyright 2001-2009 by SIA Zabbix" and "Connected as 'Admin'".

As we can see, Zabbix dashboard shows a high level overview of overall monitored system status, status of Zabbix, some of the most recent problems, and a few more things. This particular dashboard shows a very tiny Zabbix setup. Eventually your Zabbix installation will grow and provide monitoring of different devices, including servers of various operating systems, different services and hardware state on those servers, network devices, UPSes, web pages, other components of IT, and other infrastructure.

The frontend will provide various options for visualizing data, starting from problem lists and simple graphs ending with network maps and reports, while backend will work hard to provide information that this visualization is based on and send out alerts. All of this will require some configuration that we will learn to perform along the course of this book.

Before we can configure Zabbix, we need to install it. Usually you'll have two choices—either installing from distribution packages, or setting it up from the source code. Unless you highly value distribution packaging and are not ready to roll your own packages, it is suggested to set up the latest version from sources, because Zabbix is being developed at a relatively noticeable pace and there's always some neat feature in the next version that makes life easier.

At first we will set up Zabbix server, database, and frontend, all running on the same machine and using a MySQL database.

If you decide to install Zabbix from your distribution packages, installation procedure and package naming schemes will differ. Refer to your distribution's documentation for that information.

There are a few benefits to using distribution packages. These include:

- Automated installation and updating
- Dependencies usually sorted out

Compiling from source also has its share of benefits. They are:

- Newer versions with more features and improvements
- More fine-grained control over compiled-in functionality

Server and agent

The most widely-used Zabbix architecture is a server that queries agents. That's what we will learn to set up so that we can monitor our test system.

As with most software, there are some prerequisites that we will need to run Zabbix components. That includes requirements for hardware and other software that the Zabbix server and agent depend on. For the purpose of these instructions, we will settle on running Zabbix on Linux, using a MySQL database. The specific Linux distribution does not matter much—it's best to choose the one you are most familiar with.

Software requirements

Now we should get to compiling the various components of Zabbix, so make sure to install the minimum required packages to get Zabbix working with MySQL. They are:

- GCC
- Automake
- MySQL (<http://www.mysql.com/>)

Depending on your distribution and the desired functionality you might also need some or all of the following packages:

- zlib-devel
- mysql-devel (for MySQL support)
- glibc-devel
- curl-devel (for web monitoring)
- libidn-devel (curl-devel might depend on it)
- openssl-devel (curl-devel might depend on it)
- net-snmp-devel (for SNMP support)
- popt-devel (net-snmp-devel might depend on it)
- rpm-devel (net-snmp-devel might depend on it)
- OpenIPMI-devel (for IPMI support)
- libssh2-devel (for direct SSH checks)

Hardware requirements

Hardware requirements vary wildly depending on the configuration. It is impossible to give definite requirements, so any production installation should evaluate them individually. For our test environment, though, even as little RAM as 128 MB should be enough. CPU power in general won't play a huge role; Pentium II class hardware should be perfectly capable of dealing with it, although generating graphs with many elements or other complex views can require more powerful hardware to operate at an acceptable speed. You can take these as a starting point as well when installing in a virtual machine.

Of course, the more resources you give to Zabbix, the snappier and happier it will be.

Getting the source

There are several ways to download the source of Zabbix. You can get the source code from a SVN repository, which will be discussed in *Appendix B*, however for this installation procedure it is suggested to download version 1.8.1 from the Zabbix homepage; <http://www.zabbix.com/>. While it should be possible to use latest stable version, using 1.8.1 will allow to follow instructions more closely. Go to the **Download** section and grab the compressed source package. Usually only the latest stable version is available on the downloads page, so you might have to browse the source archives, though do not take development or beta version, which might be available.

To ease further references, it is suggested that you choose a directory to work in, for example, `~/zabbix` (`~` being your home directory). Download the archive into this directory.

Compilation

Once the archive has finished downloading, open a terminal and extract it:

```
$ cd ~/zabbix; tar -zxvf zabbix-1.8.1.tar.gz
```

It is suggested that you install the requirements and compile Zabbix with external functionality right away so that you don't have to recompile as we progress.

For the purpose of this book, we will compile Zabbix with server, agent, MySQL, curl, SNMP, and IPMI support.

To continue, enter the following in the terminal:

```
$ cd zabbix-1.8.1  
$ ./configure --enable-server --with-mysql --with-net-snmp --with-libcurl  
--with-openipmi --enable-agent
```

In the end, a summary of compiled components is printed. Verify that you have the following enabled:

Enable server:	yes
With database:	MySQL
WEB Monitoring via:	CURL
SNMP:	net-snmp
IPMI:	openipmi
Enable agent:	yes

If configure completes successfully, it's all good. If it fails, check the error messages printed in the console and verify that all prerequisites are installed. A file named `config.log` might provide more detail on the errors. If you can't find out what's wrong, check Appendix A, *Troubleshooting*, which lists some common compilation problems.

To actually compile Zabbix, issue the next command.

```
$ make
```

You can get a cup of tea, but don't expect to have much time—Zabbix compilation doesn't take too long, even an old 350 MHz Pentium II compiles it in approximately five minutes. After the `make` process has finished, check the last lines for any error messages. If there are none, congratulations, you have successfully compiled Zabbix.

Now we should install it. Despite the output of `configure` and many How Tos suggesting so, do not run `make install`. It unnecessarily complicates package management and can lead to weird problems in future. Instead, you should create proper packages for your distribution. This process is package system specific, learning them all would be cumbersome, but there's a universal solution—software named **CheckInstall** (<http://www.asic-linux.com.mx/~izto/checkinstall/>). Packages should be available for most distributions, and it supports the creation of Slackware, RPM, and Debian packages.

To create a proper Zabbix package make sure that CheckInstall is installed and execute as root:

```
# checkinstall --nodoc --install=yes -y
```

This will create and install a package that you will later be able to remove using your system's package management tool. The created package's location depends on your distribution—for Slackware, it's the directory you executed CheckInstall from, for RPM-based distributions it is usually `/usr/src/packages/RPMS/<architecture>`.

Initial configuration

After compilation, we have to configure some basic parameters for the server and agent. There are example configuration files provided with the Zabbix package, so let's use those. Again, as root execute:

```
# mkdir /etc/zabbix
# cp misc/conf/{zabbix_server.conf,zabbix_agentd.conf} /etc/zabbix
```

To configure the Zabbix agent, we don't have to do anything. The default configuration will do just fine for now. That was easy, right?

For the server we will need to make some changes. Open `/etc/zabbix/zabbix_server.conf` in your favorite editor (you will need to run it as root) and find the following entries in the file:

- DBName
- DBUser
- DBPassword

`DBName` should be `zabbix` by default, and we can leave it as is. `DBUser` is set to `root`, and we don't like that, so let's change it to `zabbix`. For `DBPassword`, choose any password. You won't have to remember it, so be creative.

After we insert a password into this configuration file, we should restrict the file's permissions so that the password is not that easy to obtain. This is done as shown below:

```
# chmod 400 /etc/zabbix/zabbix_server.conf  
# chown zabbix /etc/zabbix/zabbix_server.conf
```

Creating and populating the database

For the Zabbix server to store data, we have to create a database. Start a MySQL client.

```
$ mysql -u root -p
```

Enter the root user password for MySQL (you would have set this during the installation of MySQL, or the password could be something that is the default for your distribution). If you do not know the password, you can try omitting `-p`. This switch will tell the client to attempt to connect with an empty password.

Now let's create the database. Add the user that Zabbix would connect to the database as, and grant necessary permissions to this user:

```
mysql> create database zabbix character set utf 8;  
Query OK, 1 row affected (0.01 sec)  
  
mysql> grant all privileges on zabbix.* to 'zabbix'@'localhost'  
identified by 'mycreativepassword';  
  
Query OK, 0 rows affected (0.12 sec)
```

Use the same password you set in `zabbix_server.conf` file instead of `mycreativepassword`.

Quit the MySQL client by entering the following command:

```
mysql> quit
```

Let's populate the newly-created database with Zabbix schema and initial data.

```
$ mysql -u zabbix -p zabbix < create/schema/mysql.sql  
$ mysql -u zabbix -p zabbix < create/data/data.sql
```

Next, let's insert the images to be used in network maps. While these images are not required for basic functionality, we'll want to create some nice looking network maps later.

```
$ mysql -u zabbix -p zabbix < create/data/images_mysql.sql
```

All three importing processes should complete without any messages. If there are any errors, review the messages, fix the issue, and retry the failed operation. Note, if the import is interrupted in the middle of the process, you might have to clear the database—easiest way to do that is deleting the database by typing:

```
mysql> drop database zabbix;  
Query OK, 0 rows affected (0.00 sec)
```

Be careful not to delete any database with important information! After deleting the Zabbix database recreate it and assign the correct user permissions, as indicated above.

By now we should have Zabbix server and agent installed, and ready to start.

Starting up

You should never start the Zabbix server or agent as root, which is common sense for most daemon processes, so let's create user to run these processes. You can use tools provided by distribution, or the most widely available command – useradd, which we need to execute as root:

```
# useradd -m -s /bin/bash zabbix
```

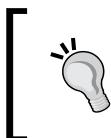
This will create user named zabbix with a home directory in the default location (usually /home/zabbix) and shell at /bin/bash, which will be needed for advanced parameters later on.

For the first startup of both server and agent let's try the direct approach – as root execute:

```
# /usr/local/sbin/zabbix_agentd
```

This will start the Zabbix agent daemon, which should start up silently and daemonize. If the above command produces errors, resolve those before proceeding. If it succeeds, continue by starting the Zabbix server.

```
# /usr/local/sbin/zabbix_server
```



We are using `zabbix_agentd` that runs as a daemon. While there's also the `zabbix_agent` executable that provides an option to be run within `inetd`, it does not support active items and in most cases will have worse performance than the agent daemon.



If you decided to install distribution packages of Zabbix above binaries will most likely be located in a different directory, but they should be in your path—so try running the agent and server without specifying a directory name.



The latest versions of Zabbix, including 1.8, automatically drop root privileges on startup and run as the `zabbix` user.



While it's nice to have Zabbix running, that's hardly a process one expects to do manually upon each system boot, so the server and agent should be added to your system's startup sequence. This is fairly distribution specific, so all possible variations can't be discussed here. Instead, examples for SUSE Linux Enterprise Server and Slackware startup scripts will be provided.

SUSE Linux Enterprise Server

The Zabbix package provides example startup scripts, but they are somewhat outdated. Here are init scripts for Zabbix server and agent daemons, based on the SUSE Linux Enterprise Server 10 SP2 skeleton file. It should be possible to use these files on most other distributions that have SysV-style init scripts. For Zabbix server, place the following in the `/etc/init.d/zabbix_server` file:

```
#!/bin/sh
#
# Original skeleton file from SLES10 Sp2 -
# Copyright (C) 1995--2005 Kurt Garloff, SUSE / Novell Inc.
# Zabbix specifics -
# Copyright (C) 2008--2009 Richlv
#
# This library is free software; you can redistribute it and/or
# modify it under the terms of the GNU Lesser General Public
# License as published by the Free Software Foundation;
# either version 2.1 of the License, or (at your option)
# any later version.
```

```
# This library is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
# See the GNU Lesser General Public License for more details.

# You should have received a copy of the GNU Lesser General Public
# License along with this library; if not, write to the
# Free Software Foundation, Inc., 59 Temple Place, Suite 330,
# Boston, MA 02111-1307, USA.

# /etc/init.d/zabbix_server
#   and its symbolic link
# /usr/sbin/rczabbix_server
#
### BEGIN INIT INFO
# Provides:          zabbix_server
# Required-Start:    $network $remote_fs $local_fs
# Should-Start:      mysql postgresql
# Required-Stop:     $network $remote_fs $local_fs
# Should-Stop:       mysql postgresql
# Default-Start:    3 5
# Default-Stop:      0 1 2 6
# Short-Description: Zabbix monitoring server daemon
# Description:       This is a server daemon for the monitoring
#                    system Zabbix
#                    For more information see http://www.zabbix.com
### END INIT INFO

ZABBIX_CONFIG="/etc/zabbix/zabbix_server.conf"

test -r $ZABBIX_CONFIG || { echo "$ZABBIX_CONFIG missing";
  if [ "$1" = "stop" ]; then exit 0;
  else exit 6; fi; }

ZABBIX_BIN="/usr/local/sbin/zabbix_server"

test -x $ZABBIX_BIN || { echo "$ZABBIX_BIN not installed";
  if [ "$1" = "stop" ]; then exit 0;
  else exit 5; fi; }

. /etc/rc.status

# Reset status of this service
rc_reset
```

Getting Started with Zabbix

```
NAME="Zabbix server daemon"
ZABBIX_PID=/var/tmp/zabbix_server.pid

case "$1" in

    start)
        echo -n "Starting $NAME "
        ## Start daemon with startproc(8). If this fails
        ## the return value is set appropriately by startproc.
        /sbin/startproc -t 1 -p $ZABBIX_PID $ZABBIX_BIN

        # Remember status and be verbose
        rc_status -v
        ;;

    stop)
        echo -n "Shutting down $NAME "
        ## Stop daemon with killproc(8) and if this fails
        ## killproc sets the return value according to LSB.

        /sbin/killproc -TERM $ZABBIX_BIN

        # Remember status and be verbose
        rc_status -v
        ;;

    try-restart|condrestart)
        ## Do a restart only if the service was active before.
        ## Note: try-restart is now part of LSB (as of 1.9).
        ## RH has a similar command named condrestart.
        if test "$1" = "condrestart"; then
            echo "${attn} Use try-restart ${done}(LSB)${attn} rather than
                  condrestart ${warn}(RH)${norm}"
        fi
        $0 status
        if test $? = 0; then
            $0 restart
        else
            rc_reset # Not running is not a failure.
        fi
        # Remember status and be quiet
        rc_status
        ;;

    restart)
        ## Stop the service and regardless of whether it was
        ## running or not, start it again.
        $0 stop
```

```
$0 start

# Remember status and be quiet
rc_status
;;
    force-reload)
## Zabbix server daemon does not support configuration reloading,
    thus it is restarted, if running.

echo -n "Reload service $NAME "

$0 try-restart
rc_status
;;
    reload)
## Zabbix server daemon does not support configuration reloading,
    thus reload fails

rc_failed 3
rc_status -v
;;
    status)
echo -n "Checking for service $NAME "
## Check status with checkproc(8), if process is running
## checkproc will return with exit status 0.

# Return value is slightly different for the status command:
# 0 - service up and running
# 1 - service dead, but pid file exists
# 2 - service dead, but /var/lock/ lock file exists
# 3 - service not running (unused)
# 4 - service status unknown :-(

# 5--199 reserved (5--99 LSB, 100--149 distro, 150--199 appl.)

# NOTE: checkproc returns LSB compliant status values.
/sbin/checkproc -p $ZABBIX_PID $ZABBIX_BIN
# NOTE: rc_status knows that we called this init script with
# "status" option and adapts its messages accordingly.
    rc_status -v
;;
    probe)
## Optional: Probe for the necessity of a reload, print out the
## argument to this init script which is required for a reload.
## Note: probe is not (yet) part of LSB (as of 1.9)
```

```
test $ZABBIX_CONFIG -nt $ZABBIX_PID && echo reload
;;
*)
echo "Usage: $0 {start|stop|status|try-restart|restart|force-
      reload|reload|probe}"
exit 1
;;
esac
rc_exit
```

For the Zabbix agent daemon, create `/etc/init.d/zabbix_agentd` with the following contents:

```
#!/bin/sh
#
# Original skeleton file from SLES10 Sp2 -
# Copyright (C) 1995--2005 Kurt Garloff, SUSE / Novell Inc.
# Zabbix specifics -
# Copyright (C) 2008--2009 Richlv
#
# This library is free software; you can redistribute it and/or
# modify it under the terms of the GNU Lesser General Public
# License as published by the Free Software Foundation; either
# version 2.1 of the License, or (at your option) any
# later version.
#
# This library is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
# See the GNU Lesser General Public License for more details.
#
# You should have received a copy of the GNU Lesser General Public
# License along with this library; if not, write to the
# Free Software Foundation, Inc., 59 Temple Place, Suite 330,
# Boston, MA 02111-1307, USA.
#
# /etc/init.d/zabbix_agentd
#   and its symbolic link
# /usr/sbin/rczabbix_agentd
#
### BEGIN INIT INFO
# Provides:          zabbix_agentd
# Required-Start:    $network $remote_fs $local_fs
# Required-Stop:     $network $remote_fs $local_fs
# Default-Start:    3 5
```

```
# Default-Stop:      0 1 2 6
# Short-Description: Zabbix monitoring agent daemon
# Description:      This is a client daemon for the monitoring
#                   system Zabbix
#                   For more information see http://www.zabbix.com
### END INIT INFO

ZABBIX_CONFIG="/etc/zabbix/zabbix_agentd.conf"

test -r $ZABBIX_CONFIG || { echo "$ZABBIX_CONFIG missing";
    if [ "$1" = "stop" ]; then exit 0;
    else exit 6; fi; }

ZABBIX_BIN="/usr/local/sbin/zabbix_agentd"

test -x $ZABBIX_BIN || { echo "$ZABBIX_BIN not installed";
    if [ "$1" = "stop" ]; then exit 0;
    else exit 5; fi; }

. /etc/rc.status

# Reset status of this service
rc_reset

NAME="Zabbix agent daemon"
ZABBIX_PID=/var/tmp/zabbix_agentd.pid

case "$1" in

    start)
    echo -n "Starting $NAME "
    ## Start daemon with startproc(8). If this fails
    ## the return value is set appropriately by startproc.
    /sbin/startproc -t 1 -p $ZABBIX_PID $ZABBIX_BIN

    # Remember status and be verbose
    rc_status -v
    ;;

    stop)
    echo -n "Shutting down $NAME "
    ## Stop daemon with killproc(8) and if this fails
    ## killproc sets the return value according to LSB.

    /sbin/killproc -TERM $ZABBIX_BIN
```

```
# Remember status and be verbose
rc_status -v
;;
    try-restart|condrestart)
## Do a restart only if the service was active before.
## Note: try-restart is now part of LSB (as of 1.9).
## RH has a similar command named condrestart.
if test "$1" = "condrestart"; then
    echo "${attn} Use try-restart ${done} (LSB)${attn} rather than
        condrestart ${warn} (RH)${norm}"
fi
$0 status
if test $? = 0; then
    $0 restart
else
    rc_reset # Not running is not a failure.
fi
# Remember status and be quiet
rc_status
;;
    restart)
## Stop the service and regardless of whether it was
## running or not, start it again.
$0 stop
$0 start

# Remember status and be quiet
rc_status
;;
    force-reload)
## Zabbix agent daemon does not support configuration reloading,
    thus it is restarted,
## if running.

echo -n "Reload service $NAME "

$0 try-restart
rc_status
```

```
; ;  
    reload)  
## Zabbix agent daemon does not support configuration reloading,  
    thus reload fails  
  
rc_failed 3  
rc_status -v  
; ;  
    status)  
echo -n "Checking for service $NAME "  
## Check status with checkproc(8), if process is running  
## checkproc will return with exit status 0.  
  
# Return value is slightly different for the status command:  
# 0 - service up and running  
# 1 - service dead, but pid file exists  
# 2 - service dead, but /var/lock/ lock file exists  
# 3 - service not running (unused)  
# 4 - service status unknown :-(  
# 5--199 reserved (5--99 LSB, 100--149 distro, 150--199 appl.)  
  
# NOTE: checkproc returns LSB compliant status values.  
/sbin/checkproc -p $ZABBIX_PID $ZABBIX_BIN  
# NOTE: rc_status knows that we called this init script with  
# "status" option and adapts its messages accordingly.  
rc_status -v  
; ;  
    probe)  
## Optional: Probe for the necessity of a reload, print out the  
## argument to this init script which is required for a reload.  
## Note: probe is not (yet) part of LSB (as of 1.9)  
  
test $ZABBIX_CONFIG -nt $ZABBIX_PID && echo reload  
; ;  
*)  
echo "Usage: $0 {start|stop|status|try-restart|restart|force-  
    reload|reload|probe}"  
exit 1  
; ;  
esac  
rc_exit
```

To use these scripts, place them in `/etc/init.d` and create appropriate symlinks in `/usr/sbin` as root:

```
# ln -s /etc/init.d/zabbix_server /usr/sbin/rczabbix_server  
# ln -s /etc/init.d/zabbix_agentd /usr/sbin/rczabbix_agentd
```

Now let's try to start our newly created services. Again, as root execute:

```
# service zabbix_agentd start  
Starting zabbix agent daemon  
done  
# service zabbix_server start  
Starting zabbix server daemon  
done
```

That should get the agent and server started, although there should be no additional output. Note: you can also use other syntax — remember those symlinks we just created? You can also use them as:

```
# rczabbix_agentd start  
# rczabbix_server start
```

Feel free to experiment with other parameters, like `stop` and `restart` — it should be obvious what those two do.

You can verify whether services are running with the `status` parameter. For a service that is not running, you would get:

```
# service zabbix_server status  
Checking for service Zabbix server daemon  
unused
```

A running service would yield:

```
# service zabbix_agentd status  
Checking for service Zabbix agent daemon  
running
```

There's also another parameter called `probe`. This will check whether the corresponding service has been restarted since the last configuration file changes. If it has been restarted, no output will be produced. If the service has not been restarted (thus possibly missing some configuration changes), the string `reload` will be output.

Now that's nice, but we started all this with the goal of adding Zabbix services to system startup. Surely the previous actions didn't help with that? To reach our goal when using SysV-style init systems input the following as root:

```
# chkconfig -s zabbix_server 35
# chkconfig -s zabbix_agentd 35
```

This will add both services to be started at runlevels 3 and 5, which are used for multiuser and networked environments (though some distributions might use runlevel 4 instead of 5 for a graphical environment; consult your distribution's documentation when in doubt). There's usually no need to start Zabbix in single user or non-networked runlevels (1 and 2), as data gathering requires network connectivity.

To be fair, with the init scripts we created earlier it is even simpler than that:

```
# chkconfig -a zabbix_server zabbix_agentd
```

This will add both services as specified in the corresponding init scripts, which in our case is runlevels 3 and 5, configured by parameter Default-Start. If the command succeeds, you'll see the following output:

```
zabbix_server          0:off  1:off  2:off  3:on   4:off  5:on   6:off
zabbix_agentd         0:off  1:off  2:off  3:on   4:off  5:on   6:off
```

Slackware

The Slackware Linux distribution (<http://www.slackware.com/>) uses a simpler startup script system. Here's an example of a script to start and stop the Zabbix server and agent daemon on Slackware systems. This script is dependant on bash, but it shouldn't be too hard to rewrite for sh compatibility, if so desired. Create script /etc/rc.d/rc.zabbix with the following content:

```
#!/bin/bash

# Init script to start/stop Zabbix server and agent daemon

BINLOCATION=/usr/local/sbin
AGENTPID=/var/tmp/zabbix_agentd.pid
SERVERPID=/var/tmp/zabbix_server.pid
TERMINATEWAIT=5

processcheck() {
    [[ "$(ps -C $1 -o pid=)" ]] || return 1
}

[ 27 ]
```

```
# Check for stray pidfiles and remove them
processcheck zabbix_agentd || {
    [ -f $AGENTPID ] && ( echo "Removing stray $AGENTPID file";
        rm $AGENTPID )
}
processcheck zabbix_server || {
    [ -f $SERVERPID ] && ( echo "Removing stray $SERVERPID file";
        rm $SERVERPID )
}

killprocess() {
    processcheck $2 || {
        echo "No process $2 running"
        return
    }
    if [ -f $1 ]; then
        kill -15 $(cat $1)
    else
        echo "pidfile $1 not found"
    processcheck $2 && killall -15 $2
    fi
    echo -n "Waiting for $2 to terminate"
    for i in $(seq 1 $TERMINATEWAIT); do
        processcheck $2 || break
        sleep 1
        echo -n "."
    done
    processcheck $2 && echo "Warning! $2 did not terminate in
        $TERMINATEWAIT seconds"
}

zagent_start() {
    if [ -x $BINLOCATION/zabbix_agentd ]; then
        if processcheck zabbix_agentd; then
            echo "Zabbix agent daemon already running"
        else
            echo "Starting zabbix agent daemon: $BINLOCATION/zabbix_agentd"
            $BINLOCATION/zabbix_agentd
        fi
    else
        echo "Executable $BINLOCATION/zabbix_agentd not present"
        fi
}
```

```
zserver_start() {
    if [ -x $BINLOCATION/zabbix_server ]; then
        if processcheck zabbix_server; then
            echo "Zabbix server already running"
        else
            echo "Starting zabbix server: $BINLOCATION/zabbix_server"
            $BINLOCATION/zabbix_server
        fi
    else
        echo "Executable $BINLOCATION/zabbix_server not present"
        fi
}

zagent_stop() {
    echo "Stopping Zabbix agent daemon"
    killprocess $AGENTPID zabbix_agentd
}

zserver_stop() {
    echo "Stopping zabbix server"
    killprocess $SERVERPID zabbix_server
}

zagent_restart() {
    zagent_stop
    zagent_start
}

zserver_restart() {
    zserver_stop
    zserver_start
}

case "$1" in
    'start')
    case "$2" in
        'agent')
        zagent_start
        ;;
        'server')
        zserver_start
        ;;
        *)
        zagent_start
    esac
esac
```

```
zserver_start
;;
esac
;;
'stop')
case "$2" in
    'agent')
    zagent_stop
    ;;
    'server')
    zserver_stop
    ;;
    *)
    zagent_stop
    zserver_stop
    ;;
esac
;;
'restart')
case "$2" in
    'agent')
    zagent_restart
    ;;
    'server')
    zserver_restart
    ;;
    *)
    zagent_restart
    zserver_restart
    ;;
esac
;;
*)
echo "Usage: $0 start|stop|restart [agent|server]"
;;
esac
```

This script combines agent and server controlling. To start both, execute the following as root:

```
# /etc/rc.d/rc.zabbix start
```

To control the server or agent only, pass its name as a second argument.
For example:

```
# /etc/rc.d/rc.zabbix restart server
```

Adding a script to the system's startup sequence is quite trivial. Its invocation has to be added to the `/etc/rc.d/rc.local` as follows:

```
# echo /etc/rc.d/rc.zabbix start >> /etc/rc.d/rc.local
```

Verifying the service's state

While the init script method is a nice way to check a service's state for some distributions, it's not available everywhere and isn't always enough. Sometimes you might want to check whether the Zabbix server or agent is running with other methods.

- Checking running processes:

The most common method to check whether a particular process is running is by looking at running processes. You can verify if Zabbix agent daemon processes are actually running by using the following:

```
$ ps -C zabbix_agentd
```

- netstat output:

Sometimes, an agent daemon might start up, but fail to bind to the port, or the port might be used by some other process. You can verify whether some other process is listening on the Zabbix port, or whether Zabbix agent daemon is listening on the correct port by issuing:

```
$ netstat -npl
```

Note that process names won't be printed for other users' processes unless you are root. In the output, look for a line looking similar to:

Proto	Recv-Q	Send-Q	Local Address	Foreign Address
State			PID/Program name	
tcp	0	0	0.0.0.0:10050	0.0.0.0:*
LISTEN			19843/zabbix_agentd	

This indicates that the process `zabbix_agentd` is running and listening on all addresses, on port 10050, just what we need.

- Telnetting to the port

Even when a service starts up and successfully binds to a port, there might be some connectivity issues; perhaps due to a local firewall. To quickly check connectivity to the desired port, you can try:

```
$ telnet localhost 10050
```

This command should open a connection to the Zabbix agent daemon, and the daemon should not close the connection immediately.

The Web frontend

Now that we have the Zabbix server and agent compiled, installed, and running, there's probably a feeling that something's missing. We have only configured some low-level behavior, so where's the meat?

That's what the frontend is for. While in theory Zabbix can have multiple frontends, the only one with full functionality so far is web frontend, which was written in PHP. We have to set it up to configure Zabbix and get to those nice graphs everybody likes.

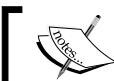
Prerequisites and setting up the environment

Of course, being a web frontend, it will require a platform to run on; a web server with a PHP environment. We will need the following installed:

- A web server that is supported by PHP; Apache is the most common choice
- PHP version 5

It is easiest to install these from distribution packages. For PHP we'll also need the following functionality:

- GD
- MySQL
- BCmath
- mbstring



Some distributions split out core PHP modules. These might include `php5-ctype` and `php-net-socket`.



Once you have all these installed, it's time to set up the frontend. First, you have to decide where the frontend code has to go. Most distributions that package web servers use `/srv/www/htdocs` or `/var/www`. If you compiled the Apache web server from source, it would be `/usr/local/apache2/htdocs` (unless you manually changed the

prefix or installed an older Apache version). We will place the frontend in a simple subdirectory, `zabbix`.

Assuming you have distribution packages installed with the web root directory at `/srv/www/htdocs`, placing the frontend where it is needed is as simple as executing it as root:

```
# cp -r frontends/php /srv/www/htdocs  
# mv /srv/www/htdocs/php /srv/www/htdocs/zabbix
```

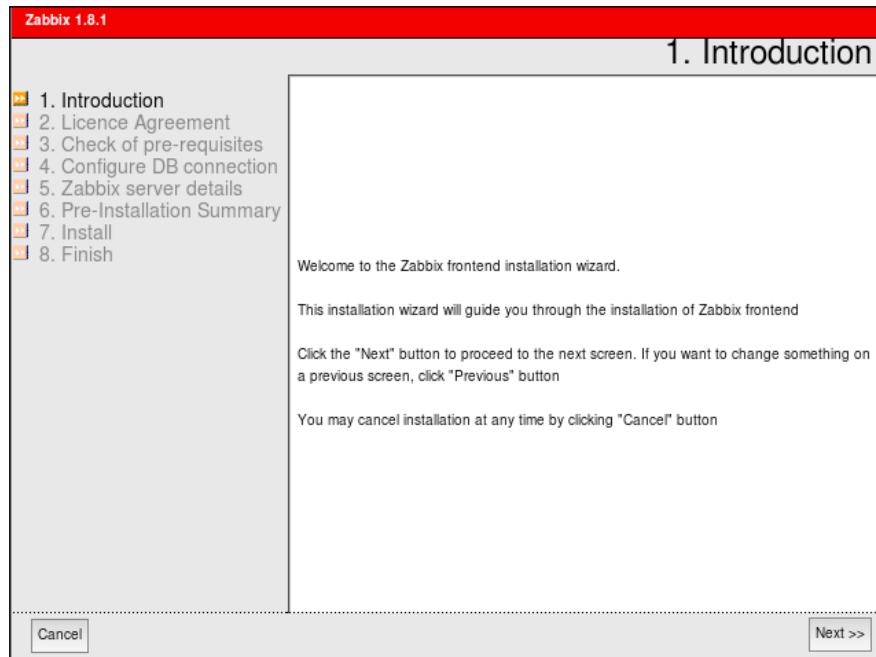
Now it's time to fire up a browser and navigate to Zabbix's address:
`http://<server_ip_or_name>/zabbix`. It should work just fine in the latest versions of most browsers, including Firefox, Opera, Konqueror, and Internet Explorer.

Installation of the web frontend

The web frontend has a wizard that helps you to configure its basics. Let's go through the simple steps it offers.

Step 1 – Welcome

If everything is configured properly, you should be greeted by the installation wizard:

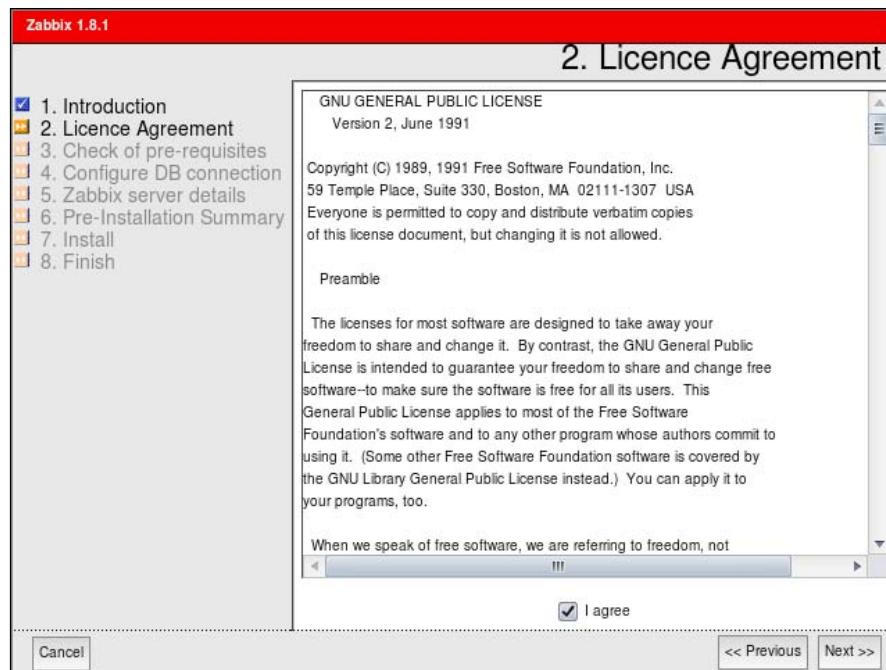


If you are not, there are several things that could have gone wrong.

If you see a blank page or some PHP code, make sure that PHP is properly installed and configured to parse files ending with the extension `.php` through the `AddType application/x-httpd-php` directive. If you see a file and directory listing instead of the installation wizard, make sure you have added `index.php` to `DirectoryIndex` directive. If these hints do not help, check the PHP documentation available at <http://www.php.net/manual/en/>.

This screen doesn't offer us much to configure, so just click **Next > >**.

Step 2 – Licence



On the second screen you are presented with the license, GPLv2. Hopefully you have no objections to this license, so check the **I agree** box and click **Next > >** again.

Step 3 – PHP prerequisites

In this step, the installation wizard checks PHP-related prerequisites. If you are lucky, all will be satisfied, and you will be greeted with all green entries.

	Current value	Required	Recommended	Status
PHP version	5.3.1	5.0	5.3.0	Ok
PHP memory limit	128M	128M	256M	Ok
PHP post max size	16M	16M	32M	Ok
PHP max execution time	300	300	600	Ok
PHP timezone	Europe/Riga			Ok
PHP databases support	MySQL			Ok
PHP BC math	yes			Ok
PHP MB string	yes			Ok
PHP MB string overload	yes			Ok
PHP Sockets	yes			Ok
PHP GD	2.0.34	2.0	2.0.34	Ok
GD PNG Support	yes			Ok
libxml module	2.7.6	2.6.15	2.7.6	Ok
ctype module	yes			Ok

Ok

If so, the **Next > >** button will be enabled, and you just have to click on it to continue to step 4.

Though, more often than not, one or more entries will have a red **Fail** warning listed next to them. This is where things get more interesting. Problems at this point fall into two categories – PHP installation, and configuration.

Entries like **PHP version**, **PHP Databases support**, **PHP BC math** **PHP MB string**, **GD Version**, and **Image formats** are PHP installation problems. To solve these, either install the appropriate distribution packages (sometimes called `php5-bcmath`, `php5-gd`, `php5-mysql`, and so on), or recompile PHP with the corresponding options.

PHP Memory limit, **PHP post max size**, **PHP max execution time**, **PHP Timezone**, and **PHP MB string overload** are configuration issues that are all set in the `php.ini` configuration file. This file is usually located at `/etc/php5` or similar for distribution packages, and `/usr/local/lib` for PHP source installations. Set the following options:

```
memory_limit = 128M
post_max_size = 16M
max_execution_time = 300
mbstring.func_overload = 2
```

For timezone, set the `date.timezone` option to a timezone that best matches your environment. For Zabbix home it's Europe/Riga, and you can see valid options at <http://www.php.net/manual/en/timezones.php>

[ Since Zabbix version 1.8.2 `mbstring.func_overload` is not required anymore.]

If you can't find `php.ini`, or you make changes but installation wizard does not pick them up, create a file named `test.php` in `htdocs` directory with only this content:

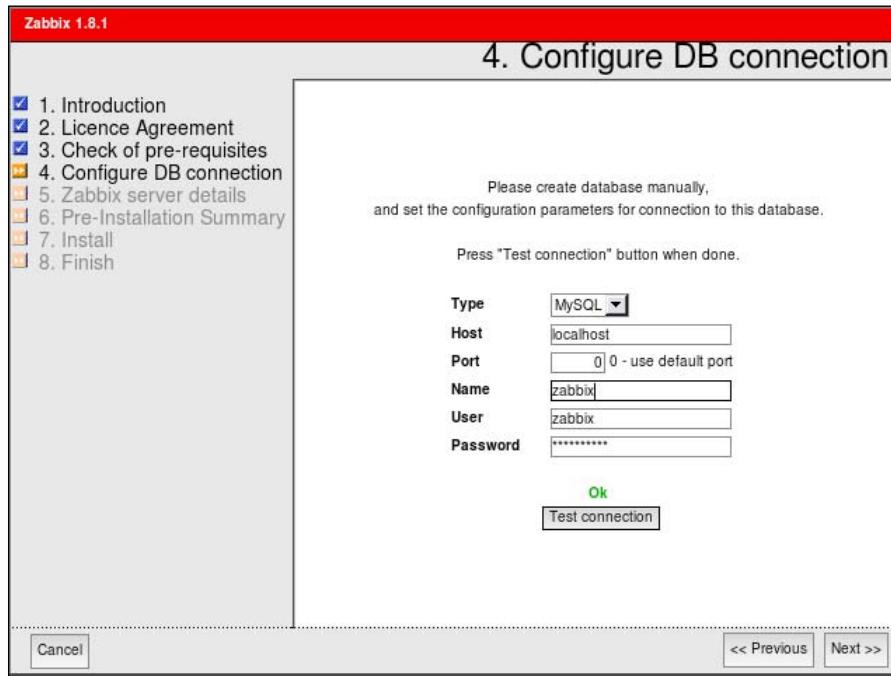
```
<?php phpinfo() ?>
```

Navigate using your browser to this file and check the value for entry **Configuration File (php.ini) Path**—that's where you should look for `php.ini`.

Once everything is fixed, click **Retry** button, and you should see more green entries.

Now the **Next >>** button should be enabled, click on it to continue.

Step 4 – Database access.



Zabbix 1.8.1

4. Configure DB connection

Please create database manually, and set the configuration parameters for connection to this database.

Press "Test connection" button when done.

Type	MySQL
Host	localhost
Port	0 - use default port
Name	zabbix
User	zabbix
Password	*****

Ok
Test connection
Cancel
<< Previous
Next >>

Remember the database we created before? That's the information we'll insert here. Default database **Type**, **Host**, and **Port** should work for us. Set both **Name** and **User** to zabbix. If you have forgotten the password, just look it up or copy it from `/etc/zabbix/zabbix_server.conf`. After entering the data, click the **Test connection** button. If all the information is correct, the status should be **Ok**, so click **Next >** again.

Step 5 – Zabbix server details

Zabbix 1.8.1

5. Zabbix server details

- 1. Introduction
- 2. Licence Agreement
- 3. Check of pre-requisites
- 4. Configure DB connection
- 5. Zabbix server details
- 6. Pre-Installation Summary
- 7. Install
- 8. Finish

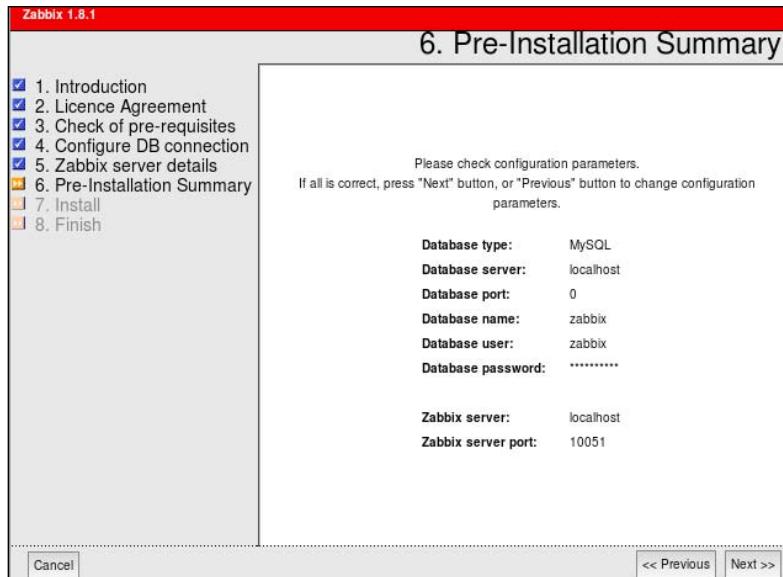
Please enter host name or host IP address
and port number of Zabbix server

Host:

Port:

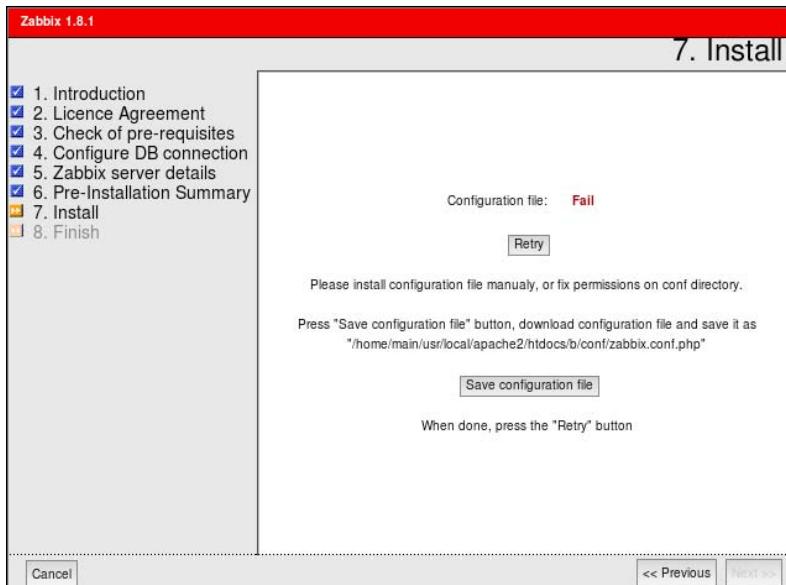
The next screen specifies the Zabbix server location, with defaults being suitable for us, so **Next** it is again. The following screen is a summary of choices made in the previous screen.

Step 6 – Summary



If you left the defaults where appropriate and your database connection test was successful, it should be safe to continue by clicking **Next >>**.

Step 7 – Writing the configuration file



It is quite likely that in the next screen you will be greeted with failure. The installation wizard attempted to save the configuration file, but with the access rights that it has, that should not be possible. There are two possible solutions:

- Click **Save configuration file** and place this file in the `htdocs/zabbix/conf` directory.
- Make a directory `htdocs/zabbix/conf` writable by web server user (execute as root):

```
# chown <username> /path/to/htdocs/zabbix/conf
# chmod 700 /path/to/htdocs/zabbix/conf
```

Obviously, we need to insert the correct username and directory in the above commands. Remember, common locations were `/srv/www/htdocs` and `/usr/local/apache2/htdocs`, use the one you copied the Zabbix frontend code to. Common users are `wwwrun`, `www-data`, `nobody`, and `daemon`— you can find out what the correct user for your system is by running:

```
$ ps aux | grep http
```

The username that most `httpd` processes are running under will be the correct one. Once the permissions have been changed, click **Retry**. That should successfully save the configuration file.



You can also skip configuration wizard by copying `zabbix.conf.php.example` in the `conf` directory to `zabbix.conf.php` and editing it directly.

It is suggested that you restrict the permissions on this file afterwards to be readable only by web server user, by issuing as root:

```
# chmod 400 /path/to/htdocs/zabbix/conf/zabbix.conf.php
# chown <username> /path/to/htdocs/zabbix/conf/zabbix.conf.php
```

The file contains the database password which is better kept secret.



It is possible to personalize Zabbix frontend installation a bit. Adding in the `zabbix.conf.php` configuration file after the line that starts with `$ZBX_SERVER_PORT` another line `$ZBX_SERVER_NAME = "Test server";` will make all pages in Zabbix setup use this name in page titles and display it in the upper-right corner of the frontend, so it would be easier to distinguish this installation from the production instance created later.

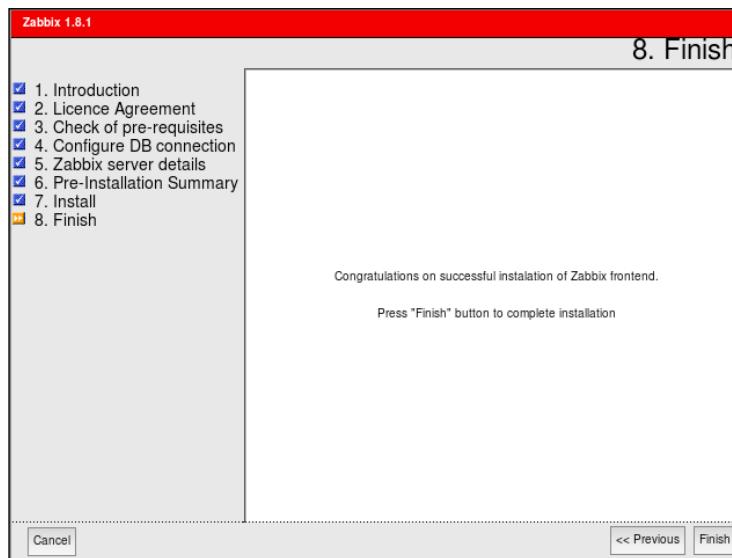
Step 8 – Configuration file in place

After successfully performing this step and clicking **Retry**, the installation wizard should be happy, so do as it says and click **Next**.



Step 9 – Finishing the wizard

Congratulations, this is the last wizard screen which only wants you to be friendly to it and press **Finish**.



Step 10 – Logging in

Immediately after clicking **Finish** you should see a login form.



The screenshot shows the Zabbix 1.8.1 login interface. At the top, there's a red header bar with the word "ZABBIX" in white. To its right are links for "Help|Get support|Print|Logout". Below the header is a navigation menu with tabs for "Monitoring", "Inventory", and "Reports". A search bar labeled "SEARCH:" is positioned above the login form. The main area contains a "Login" dialog box with fields for "Login name" and "Password", and a blue "Enter" button. At the bottom of the page, a footer bar displays the text "Zabbix 1.8.1 Copyright 2001-2009 by SIA Zabbix | Connected as 'guest'".

The Zabbix database data that we inserted previously also supplied the default username and password, so enter **admin** as the **Login name**, and **zabbix** as the **Password**.

That should get you to the initial frontend screen, which doesn't show much; it only greets you.



Congratulations! The web frontend is now set up and we have logged in.

Note that with some browsers it is possible that you will see either the same page or a blank screen after pressing **Finish** in the installation wizard, or after logging in. If that is the case, you can try either refreshing the page, or opening the Zabbix root URL, http://<server_ip_or_name>/zabbix.

Getting Started with Zabbix

This view isn't too exciting, so let's check whether the frontend can see if the Zabbix server is running. Hover your mouse over **Reports** and click **Status of Zabbix**, the very first report.

The screenshot shows the Zabbix web interface with the title 'ZABBIX' at the top. The navigation bar includes 'Monitoring', 'Inventory', 'Reports' (which is highlighted in blue), 'Configuration', and 'Administration'. On the right of the navigation bar are links for 'Help', 'Get support', 'Print', 'Profile', and 'Logout'. Below the navigation bar is a search bar with the placeholder 'SEARCH:'. A link 'Status of Zabbix' is visible above the main content area. The main content is titled 'REPORT' and contains a table with the following data:

Parameter	Value	Details
Zabbix server is running	Yes	-
Number of hosts (monitored/not monitored/templates)	43	0 / 1 / 42
Number of items (monitored/disabled/not supported)	0	0 / 0 / 0
Number of triggers (enabled/disabled)[true/unknown/false]	0	0 / 0 [0 / 0 / 0]
Number of users (online)	2	2
Required server performance, new values per second	0	-

At the bottom of the report, it says 'Updated: 14:33:54'. The footer of the page includes the text 'Zabbix 1.8.1 Copyright 2001-2009 by SIA Zabbix' and 'Connected as 'Admin''. The entire screenshot is framed by a thick black border.

So that's some more information already. The frontend successfully sees that Zabbix server is running. It also knows some basic things about Zabbix's configuration – there are in total **43** hosts configured. Wait, what's that? We have only set it up and did not configure anything, how can there be **43** hosts already? Let's take a closer look at the **Details** column. Those values correspond to descriptions in parenthesis, located in **Parameter** column. So there are **0** monitored hosts, **1** that is not monitored and **42** templates. Now that makes more sense – 42 of those 43 are templates, not actual hosts. Still, there's one host that isn't monitored, what's up with that?

Hover your mouse over **Configuration** and choose **Hosts**.

Name	Applications	Items	Triggers	Graphs	DNS	IP	Port	Templates	Status	Availability
Zabbix Server	Applications (12)	Items (102)	Triggers (44)	Graphs (4)	-	127.0.0.1	10050	Template_Linux	Not monitored	

So there it is. It turns out that the default Zabbix database already has one server configured; the local Zabbix server. It is disabled by default, as indicated in the **Status of Zabbix** screen and here by **Not monitored** string in **Status** column.

Summary

In this chapter we set up a fresh Zabbix installation consisting of a database, a server, and an agent daemon; all running on the same machine. We also installed and configured the Zabbix web frontend, based on PHP, to access the database.

Hopefully, by the end of this book you will be able to exploit full power of all of these methods, whether you go to sleep thinking about SNMP MIB syntax and wake with new ideas for scripts, or if you have only heard of TCP and get scared by SNMP.

2

Getting Your First Notification

We have now installed Zabbix, but it's not doing much – at least that's what we'd expect. Software that starts doing something on its own probably would be a bit undesirable, at least for now. The promise of Zabbix was to inform you about problems as soon as possible, preferably before your users and management notice them. But how do we get data, where do we place it, and how do we define what a problem is? We will try to quickly get Zabbix working and alerting on a single monitored item, which is the most common scenario. Before we can tell Zabbix who to send notifications to, we will have to explore and use some basic Zabbix concepts. They are:

- Navigating around the frontend
- Creating a host and item (Zabbix term for a monitored metric) for it
- Looking at the gathered data and finding out how to get it graphed
- Defining problem threshold with a trigger
- Telling Zabbix that it should send e-mail when this threshold is exceeded
- Causing a problem to actually receive a notification

Exploring the frontend

While we have already looked at some data provided by the frontend, we should get a bit more familiar with it before attempting some more configuration tasks.

Configuration steps will be followed by verifying results in **Monitoring** section. We will then explain some generic item terms used in Zabbix, and their use. Items, being the basis of information gathering, have a fair amount of configuration possibilities.

Getting Your First Notification

In your browser, open Zabbix's root URL; `http://<server_ip_or_name>/zabbix` and log in again, if you have been logged out. Hover your mouse cursor over the **Monitoring** section and click on the **Dashboard** entry. You should now see a pretty empty dashboard with little information:

The screenshot shows the Zabbix 1.8.1 dashboard. At the top, there is a navigation bar with links for Monitoring, Inventory, Reports, Configuration, Administration, Dashboard, Overview, Web, Latest data, Triggers, Events, Graphs, Screens, Maps, Discovery, IT services, and a search bar. Below the navigation bar, the history shows "Status of Zabbix » Hosts » Dashboard".

The main area is titled "PERSONAL DASHBOARD". On the left, there is a sidebar with sections for Favourite Graphs, Favourite Screens, Favourite Maps, and Favourite Maps. The main content area contains several cards:

- Status of Zabbix**: A table showing system status with rows for Zabbix server is running (Yes), Number of hosts (43), Number of items (0), Number of triggers (0), and Number of users (2). Last updated: 09:34:41.
- System status**: A table showing host group distribution: Disaster (0), High (0), Average (0), Warning (0), Information (0), and Not classified (0). Last updated: 09:34:41.
- Host status**: A table showing host status distribution: Without problems (0), With problems (0), and Total (0). Last updated: 09:34:41.
- Last 20 issues**: A table showing the last 20 issues with columns for Host, Issue, Last change, Age, Ack, and Actions. Last updated: 09:34:41.
- Web monitoring**: A table showing web monitoring status distribution: Ok (0), Failed (0), In progress (0), and Unknown (0). Last updated: 09:34:41.

At the bottom, it says "Zabbix 1.8.1 Copyright 2001-2009 by SIA Zabbix" and "Connected as 'Admin'".

Move your mouse cursor over the entries of the top menu bar and observe how the lower menu bar opens to show sub-entries of the chosen category. Hovering mouse cursor over your chosen option, then clicking to select it will give you access to your chosen piece of Zabbix's frontend. You will be using the menus a lot, so in the future we'll refer to the action we just performed as something similar to **Monitoring | Dashboard**. (Whenever you see such a notation, first is the main category, second is entry under it.)

As you can see in the following screenshot, there are five main categories:



- **Monitoring:** This category contains most of the monitoring-related pages. You will be able to view data, problems, and graphs here.
- **Inventory:** Here inventory data for monitored systems can be viewed, if properly filled.
- **Reports:** Whenever you'll need some nice report, visualizing more than few items, this will be the category to look at most likely.
- **Configuration:** Setting up everything related to monitoring of systems, parameters, notification sending, and so on happens here.
- **Administration:** This section allows to set up more of the Zabbix internals, including authentication methods, users, permissions, and global Zabbix configuration.

Before we venture deeper into these categories, it might be worth visiting the **Profile** section—see the link in the upper-right corner.

User "Admin"	
Password	<input type="button" value="Change password"/>
Language	English (GB) <input type="button" value="▼"/>
Theme	System default <input type="button" value="▼"/>
Auto-login	<input type="checkbox"/>
Auto-logout (min 90 seconds)	<input type="checkbox"/> 900
Refresh (in seconds)	30
Rows per page	50
URL (after login)	
Media	No media defined <input type="button" value="Add"/>
<input type="button" value="Save"/> <input type="button" value="Cancel"/>	

Here you can set some options concerning your user account for example, changing the password, the frontend language, or the frontend theme. As we will use an English frontend, it is suggested to leave the defaults for these three as it is. Notice that you can find out user account that you currently are connected as in the lower-right corner of the frontend. When you are not logged in, **guest** is displayed in that area.

There are two options related to logging in. **Auto-login**, which will automatically log the user in, using a cookie saved by their browser, and **Auto-logout**. It is suggested that you disable the latter for our test installation, as shown above. This way, you won't be logged out if your workflow is interrupted for a few minutes.

We won't change the **URL** option at present, but we'll discuss the benefits of setting a custom default URL for a particular user later. The **Refresh** sets the period in seconds that some pages in the frontend will refresh automatically to show new data. It might be beneficial to increase this parameter for huge screens that we do not have yet.

After you have disabled **Auto-logout**, click **Save**.

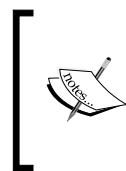
Take a quick look at below the menu — there's an option called **History**, commonly called **breadcrumbs** in computer software.

History: Status of Zabbix » Hosts » Dashboard » User profile

Here you can see last five pages that you have visited in the Zabbix frontend. Each of them can be clicked to quickly return to that page. After you click on another category, the page you just left is added to the right-hand side of the history.

Monitoring quickstart

Now that we have basic understanding of the frontend navigation, it's time to look at the basis for data gathering in Zabbix — items. In general, anything you want to gather data about will eventually go into an **item**.



An item in Zabbix is a configuration entity that holds information on gathered metrics. It is the very basis of information flowing into Zabbix, and without items nothing can be retrieved. An item does not hold any information on thresholds — that functionality is covered by **triggers**.

If items are so important in Zabbix, we should create some. After all, if no data retrieval is possible without items, we can't monitor anything without them. To get started with item configuration, open **Configuration | Hosts**. If it's not selected by default, choose **Zabbix Server** in the **Group** dropdown (in the top-right corner). This is a location we will visit quite a lot, as it provides easy access to other entity configurations, including **Items** and **Triggers**. Let's figure out what's what in this area. The most interesting functionality is the host list.

	Name	Applications	Items	Triggers	Graphs
	Zabbix Server	Applications (12)	Items (102)	Triggers (44)	Graphs (4)

Primarily, it provides access to host details in the very first column, but that's not all. The usefulness of this screen comes from the other columns, which not only provide access to elements that are associated with hosts, but also lists the count of those elements. Further down the host entry we can see a quick overview of the most important host configuration parameters, as well as status information that we will explore in more detail later.

DNS	IP	Port	Templates	Status	Availability
-	127.0.0.1	10050	Template_Linux	Not monitored	

We came here looking for items, so click on **Items** next to **Zabbix Server**. You should see a list similar to the one in the following screenshot:

Log	Description	Triggers	Key	Interval	History	Trends	Type	Status	Applications	Error
	Template_Linux:Buffers memory	Triggers (0)	vm.memory.size[buffers]	30	7	365	Zabbix agent	Active	Availability, Memory	
	Template_Linux:Cached memory	Triggers (0)	vm.memory.size[cached]	30	7	365	Zabbix agent	Active	Availability, Memory	
	Template_Linux:Checksum of /usr/sbin/sshd	Triggers (1)	vts.file_cksum[/usr/sbin/sshd]	600	7	365	Zabbix agent	Active	Integrity	
	Template_Linux:Checksum of /usr/bin/ssh	Triggers (1)	vts.file_cksum[/usr/bin/ssh]	600	7	365	Zabbix agent	Active	Integrity	
	Template_Linux:Checksum of /vmlinuz	Triggers (1)	vts.file_cksum[/vmlinuz]	600	7	365	Zabbix agent	Active	Integrity	
	Template_Linux:Checksum of /etc/services	Triggers (1)	vts.file_cksum[/etc/services]	600	7	365	Zabbix agent	Active	Integrity	
	Template_Linux:Checksum of /etc/inetd.conf	Triggers (1)	vts.file_cksum[/etc/inetd.conf]	600	7	365	Zabbix agent	Active	Integrity	
	Template_Linux:Checksum of /etc/passwd	Triggers (1)	vts.file_cksum[/etc/passwd]	600	7	365	Zabbix agent	Active	Integrity	
	Template_Linux:CPU iowait time (avg1)	Triggers (0)	system.cpu.util[iowait,avg1]	10	90	365	Zabbix agent	Active	CPU, Performance	

Note the method we used to reach the items list for a particular host—we were using convenience links for host elements, which is the easiest way to get there and the reason why we will use **Configuration | Hosts** often.

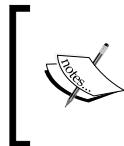
Back to what we were after, we can see a fairly long list of already existing items. But wait, didn't the Zabbix status screen that we saw earlier claim there's a single host and no items? That's clearly wrong! Return to **Reports | Status of Zabbix** (or **Monitoring | Dashboard**, which shows the same data). It indeed shows zero items. Now move the mouse cursor over the text that reads **Number of items**, see the tooltip.

Parameter
Zabbix server is running
Number of hosts (monitored/not monitored/templates)
Number of items (monitored/disabled/not supported)
Number of triggers (enabled/disabled)/true/false/unknown/false
N Only items assigned to enabled hosts are counted
Required server performance, new values per second

Ah-ha, so it counts only those items that are assigned to enabled hosts. As this example host, **Zabbix server**, is disabled, it's now clear why the Zabbix status report shows zero items. This is handy to remember later; once you try to evaluate a more complex configuration.

Creating a host

Instead of using this predefined host configuration we want to understand how items work. But items can't exist in an empty space—each item has to be attached to a **host**.



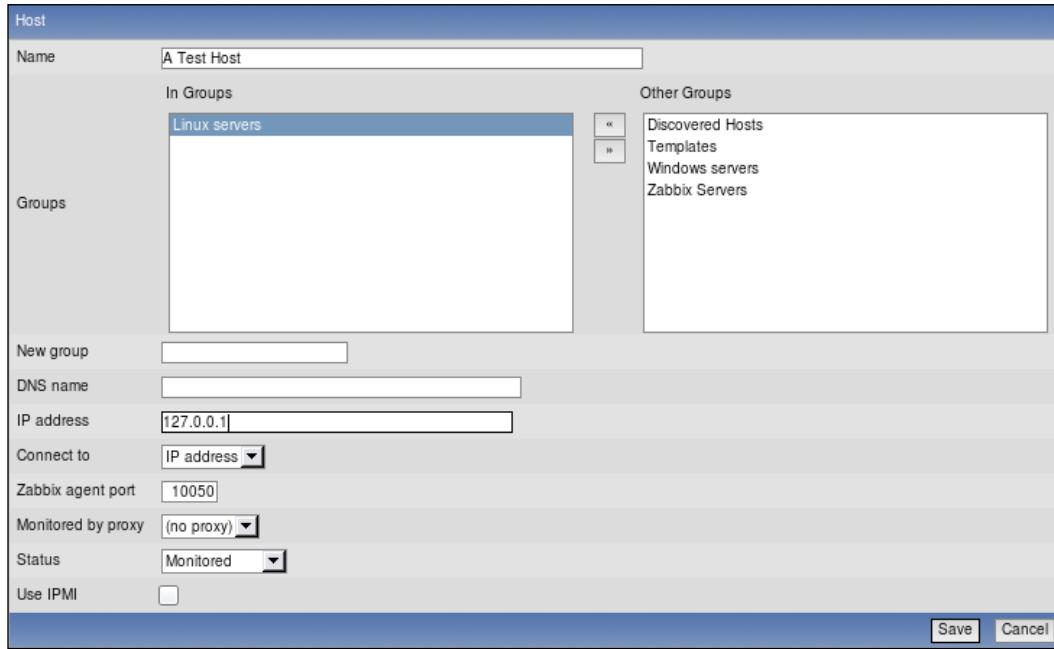
In Zabbix, a host is a logical entity that groups items. Definition of what a host is can be freely adapted to specific environment and situation. Zabbix in no way limits this choice; thus a host can be network switch, a physical server, a virtual machine, or a website.

If a host is required to attach items to then we must create one. Head over to **Configuration | Hosts** and click the **Create Host** button, located at the top-right corner. We are presented with a host creation screen. This time we won't concern ourselves with the details, so let's input only some relevant information.

- **Name:** Enter **A Test Host**
- **Groups:** Select **Linux servers** in the right-hand selectbox, named **Other Groups**, press the << button to add this group then select **Zabbix Servers** in the **In Groups** selectbox and press the >> button to remove our new host from this pre-defined group
- **IP address:** Enter **127.0.0.1**

 Why did we have to select a group for this host? All permissions are assigned to host groups, not individual hosts, and thus a host must belong to at least one group. We will cover permissions in more detail in *Chapter 5*.

The configured host should look as follows:



DNS	IP
-	127.0.0.1

When you are ready, click **Save**.

DNS	IP
-	127.0.0.1

Back in the host list, take a look at **DNS** and **IP** columns. Notice how IP value is listed in bold – that's because it was selected in the **Connect to** field.

Creating an item

So we created our very own first host. But given that items are the basis of all the data, it's probably of little use right now. To give more substance we should create items, so select **Linux servers** in the **Group** dropdown, then click **Items** next to the host we just created, **A Test Host**. This host has no items to list—click **Create Item** button.

There's a form, vaguely resembling the one for host creation, so let's fill some values.

- **Description:** Enter value as **CPU Load**. This is how the item will be named—basically the name that you will use to refer to the item in most places.
- **Key:** The value in this field will be `system.cpu.load`. This is the "technical name" of the item that identifies what information it gathers.
- **Type of information:** Choose **Numeric (float)**. This defines what formatting and type the incoming data will have.

The screenshot shows the configuration dialog for an item named 'A Test Host:CPU Load'. The fields are as follows:

- Host:** A Test Host
- Description:** CPU Load
- Type:** Zabbix agent
- Key:** system.cpu.load
- Type of information:** Numeric (float)
- Units:** (empty)
- Use multiplier:** Do not use
- Update interval (in sec):** 30
- Flexible intervals (sec):** No flexible intervals
- New flexible interval:** Delay 50 Period 1-7:00:00-23:59 (with an 'Add' button)
- Keep history (in days):** 90
- Keep trends (in days):** 365
- Status:** Active
- Store value:** As is
- New application:** (empty)
- Applications:** -None-
- Group:** Discovered Hosts
- Buttons:** Save, Cancel, Add to group, do

We will look at the other defaults in more detail later, so click **Save**.



More information on item keys is provided in *Chapter 3*.

You should now see your new item in the list. But we were interested in the associated data, so navigate to **Monitoring | Latest data** and click on the + sign next to **- other -**. Wait for a minute to pass since saving the item, and you should see that this newly created item has already gathered some data:

Host	+ Description	Last check	Last value	Change	History
A Test Host	- other - (1 Items)				
	CPU Load	01 Feb 2010 10:46:22	0.210000	-	Graph

What should you do if you don't see any entries at all? This usually means that data has not been gathered, which can happen for a variety of reasons. If that is the case, check some common causes:

- Did you enter item configuration exactly as in the screenshot? Check the item key and type of information.
- Are both the agent and the server, running? You can check that by executing as root:

```
# netstat -npl | grep zabbix
```

The output should list both the server daemon and the agent daemon running on the correct ports:

```
tcp      0      0 0.0.0.0:10050          0.0.0.0:*
LISTEN    23569/zabbix_agentd
tcp      0      0 0.0.0.0:10051          0.0.0.0:*
LISTEN    23539/zabbix_server
```

If any one of them is missing, make sure to start it.

- Can the server connect to the agent? You can verify that by executing from Zabbix server:

```
$ telnet localhost 10050
```

If the connection fails, that could mean that either the agent is not running, or some restrictive firewall setting prevents the connection.

If the connection succeeds but is immediately closed, then the IP address that agent receives connection from does not match the one specified in its configuration file for the `Server` directive. On some distributions this can be caused by IPv6 being used by default, so you should try to add IPv6 localhost representation to this directive, `::1`.

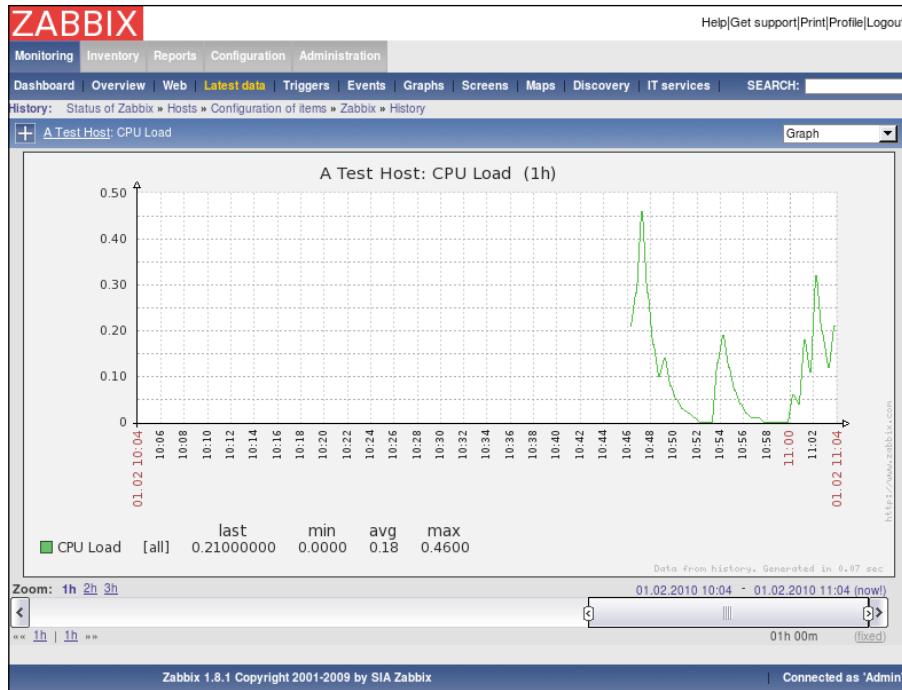
Since version 1.8, Zabbix server reads all the information on items to monitor into cache every minute by default. This means that configuration changes like adding a new item might show an effect in collected data after one minute. This interval can be tweaked in `/etc/zabbix/zabbix_server.conf`, `CacheUpdateFrequency` parameter.

Once data is arriving, you might see no value in the **Change** column. This means you moved to this display quickly, and the item managed to gather single value only, thus there's no change yet. If that is the case, waiting a bit should result in page automatically refreshing (look at the page title, remember the 30 second refresh we left untouched in user profile?) and the **Change** column will be populated. So we are now monitoring single value – the UNIX system load. Data is automatically retrieved and stored in the database. If you are not familiar with the concept, it might be a good idea to read the overview at [http://en.wikipedia.org/wiki/Load_\(computing\)](http://en.wikipedia.org/wiki/Load_(computing)).

Introducing simple graphs

If you went reading about system load, several minutes should have passed by now. Now is a good time to look at another feature in Zabbix – graphs. Graphs are freely available for any monitored item without any additional configuration.

You should still be in the **Latest data** screen with **CPU Load** item visible, so click on the link named **Graphs**.



While you probably will get less data, unless reading about system loads took more than hour, overall your screen should look very similar. Let's explore some basic graph controls.



If you don't see any data even after several minutes have passed, try dragging the scrollbar below the graph to the right-most position.

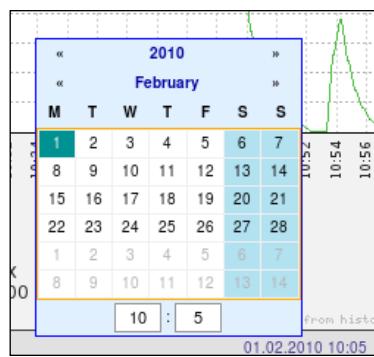
The **Zoom** controls in the lower-left corner allow you to quickly switch the displayed period. Clicking on any of the entries will make graph show data for the chosen duration. As more data is gathered, longer zoom periods will become available here.



Below these controls are time period moving options, clicking on them will move the displayed period by the exact time that was clicked.

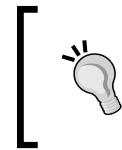
The **scrollbar** at the bottom allows you to make small changes to the displayed period – drag it to the left (and notice period at the bottom of the graph changing), and then release, and the page reloads and the period displayed is changed. Notice the arrows at both ends of the scrollbar, they allow you to change the duration displayed. Drag these with mouse just like scrollbar. You can also click the buttons at both ends for exact adjustments. Using these buttons moves the period back and forth by the time period that we currently have displayed.

The **Date** entries in the lower-right corner show the starting and ending times for the currently displayed data, and they also provide calendar widgets that allow a wider range of arbitrary period settings. Clicking one of these time periods will open a calendar, where you can enter time, then click on a date and have either the start or end time set to this choice.



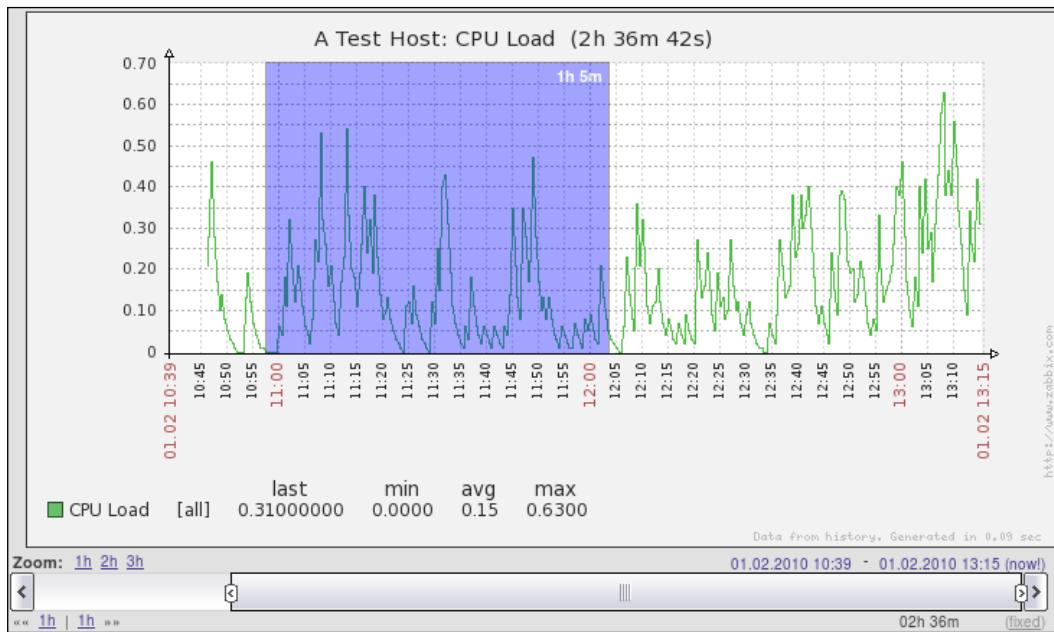
Try entering a time in the past for the starting (leftmost) calendar, then clicking on current date. That will move the displayed period, not changing its length. This is great if we are interested in a time period of a specific length, but what if we want to see a graph for yesterday, from 08:30 till 17:00? For that the control (**fixed**) in the lower-right corner will help. Click on it once – it changes to (**dynamic**). If you now use calendar widgets to enter start or end time for the displayed period, only this edge of the period will be changed.

For example, if a one-hour period from 10:00 to 11:00 is displayed, setting the first calendar to 09:00 while in (**fixed**) mode will display period from 09:00 till 10:00. If the same is done while in (**dynamic**) mode, two-hour period from 09:00 till 11:00 will be displayed. The end edge of the period is not moved in the second case.



Depending on the time when you are looking at the graphs, some areas of the graph might have gray background. This is the time outside of working hours, as defined in Zabbix. We will explore this in more detail later.

Clicking and dragging over the graph area will zoom to the selected period once the mouse button is released. This is handy for a quick drilldown to some problematic or interesting period.



The blue area denotes the time period we selected by clicking, holding down the mouse button, and dragging over the graph area. When we release the mouse button, the graph is zoomed to the selected period.



The graph period can't be shorter than one hour in Zabbix. Attempting to set it to a smaller value will do nothing.

Creating triggers

Now that we have an item successfully gathering data, we can look at it and verify whether it is reporting as expected (in our case, that system is not overloaded) or not. Sitting and staring at a single parameter would make for a very boring job. Doing that with thousands of parameters doesn't sound too entertaining, so we are going to create a trigger. In Zabbix, a trigger is an entry containing an expression to automatically recognize problems with monitored items.



Item alone does nothing more than collect data. To define thresholds and what is considered a problem we have to use triggers.



Navigate to **Configuration | Hosts**, click **Triggers** next to **A Test Host** and click on **Create Trigger**. Again, we are presented with a form to be filled.

The screenshot shows the 'Trigger' configuration dialog box. The 'Name' field contains 'CPU Load too high on Test Host for last 3 minutes'. The 'Expression' field contains '{A Test Host:system.cpu.load.avg(180)}>2'. The 'Event generation' dropdown is set to 'Normal'. The 'Severity' dropdown is set to 'Not classified'. The 'Comments' and 'URL' fields are empty. The 'Disabled' checkbox is unchecked. At the bottom right are 'Save' and 'Cancel' buttons.

Here, only two fields need to be filled in.

- **Name:** Enter **CPU Load too high on Test Host for last 3 minutes**
- **Expression:** Enter **{A Test Host :system.cpu.load.avg(180)}>2** (outer being curly braces)

It is important to get the expression correct down to the last symbol. Once done, click **Save**. Don't worry about understanding the exact trigger syntax yet, we will get to that later.

Notice how our trigger expressions refers to the **item key**, not the description. Whenever you will have to reference an item inside Zabbix, that will be done by the item key.

The trigger list should be now displayed, with single trigger – the one we just created. Let's take a look at what we just added; open **Monitoring | Triggers**. You should see freshly added trigger, hopefully already updated, with a green **OK** flashing in the **Status** column.

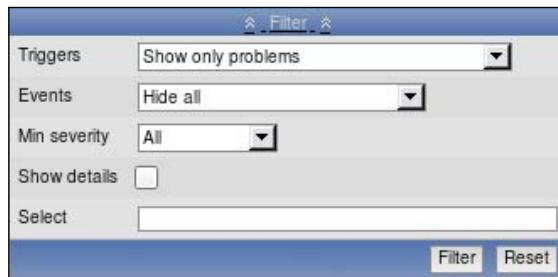
Severity	Status	Last change ▾	Age	Acknowledged	Host	Name	Comments
Not classified	OK	01 Feb 2010 13:20:22	30m 59s	Acknowledged	A Test Host	CPU Load too high on Test Host for last 3 minutes	Add



You might see **PROBLEM** in the **Status** field. This means exactly what the trigger name says – the CPU load has been too high for the last three minutes.



Notice the link above the trigger list saying **Filter** – click it.



Show all option might be unavailable if you have chosen to display data for all hosts in all host groups. To access this option, choose one host group or host.



Great, we can filter displayed triggers, but why is our **OK** trigger displayed even though the default filter says **Show only problems**? The thing is, Zabbix always shows triggers that have recently changed their state with the status indicator flashing. Such triggers show for 30 minutes, then they obey normal filtering rules. Click **Filter** again to close the filter. We will explore this filter in more detail later.

You could take a break now, and notice how in 30 minutes there are no triggers displayed. With the filter set to show only problems this screen becomes quite useful for a quick overview of all issues concerning monitored hosts. While that sounds much better than staring at plain data, we would still want to get some more to the point notifications delivered.

Configuring e-mail parameters

The most common notification method is e-mail. Whenever something interesting happens in Zabbix, some action can be taken, and we will set it up so that an e-mail is sent to us. Before we decide when and what should be sent, we have to tell Zabbix *how* to send it.

To configure the parameters for e-mail sending, open **Administration | Media types** and click on **Email** in the **Description** column. You'll get a simple form to fill in with appropriate values for your environment:

The screenshot shows a configuration dialog titled 'Media'. It has fields for 'Description' (set to 'Email'), 'Type' (set to 'Email'), 'SMTP server' (set to 'mail.company.com'), 'SMTP helo' (set to 'company.com'), and 'SMTP email' (set to 'zabbix@company.com'). At the bottom are 'Save', 'Delete', and 'Cancel' buttons.

Change the **SMTP server**, **SMTP helo**, and **SMTP email** fields to use a valid e-mail server. The **SMTP email** address will be used as the `From` address, so make sure it's set to something your server will accept.



SMTP authentication is currently not supported.



So we have configured the server to send e-mail messages, and set what the `From` address should be but it still doesn't know the e-mail addresses that our defined users have, which is required to send alerts to them. To assign an e-mail address for a user, open **Administration | Users**, then choose **Users** in the first dropdown. You should see only two users, **Admin** and **guest**. Click on **Admin** in the **Alias** column.

User "Admin"

Alias	Admin
Name	Zabbix
Surname	Administrator
Password	Change password
Zabbix administrators	
Groups	
Add Delete selected	
Language	English (GB)
Theme	System default
Auto-login	<input type="checkbox"/>
Auto-logout (min 90 seconds)	<input type="checkbox"/> 90
Refresh (in seconds)	30
Rows per page	50
URL (after login)	
Media	No media defined
Add	
User rights (Show)	
Save Delete Cancel	

We have to add a way to contact this user, which is done in the **Media** entry. Click the **Add** button below the **No media defined** text.

New media

Type	Email
Send to	
When active	1-7:00:00-23:59
<input checked="" type="checkbox"/> Not classified <input checked="" type="checkbox"/> Information <input checked="" type="checkbox"/> Warning <input checked="" type="checkbox"/> Average <input checked="" type="checkbox"/> High <input checked="" type="checkbox"/> Disaster	
Status	Enabled
Add Cancel	

The only thing you have to enter here is a valid e-mail address in the **Send to** textbox, preferably yours. Once you are done, click **Add** and then **Save** in user properties.

That finishes the very basic configuration needed to send out notifications through e-mail for this user.

Creating an action

And now it's time to tie all this together and tell Zabbix that we want to receive e-mail notification when our test box is under a heavy load.

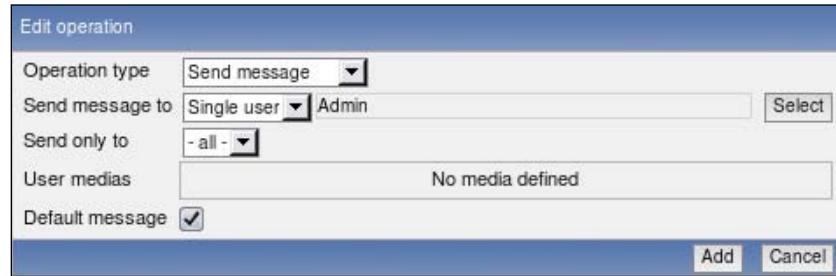
Things that tell the Zabbix server to do something upon certain conditions are called **actions**. An action has three main components:

- **Main configuration:** This allows us to set up general options, such as the e-mail subject, and the message.
- **Action operations:** Specify what exactly has to be done, including who to send the message to, and what message to send.
- **Action conditions:** Allow us to specify when this action is used and when operations are performed. Zabbix allows us to set many detailed conditions, including hosts, host groups, time, specific problems (triggers) and their severity, as well as others.

To configure actions, open **Configuration | Actions**. There are no existing actions listed, so click **Create Action**. A form is presented to configure preconditions and the action to take.

The screenshot shows the 'Action' configuration dialog box. On the left, there is a main configuration section with fields for Name (Test Action), Event source (Triggers), Enable escalations (unchecked), Default subject ([TRIGGER.NAME]: [STATUS]), Default message (a text area containing two lines: [TRIGGER.NAME]: [STATUS]), Recovery message (unchecked), and Status (Enabled). At the bottom of this section are Save and Cancel buttons. To the right of this is a sidebar titled 'Action operations' with tabs for Details and Action. The Details tab is selected, showing the message 'No operations defined'. Below this is a 'New' button. At the bottom of the sidebar is another 'New' button. At the very bottom of the dialog is a 'Conditions' section with the message 'No conditions defined' and a 'New' button.

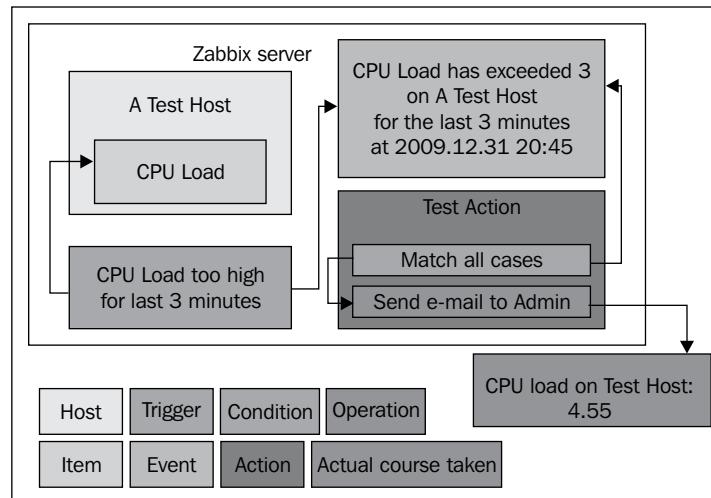
First, enter some name for our new action, like **Test Action**. Next, we should define the operation to perform, so click **New** in the **Action operations** block (that's the one to the right), which will open the operation details block.



Choose **Single user** in the **Send message to** drop-down and click the **Select** button. Click on **Admin** user and then on **Add** button. Congratulations! You have just configured the simplest possible action, so click the **Save** button in **Action** block.

Information flow in Zabbix

We have now configured various things in Zabbix frontend, including data gathering (item), threshold definition (trigger), and instructions on what to do if threshold is exceeded (action). But how does it all work together? The flow of information between Zabbix entities can be non-obvious at first glance. Let's look at a schematic, showing how pieces go together.



In our Zabbix server installation we created a **host** (A Test Host), which contains an **item** (CPU Load). A **trigger** references this item. Whenever the trigger expression matches current item value, the trigger switches to a PROBLEM state. When it ceases to match, it switches back to an OK state. Each time the trigger changes its state, an **event** is generated. The event contains details of the trigger state's change – when did it happen and what the new state is. When configuring an action, we can add various **conditions** so that only some events are acted upon. In our case, we did not add any, so all events will be matched. Each **action** also contains **operations**, which define what exactly has to be done. In the end some course is actually carried out, which usually happens outside of the Zabbix server itself, like sending an e-mail.



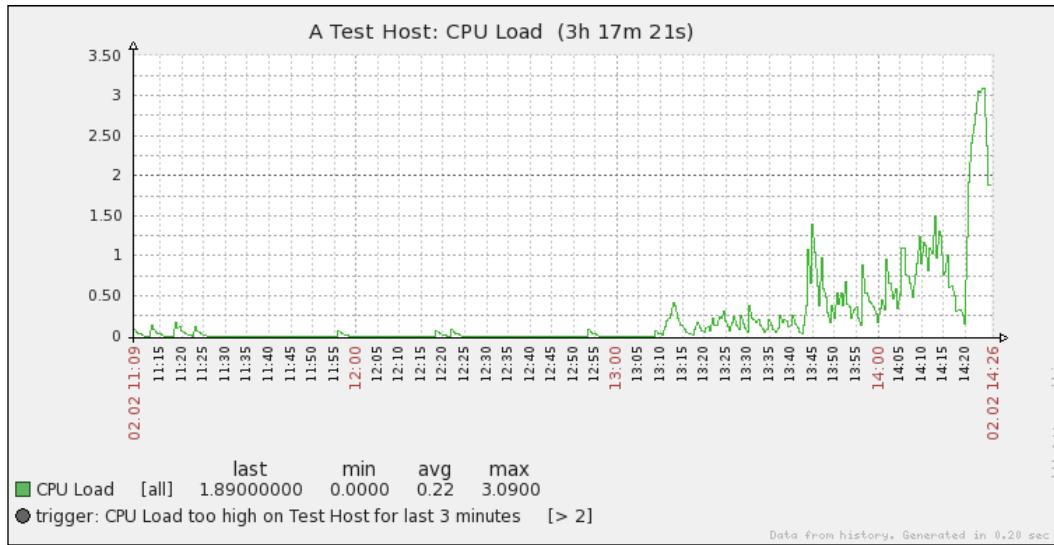
A trigger can also be in an UNKNOWN state. This happens after a trigger has been edited and if there is not enough data to determine current state. Events for changing to or from the UNKNOWN state do not match action conditions.

Let's create some load

Right, so we configured e-mail sending. But it's not so interesting until we actually receive some notifications. So let's increase the load on our test system. In the console, launch:

```
$ cat /dev/urandom | md5sum
```

This grabs a pseudorandom, never ending character stream and calculates the MD5 checksum on it, so system load should increase as a result. You can observe the outcome as a graph – navigate to **Monitoring** | **Latest data** and click on **Graph** for our single item again.



Notice how the system load has climbed. If your test system can cope with such a process really well, it might not be enough—in that case you can try running multiple such MD5 checksum calculation processes simultaneously.

Allow four minutes to pass and open **Monitoring | Triggers**. You should see the trigger **CPU Load too high on Test Host for last 3 minutes** visible with red, flashing **PROBLEM** text in the **Status** column.

Severity	Status	Last change	Age	Acknowledged	Host	Name	Comments
Not classified	PROBLEM	01 Feb 2010 19:22:52	1m 1s	Acknowledge (1)	A Test Host	CPU Load too high on Test Host for last 3 minutes	Add

Remember, the flashing indicates that a trigger has recently changed state, which we just made it do with that increased system load.

If you have a new e-mail notification, you should already be aware of this state change before opening **Monitoring | Triggers**, though... If all went as expected, you should have received an e-mail, informing you about the problem, so check your e-mail client if you haven't yet. There should be a message with an identical subject and body—"CPU Load too high on Test Host for last 3 minutes: PROBLEM".



Did the mail fail to arrive? This is most often caused by some configuration in the mail delivery chain preventing the message from passing. If possible, check your e-mail server's log files, as well as network connectivity and spam filters.

You can stop all MD5 checksum calculation processes now with a simple *Ctrl + C*. The trigger should then change status to **OK**, though you should allow at least the configured period of 30 seconds to pass.

Again, check your e-mail — there should be another message, this time informing you that it's alright now, having both the subject and body as "CPU Load too high on Test Host for last 3 minutes: OK".

Congratulations, you have set up all required configuration to receive alerts whenever something goes wrong, and also when things are back to normal. Let's recall what we did and learned:

- Created a host. Hosts are monitored device representations in Zabbix that can have items attached.
- We also created an item, which is a basic way to get information into Zabbix. Remember, the unique item identifier is **key**, which is also the string specifying what data will actually be gathered. A host was required to attach this item to.
- We explored a simple graph that was immediately available for the item without any configuration. The easy to use time period selection controls allowed us to view any period and quickly zoom in for drilldown analysis.
- Having data alone already is an achievement, but defining what a problem is frees us from manually trying to understand a huge amount of values. That's where **triggers** come in. They contain expressions that define thresholds.
- Have a list of problems instead of raw data is a step forward, but it would still require someone looking at the list. We'd prefer being notified instead — that's what actions are for. We were able to specify who and when should be notified.

Basic item configuration

We rushed through configuration of our simple item, so you might have gotten curious about what those parameters we didn't change or talk about. Let's take a quick look at what can be monitored and what can we configure about each item.

Zabbix can monitor quite a wide range of system characteristics. Functionally, we can split them into categories, while technically each method used corresponds to an item type.

Monitoring categories

Let's take a look at the generic categories that we can keep an eye on. Of course, this is not an exhaustive list of things to monitor – consider this as an example subset of interesting parameters. You'll soon discover many more areas to add in the Zabbix configuration.

Availability

While the simplified example we started with (the unlucky administrator in a party, remember him?) might not frighten many, there are more nightmare scenarios available than we'd like to think about. Various services can die without a sign until it's too late, and a single memory leak can bring system down easily.

We'll try to explore the available options to make sure such situations are detected as early as possible; to help our administrator deal with disk space problems during the working day and find out that an important service has died because of a database hiccup just as he goes through the door.

Performance

Performance is one of several holy grails in computing. Systems are never fast enough to accommodate all needs, so we have to balance desired operations with available resources. Zabbix can help you with both evaluating the performance of a particular action and monitoring the current load.

You can start with simple things such as network performance; as indicated by a ping roundtrip or the time it takes for a website to return content, and move forward with more complex scenarios such as the average performance of a service in a cluster coupled with disk array throughput.

Security

Another holy grail in computing is security. A never ending process where you are expected to use many tools, one of which can be Zabbix.

Zabbix can, independently of other verification systems, check simple things such as open ports, software versions, and file checksums. While these would be laughable as the only security measures, they can turn out to be quite valuable additions to already existing processes.

Management

System management involves doing many things, and that means following a certain set of rules in all of those steps. Good system administrators never fail at that, except when they do.

There are many simple and advanced checks you can use to inform you about tasks to perform or problems that arise when configuring systems – that means cross-platform notifications about available upgrades, checking that the DNS serial number is updated correctly, and a myriad of other system management pitfalls.

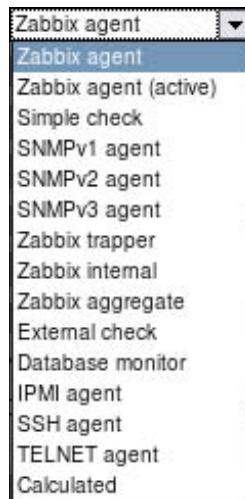
Efficiency

While generally considered a subset of availability or performance, some aspects of efficiency do not quite fit in there. Efficiency could be considered the first step to improved availability and performance, which increases the importance of knowing how efficient your systems are.

Efficiency parameters will be more service specific than others, but some generic examples might include Squid hit ratios and MySQL query cache efficiency. Other applications, including custom in-house ones, might provide other efficiency measuring methods.

Item types

As explored before, Zabbix gathers all its data within items. But surely we'll want to get information in more ways than through the Zabbix agent only – what are our options?



This is the item type configuration dropdown, when editing an item. We pretty much skipped this selection when creating our item because the default value suited us. Let's take a quick look at the types available now.

- **Zabbix agent:** This is the default type. Server connects to agent and gathers data.
- **Zabbix agent (active):** This can be considered as the opposite of previous type. Zabbix agent gathers data and connects to the server as needed.
- **Simple check:** As the name implies, this type groups simple checks that are performed by server. This includes checking for open TCP ports, ICMP ping, and so on.
- **SNMP agents:** These three types deal with gathering SNMP data. Versions, obviously, select protocol version to use when connecting to the monitored host.
- **Zabbix trapper:** This item type accepts incoming data instead of querying for it. It is most widely used for SNMP trap receiving, but useful for any data you might want to feed into Zabbix.
- **Zabbix internal:** This groups items that gather information about internal state of Zabbix.
- **Zabbix aggregate:** These items aggregate values across a host group. It is mostly useful for clusters where overall state is more important than state of individual machines.
- **External check:** External checks allow the Zabbix server to execute external command and store the returned value in the item. This allows it to pass along any information that isn't accessible with any of the other item types.
- **Database monitor:** This type includes built-in native checks for querying various database parameters.
- **IPMI agent: Intelligent Platform Management Interface (IPMI)** is a specification for managing and monitoring (which we're mostly after) systems, especially for out of band solutions. The IPMI item type allows direct access to this data.
- **SSH agent:** It is possible to directly query a host with SSH and retrieve shell command output. This check supports both password and key-based authentication.
- **TELNET agent:** For some systems where SSH is unavailable, direct Telnet check can be used. While insecure, it might be the only way to access some devices, including older generation switches or UPSes.
- **Calculated:** These are advanced items that allow to create new values from other, pre-existing Zabbix items without duplicate data retrieval.

While all these types might look a bit confusing at this point, an important thing to remember is that they are available for your use, but you don't *have* to use them. You can have a host with a single ICMP ping item, but if you want to monitor more, the advanced functionality will always be there.

As you might have noticed, the item type is set per individual item, not per host. This allows for great flexibility when setting up monitored hosts. For example, you can use ICMP to check general availability, a Zabbix agent to check the status of some services and simple TCP checks for others, trapper to receive SNMP traps, and IPMI to monitor parameters through the management adapter – all on the same host. The choice of item type will depend on network connectivity, the feature set of the monitored host, and the ease of implementation. Zabbix will allow you to choose the best fit for each individual item.

How items can be monitored

While that covered categories and item types, we skipped some other parameters when creating the item, so it might be helpful to learn about basic values that will have to be set for most item types. Let's take a quick look at the item creation/editing window again:

The screenshot shows the 'Item 'A Test Host:CPU Load'' configuration window. The 'Host' field is set to 'A Test Host'. The 'Description' field contains 'CPU Load'. The 'Type' dropdown is set to 'Zabbix agent'. The 'Key' field is 'system.cpu.load'. The 'Type of information' dropdown is set to 'Numeric (float)'. The 'Units' field is empty. The 'Use multiplier' dropdown is set to 'Do not use'. The 'Update interval (in sec)' input field is '30'. Under 'Flexible intervals (sec)', it says 'No flexible intervals'. A 'New flexible interval' section shows 'Delay' as '50' and 'Period' as '1-7,00:00-23:59', with an 'Add' button. The 'Keep history (in days)' is '90' and 'Keep trends (in days)' is '365'. The 'Status' is 'Active'. The 'Store value' dropdown is 'As is'. The 'Applications' section shows '-None-' and an empty list. At the bottom, the 'Group' dropdown is 'Discovered Hosts', and there are 'Save', 'Cancel', 'Add to group', and 'do' buttons.

- **Host:** Shows which host this item will be attached to. If we are entering new item creation form from the item list that is filtered for a particular host, this field is pre-populated.
- **Description:** This can be considered as the name of the item. That's what you will see in most places where the data is referred to.
- **Type:** This is the main property, affecting other fields and the way item data is gathered, as discussed above.
- **Key:** This is the property that explicitly specifies what data has to be gathered for this item. It is sort of a technical name for the item. Key value must be unique per host. For some other item types this field might be **SNMP OID** or **IPMI sensor**.
- **Type of information:** This allows you to choose the data type that will be gathered with the item. You'll have to set it according to the values provided: integers, decimals, and so on.
- **Data type :** This property provides a way to query data in hexadecimal or octal format and convert it to decimal values automatically. Some SNMP capable devices (mostly printers) are exposing information in those formats.
- **Units:** This property allows you to choose the unit to be displayed besides data and for some units Zabbix will calculate corresponding conversions as needed (called "human-readable" in many tools, so you get 32.5 MB instead of the same value in bytes).
- **Use multiplier:** This property multiplies incoming data with the value specified here. This is useful if data arrives in one unit but you want to store it as another (for example; if the incoming data is in bytes, but you want it in bits, you'd use a multiplier of 8).
- **Update interval:** This sets the interval between data retrieval attempts.
- **Flexible intervals:** This setting allows you to modify the update interval during some specific times – either because you have no need for a particular item during the night, or because you know some particular service will be down; for example during a backup window.
- **Keep history:** This sets the time period that actual retrieved values are stored in the database.
- **Keep trends:** This does the same as the **Keep history** option, except for trends. Trends are data calculated from history, and averaged for every hour to reduce long term storage requirements.
- **Status:** This enables or disables the item.

- **Store value:** This property is for numeric data only and allows the Zabbix server to do some basic calculations on the data before inserting it into the database; such as calculating difference between two checks for counter items.
- **Applications:** This property makes it possible to do logical grouping of items, for example, in **Monitoring | Latest data** screen.

Don't worry if these short descriptions didn't answer all of your questions about each option. We'll dig deeper into each of these later, and there are more options available for other item types as well.

Summary

This was the chapter where we finally got some real action; monitoring an item, creating a trigger, and getting a notification on this trigger. We also explored the Zabbix frontend a bit and looked at the basic item parameters. Let's review what basic steps were required to get our first alert.

- We started by creating a **host**. In Zabbix, everything to be monitored is attached to a logical entity called a host.
- Next we created an **item**. Being the basis for information gathering, items define parameters about monitored metrics, including what data to gather, how often to gather it, how to store the retrieved values, and other things.
- After item we created a **trigger**. Each trigger contains an expression that is used to define thresholds. For each trigger a severity can be configured as well.
- To let Zabbix know how to reach us, we configured the e-mail settings. This included specifying an e-mail server for the **media type** and adding **media** in our user profile.
- As the final configuration step we created an **action**. Actions are configuration entities that define actual operations to perform and can have conditions to create flexible rules on what to do about various events.
- Well, we actually did one more thing to make sure it all works – we **created a problem**. It is useful to test your configuration, especially when just starting with Zabbix. Our configuration was correct, so we were promptly notified about the problem.

While this knowledge is already enough to configure a very basic monitoring system, we'll have to explore other areas before it can be considered a functional one.

3

Monitoring with Zabbix Agents and Basic Protocols

Now that we have explored the basics of information gathering and acting upon it in Zabbix, let's take a closer look at two simple and widely used methods for obtaining data – the already mentioned Zabbix agents, and so called simple checks; which include TCP connectivity and ICMP checks.

Using Zabbix agent

Previously, we installed the Zabbix agent on the same host and monitored a single item for it. It's now time to expand and see how inter-host connectivity works.

To continue, install the Zabbix agent on another host. Compiling an agent only is done in a similar way to how we compiled everything in the beginning, but instead of the full configure line, you just need a single flag this time:

```
$ ./configure --enable-agent
```

Configuration should complete successfully and the following summary lines are important:

Enable server:	no
Enable proxy:	no
Enable agent:	yes

If the output you see matches the above output, continue by issuing the following command:

```
$ make
```

Compilation should complete without any errors, and it should do so relatively quickly.



You can also decide to install distribution packages of Zabbix agent daemon.



Once it's done, create the installation package as root:

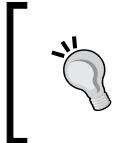
```
# checkinstall --nodoc --install=yes -y
```

If you install distribution packages, don't worry when the agent daemon has an older version than server. This is supported and should work well. In fact, Zabbix agent daemon version 1.1 works quite well with a 1.8 version server. The other way usually won't work; you should avoid using an older server with new agents.

Staying with an older agent can be more convenient as you already have one installed and working well. When setting up new ones it is suggested to go with the latest one, as it might have bugs fixed, improved performance, more supported items for a particular platform, or other benefits.

We also have to create a configuration file, so, as root, copy example configuration (usually not needed when installing from distribution packages):

```
# mkdir /etc/zabbix  
# cp misc/conf/zabbix_agentd.conf /etc/zabbix
```



Make sure to use and edit `zabbix_agentd.conf`, with **d** in the name. The other file, `zabbix_agent.conf`, is used by the limited functionality `zabbix_agent`, the use of which is generally discouraged.



With the agent installed, now is the time to start it up. As this is a distribution-specific process, either refer to *Chapter 1* for example startup scripts and their usage, or, if you installed from distribution packages, use the startup script supplied with them.

We also have to add this new host to the configuration, so open **Configuration | Hosts**, make sure that the dropdown in the upper-right corner says **Hosts** and **Group** dropdown says **Linux servers**. Click on the **Create Host** button and fill in the form.

The screenshot shows the 'Host' configuration dialog in Zabbix. The 'Name' field is filled with 'Another Host'. In the 'Groups' section, 'Linux servers' is selected under 'In Groups'. Under 'Other Groups', there are four options: 'Discovered Hosts', 'Templates', 'Windows servers', and 'Zabbix Servers'. Below these groups, there are several input fields: 'DNS name', 'IP address' (containing '10.1.1.100'), 'Connect to' (set to 'IP address'), 'Zabbix agent port' (set to '10050'), 'Monitored by proxy' (set to '(no proxy)'), 'Status' (set to 'Monitored'), and 'Use IPMI' (unchecked). At the bottom right are 'Save' and 'Cancel' buttons.

- **Name:** Feel free to choose a descriptive name, or simply enter **Another Host**
- **DNS name or IP address:** Depending on which connection method you want to use, fill in the corresponding field
- **Connect to:** Choose either **IP address** or **DNS name** according to the method you selected in the previous step

When done, click **Save**.

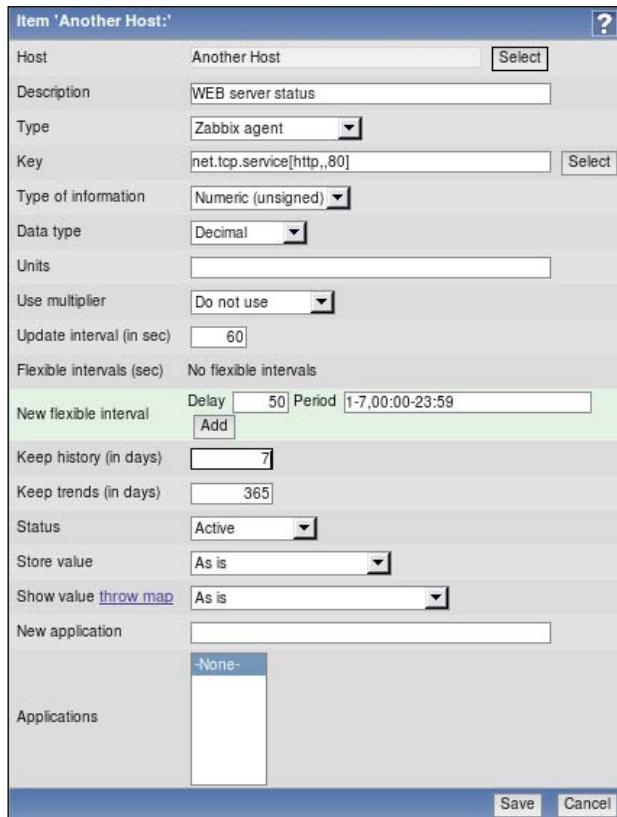
Passive items

The item we created before was a so called "passive" item, which means that Zabbix server initiates a connection to agent every time configuration specifies it has to be checked. In most locations they are simply referred to as being of type "Zabbix agent". Let's create another passive item to check for the remote host.

Open **Configuration | Hosts**, click on **Items** next to the host you just created then click on **Create Item** button.

- **Description:** Enter WEB server status
- **Key:** Enter `net.tcp.service[http,,80]` (that's two subsequent commas preceding 80)
- **Update interval:** Change to 60 from the default (30) – once a minute should be more than enough for our needs
- **Keep history:** Change to 7 from the default (90) – that's still a whole week of exact minutely service status records kept

The end result should be as follows:



But what's up with that ",,80" added to the service name? Click the **Select** button next to the **Key** field. This opens a window with a nice list of keys to choose from, along with a short description of each.

STANDARD ITEMS		Type	Zabbix agent
Key	Description		
agent.ping	Check the agent usability. Always return 1. Can be used as a trigger condition.		
agent.version	Version of zabbix_agent(d) running on monitored host. String		
kernel.maxfiles	Maximum number of opened files supported by OS.		
kernel.maxproc	Maximum number of processes supported by OS.		
net.if.collisions[if]	Out-of-window collision. Collisions count.		
net.if.in[if <,mode>]	Network interface input statistic. Integer value. If mode is missing, it's assumed to be "bytes".		
net.if.out[if <,mode>]	Network interface output statistic. Integer value. If mode is missing, it's assumed to be "bytes".		
net.tcp.dns.query[ip, zone, type]	Performs a query for the record type specified by the parameter type for the specified IP address and zone.		
net.tcp.dns[ip, zone]	Checks if DNS service is up. 0 - DNS is down, 1 - DNS is up.		
net.tcp.listen[port]	Checks if this port is in LISTEN state. 0 - it is not, 1 - it is in LISTEN state.		
net.tcp.port[<ip>, port]	Check, if it is possible to make TCP connection to the port number port from the IP address ip.		
net.tcp.service.perf[service <,ip> <,port>]	Check performance of service "service". 0 - service is down.		
net.tcp.service[service <,ip> <,port>]	Check if service server is running and accepting connections.		
perf_counter[counter]	Value of any performance counter, where parameter is the counter name.		
proc_info[<process>, <attribute>, <type>]	Different information about specific process(es).		
proc.mem[<name> <,user> <,mode> <,cmdline>]	Memory used by process with name name running under user user in mode mode with commandline cmdline.		
proc.num[<name> <,user> <,state> <,cmdline>]	Number of processes with name name running under user user in state state with commandline cmdline.		
service.state[service]	State of service. 0 - running, 1 - paused, 2 - start pending, 3 - stopped.		

The **Type** dropdown in the upper-right corner will allow you to switch between several item types; we'll discuss the other types later. For now, find **net.tcp.service** in the list and look at the description. There are two things to learn here. First, we didn't actually have to add that 80—it's a port, and, given that default already is 80, adding it was redundant. However, it is useful if you have a service running on a non-standard port. Second, there's a key list just one click away to give you quick hint in case you have forgotten a particular key or what its parameters should be like.

This key, **net.tcp.service**, is a bit special—it tries to verify that the corresponding service actually does respond in a standard manner, which means the service must be explicitly supported. As of writing this, Zabbix supports the following services for the `net.tcp.service` key:

- FTP
- HTTP
- IMAP
- LDAP
- NNTP

- NTP
- POP
- SMTP
- SSH
- TCP

The last service is a bit special in its own way. While others perform service-specific checks, TCP just checks the TCP connection. It's closer to a key you can see a couple of rows above in the itemlist, net.tcp.port. As the description says, this one just tries to open TCP connection to any arbitrary port without performing any service-specific checks on the returned value. If you try to use an arbitrary service string that is not supported, you would simply get an error message saying **Not supported by Zabbix agent**.

Feel free to look at other available keys – we will use a couple of them later as well, then close this pop up and click on **Save**.

You probably have already noticed the green strip at the top of the screen when some operation successfully completes. This time there's also control **Details** available, so click on it to expand details.



You can click on **Details** again to collapse contents. Of course, this can be done whenever **Details** link is available after some operation.

Now, we could go over to **Monitoring | Latest data** and wait for the values appearing there but that would be useless. Instead, after a couple of minutes you should visit **Configuration | Hosts**. Depending on your network configuration, you might see a red icon with the letter **Z** in it. This icon represents errors that have happened when attempting to gather data from the Zabbix agent.



To see the actual error message, move your mouse cursor over the icon, and a tooltip will open. Clicking the error icon will make the tooltip permanent and allow you to copy the error message

Items	Triggers	Graphs	DNS	IP	Port	Templates	Status	Availability
Items (1)	Triggers (0)	Graphs (0)	-	10.1.1.100	10050	-	Monitored	
Get value from agent failed: Cannot connect to [10.1.1.100:10050] [Interrupted system call]								



The two other icons represent SNMP and IPMI data gathering errors. We will monitor SNMP and IPMI devices later.

If you see an error message similar to **Get value from agent failed: Cannot connect to [10.1.1.100:10050] [Interrupted system call]** (most likely, with a different IP address), it means that Zabbix server was unable to connect to agent daemon port. This can happen because of a variety of reasons, the most common being a firewall – either a network one in between Zabbix server and the remote host, or a local one on the remote host. Make sure to allow connections from Zabbix server to the monitored machine on port 10050. Then mark the checkbox in the host list next to the remote host, choose **Activate selected** from the dropdown below the item list, then click **Go** button and confirm the pop up. While Zabbix will retry connecting to unavailable hosts, this will speedup the reconnection attempt. Alternatively, you can click on the **Monitored** text in the **Status** column, which will turn into **Not monitored**, click on that again. Clicking on **Status** column values toggles host monitoring.

If you did this correctly (or if you did not have firewall blocking the connection), you could again go to **Monitoring | Latest data**, only that would be pointless, again. To see why, refresh the host list. Soon, you should see the Zabbix agent status icon turn red again; and moving your mouse cursor over it will reveal another error message, **Got empty string from [10.1.1.100]. Assuming that agent dropped connection because of access permissions**. Now that's different. What access permissions is it talking about, why did it work for our first host?

From Zabbix server, execute:

```
$ telnet 10.1.1.100 10050
```



You should always verify network connectivity and access permissions from the Zabbix server. Doing that from another machine can have wildly differing and useless results.

Replace the IP address with the one of your remote host. You should see the following output, and the connection should be immediately closed:

```
Trying 10.1.1.100...
Connected to 10.1.1.100.
Escape character is '^].
Connection closed by foreign host.
```

Now, try the same with localhost.

```
$ telnet localhost 10050
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^].

```

Notice how this time the connection is not closed, so there's a difference in configuration. To close the connection, press *Ctrl+J*, as instructed, then enter *quit*.

```
^]
telnet> quit
Connection closed.
```

It turns out that configuring the Zabbix agent daemon on another machine is going to be a tiny bit harder than before.

As opposed to the installation on the Zabbix server, we have to edit the agent daemon configuration file on the remote machine. Open `/etc/zabbix/zabbix_agentd.conf` as root in your favorite editor and take a look at `Server` parameter. It is currently set to `127.0.0.1`, which is the reason we didn't have to touch it on the Zabbix server. As the comment states, this parameter should contain Zabbix server IP address, so replace `127.0.0.1` with the correct server address here. Save the file and restart the agent daemon. To verify the change, try telnetting to the remote machine again.

```
$ telnet 10.1.1.100 10050
```

This time, the outcome should be the same as we had with localhost, so close the connection the same way.



The port, specified in host properties, is used for passive Zabbix agent checks only. If such items are not configured, this port is simply ignored.

To speed up refreshing of our problematic item, open **Configuration | Hosts**, click on **Items** next to **Another Host**, and reactivate it just like we did with the host before – either by marking the checkbox and choosing **Activate selected**, or by clicking on **Status** column control.

You can verify that this has worked by refreshing the item list after few seconds.

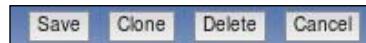
Finally, it should be worth opening **Monitoring | Latest data**, where we should see two monitored hosts now, each having a single item – click on the + icon to expand the newly added item.

Host	+ Description	Last check	Last value	Change	History
A Test Host	⊖ - other - (1 items)				
	CPU Load	09 Feb 2010 14:29:22	0.040000	-0.03	Graph
Another Host	⊖ - other - (1 items)				
	WEB server status	09 Feb 2010 14:29:24	1	-	Graph

By the way, did you notice the + icon in the table heading, next to the **Description** heading? That one can be used to expand and collapse all entries.

Cloning items

Let's try to monitor another service now; for example, the one running on port 22, SSH. To keep things simple for us, we won't create an item from scratch – instead, go back to **Configuration | Hosts**, click on **Items** next to **Another Host** and click on link **WEB server status** in the **Description** column. This will open item editing screen, showing all the values we entered before. Notice how this time there are more buttons available at the bottom. Along with the **Save** and **Cancel** buttons we also have **Delete**, which obviously, deletes the currently open item. We don't want to do that now. Instead, click **Clone**.



Notice how the opened form proposes to create new item, but this time all values are set to what the original item we cloned had. Click **Save** – but that should fail. Remember, we talked about the key being unique per host, that's what the error message says as well.

Details	ERROR: Cannot add item
✖ An item with the Key [net.tcp.service[http,,80]] already exists for host [Another Host]. The key must be unique.	

The item editing form is still open, so we can correct our mistake. Do the following modifications:

- **Description:** Change it to "SSH server status"
- **Key:** Change `http`, `,80` to `ssh`, so that it looks like this :
`net.tcp.service[ssh]`

That's all we have to do this time, so click **Save** again. This time, item should be added successfully. Now navigate to **Monitoring | Latest data**, where **Another Host** should have two items listed, **SSH server status** and **WEB server status**. Their status would depend on what services are running on the remote host. As it's remote, SSH most likely is running (and thus having a value of 1), but whether or not the web server is running would be specific to your situation.

Another Host	⊖ - other - (2 Items)				
	SSH server status	09 Feb 2010 14:34:25	1	-	Graph
	WEB server status	09 Feb 2010 14:34:24	1	-	Graph

 Monitoring a port is often done to make sure the service on it is available, but that is not a strict requirement. If some system is not supposed to have SSH available from internet, we could use such a check to verify that it has not been accidentally exposed either by inadvertent starting of the SSH daemon or an unfortunate change in the firewall.

This covers the basics of normal, or passive, Zabbix items where the server queries agents, so let's move to other item types.

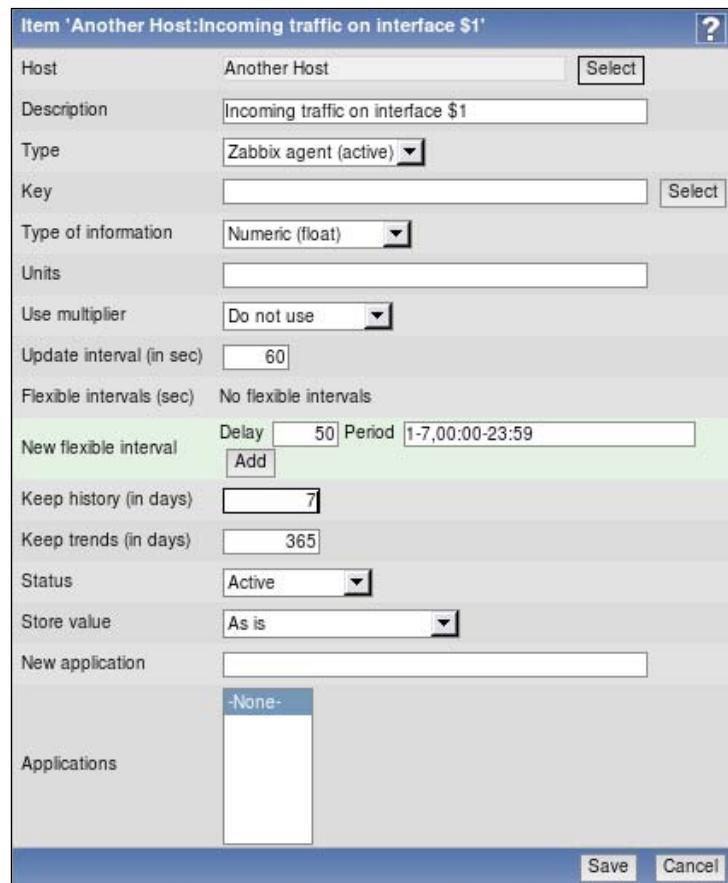
Active items

Passive Zabbix items are fine if you can connect to all the monitored hosts from the Zabbix server but what if you can't allow incoming connections to the monitored hosts because of security reasons, or you have to deal with a daft firewall admin?

This is where active items come into play. As opposed to passive items, for active items it's the agent that connects to the server, and the server never connects to the agent. When connecting, the agent downloads a list of items to check, then reports to the new data server periodically. Let's create an active item, but this time we'll try to use some help when selecting item key.

Open Configuration | Hosts, click on **Items** next to **Another Host** and click **Create Item**. For now, fill in these values:

- **Description:** Enter **Incoming traffic on interface \$1**
- **Type:** Select **Zabbix agent (active)**
- **Type of information:** Choose **Numeric (float)**
- **Update interval:** Enter **60**
- **Keep history:** Enter **7**



We'll do something different with the **Key** field, click on **Select** and in the upcoming dialog that we already saw before, click on **net.if.in[if <,mode>]**. This will fill in the chosen string.

<code>net.if.in[if <,mode>]</code>	Network interface input statistic. Integer value. If mode is missing bytes is used.
--	---

Replace the content in the square brackets with `eth0`, so that the field contents read `net.if.in[eth0]`. When done, click **Save**.

Open **Monitoring | Latest data** and check whether new values have arrived.

Description	Last check	Last value	Change	History
- other - (3 Items)				
Incoming traffic on interface eth0	-	-	-	Graph
SSH server status	24 Mar 13:48:24	1	-	Graph
WEB server status	24 Mar 13:48:25	1	-	Graph

Well, doesn't look like it. You could wait a bit to be completely sure, but, most likely, no data will appear for this new active item. Which means we're in for another troubleshooting session.

First we should test basic network connectivity. Remember, active agents connect to the server, so we have to know which port they use (by default, it's port 10051). So let's start by testing from the remote monitored machine that it can connect to the Zabbix server:

```
$ telnet <Zabbix server IP or DNS name> 10051
```

This should produce output similar to the following:

```
Trying <Zabbix server IP>...
Connected to <Zabbix server IP or DNS name>.
Escape character is '^]'.
```

As before, press *Ctrl+J* and enter in the resulting prompt:

```
telnet> quit
Connection closed.
```

Such a sequence indicates that the network connection is working properly. If it didn't, verify possible network configuration issues, including network firewalls and local firewall on the Zabbix server. Make sure to allow incoming connections on port 10051.



Both agent and server ports for Zabbix are registered with the Internet Assigned Numbers Authority.

So there might be something wrong with the agent, let's take a closer look. On the remote machine, open the configuration file `/etc/zabbix/zabbix_agentd.conf` and make sure that configuration option `DisableActive` is either commented out, or set to 0 (it should be commented out by default).

 While you have the configuration file open, notice another parameter just below `DisablePassive`. As the name indicates, this parameter will prevent the agent from listening on incoming connections from the server, so you can customize your agents to support either one or both of the methods. Disabling both won't work; the agent daemon will complain and refuse to start up—and it's correct, starting with both disabled would be a pointless thing to do.

As the `DisableActive` directive is not set, we could try to look at the agent daemon's log file, so find configuration parameter `LogFile`—by default, it should be set to log to a file `/tmp/zabbix_agentd.log`. Open this logfile and look for any interesting messages regarding active checks. Each line will be prefixed with a PID and timestamp in the syntax `PID:YYYYMMDD:HHMMSS`. You'll probably see lines similar to these:

```
PID:YYYYMMDD:HHMMSS zabbix_agentd active check started [<Zabbix server IP>:10051]
```

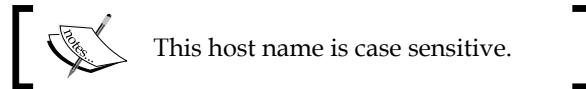
It looks like agent is requesting active check list, but nothing happens after that. Probably something is misconfigured, but this log file entry doesn't help much. Let's try to make agent more verbose—again, open `/etc/zabbix/zabbix_agentd.conf`. This time, look for the parameter named `DebugLevel` and set it to 4 (as comment warns, that's debugging level and will produce quite a lot of output). Save and close configuration file, then restart Zabbix agent daemon. Take a look at the log file `/tmp/zabbix_agentd.log` again—you should be able to find log entries similar to these:

```
PID:YYYYMMDD:HHMMSS refresh_active_checks('<Zabbix server IP>',10051)
PID:YYYYMMDD:HHMMSS Sending [{ "request": "active checks", "host": "Zabbix Server"}]
PID:YYYYMMDD:HHMMSS Before read
PID:YYYYMMDD:HHMMSS Got [{ "response": "success", "data": [] }]
```

Now that's a bit more verbose—agent is requesting active check list, but receives back nothing. Wait, but it's identifying itself as `Zabbix Server`, which can't be right. Edit `/etc/zabbix/zabbix_agentd.conf` and set `DebugLevel` back to 3, otherwise logfile will be filled with lots of debugging messages.

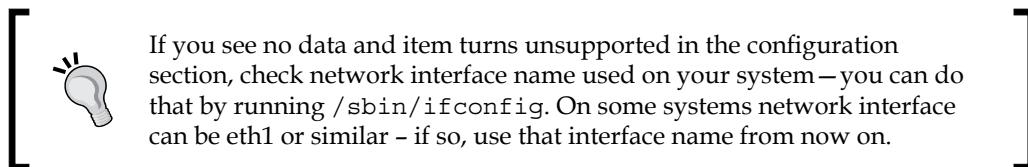
Take a look a bit lower in the configuration file – there's parameter, named `Hostname`, which currently reads `Zabbix Server`. Given that comment for this parameter says "Unique hostname. Required for active checks.", it has to be what we're after. Change it to `Another Host`, save and close configuration file, then restart the Zabbix agent daemon. Check for new entries at `/tmp/zabbix_agentd.log`—there should be no more errors.

If there still are errors about host not found on the server, double check that hostname in Zabbix frontend host properties and agent daemon configuration file (the one just changed) match.



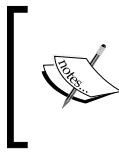
It's now time to return to the frontend and see whether data has started flowing in at **Monitoring | Latest data** section.

+ Description	Last check	Last value	Change	History
- other - (3 Items)				
Incoming traffic on interface eth0	09 Feb 2010 15:37:02	81626854.000000	+24792.00	Graph
SSH server status	09 Feb 2010 15:37:25	1	-	Graph
WEB server status	09 Feb 2010 15:37:24	1	-	Graph



Great, it indeed is but the values look really weird. If you wait for a while, you'll see how the number in **Last check** column just keeps on increasing. So what is it? Well, network traffic keys gather data from interface counters – that is, the network interface adds up all traffic, and this total data is fed into Zabbix database. This has one great advantage – even when data is polled at large intervals, traffic spikes would not go unnoticed as the counter data is there but it also makes data pretty much unreadable for us, and graphs would also look like an ever-increasing line (if you feel like it, click on **Graph** link for this item). Luckily, Zabbix provides a built-in capability to deal with data counters like this. Open **Configuration | Hosts**, then click on **Items** next to **Another Host** and click on **Incoming traffic on interface eth0** in the **Description** column. Change the **Store value** dropdown to read **Delta (speed per second)**, then click **Save**.

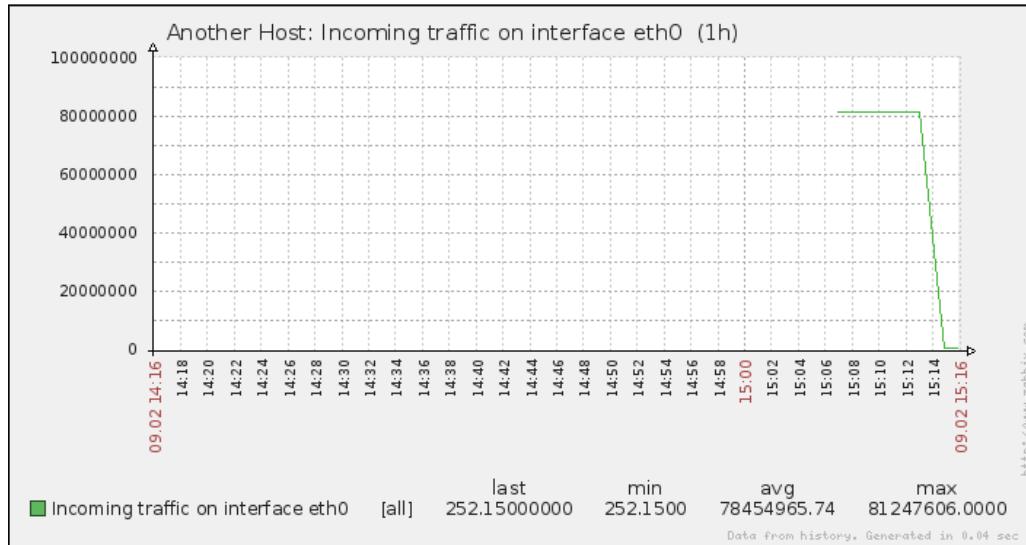
Check out **Monitoring | Latest data** again.



Keep in mind that configuration changes might take up to three minutes in the worst case to propagate to Zabbix agent—one minute to get into server cache, and two minutes until agent would refresh its own item list.

+ Description	Last check	Last value	Change	History
- other - (3 Items)				
Incoming traffic on interface eth0	09 Feb 2010 15:39:02	208.083333	-81626645.91	Graph
SSH server status	09 Feb 2010 15:38:25	1	-	Graph
WEB server status	09 Feb 2010 15:38:24	1	-	Graph

That's better, Zabbix now automatically calculates the change between every two checks (that's what delta is for) and stores that, but the values still don't seem to be too user friendly. Maybe they're better in the graph—let's click on the **Graph** link to find out.



Ouch. While we can clearly see the effect our change had, it also has left us with a very ugly historical data. The upper part of that graph represents total counter (thus showing total since the monitored system was started up), but the lower part represents correct (delta) data. You can also take a look at values numerically—see the dropdown at the upper-right corner, which reads **Graph** currently. Choose **500 latest values** in there.

Another Host: Incoming traffic on interface eth0		500 latest values	As plain text
Timestamp	Value		
2010.Feb.09 15:17:02	259.2373		
2010.Feb.09 15:16:03	228.9344		
2010.Feb.09 15:15:02	252.15		
2010.Feb.09 15:13:02	81247606		
2010.Feb.09 15:12:02	81232555		
2010.Feb.09 15:11:03	81217900		
2010.Feb.09 15:10:02	81206305		
2010.Feb.09 15:09:02	81192875		
2010.Feb.09 15:08:02	81179936		
2010.Feb.09 15:07:02	81164705		

In this list, we can nicely see the change in data representation, as well as the exact time when the change was performed. But those huge values have come from the counter data, and they pollute our nice, clean graph by being so much out of scale—we have to get rid of them. Open **Configuration | Hosts** and click on **Items** next to **Another Host**, then mark the checkbox next to **Incoming traffic on interface eth0** item and look at the activity dropdown positioned at the bottom of the itemlist.



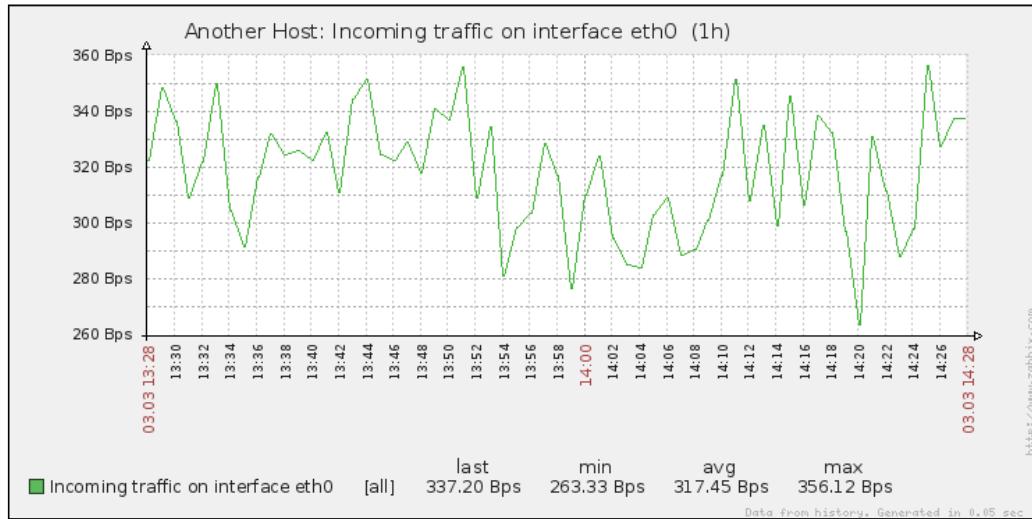
The second entry from the bottom; named **Clear history for selected** probably does what we want, so select it. Notice the **Go** button next to the activity selection—it shows the amount of entries selected, so we always know on how many elements we are operating on. Click this button. You should get a JavaScript pop up, asking for confirmation to continue. While history cleaning can take a long time with large datasets, in our case, it should be nearly instant, so click the **OK** button to continue. This should get rid of all history values for this item, including the huge ones.

Still, looking at the y-axis in that graph, we see those large values that are hard to grasp easily. It would be so much better if Zabbix knew how to calculate it in bytes or similar units. Right, so navigate to **Configuration | Hosts** and click **Items** next to **Another Host**, then click **Incoming traffic on interface eth0** in the **Description** column. Edit the **Units** field and enter "B", then click **Save**.

Let's see whether there's any improvement in **Monitoring | Latest data**.

+ Description	Last check	Last value	Change	History
[- other - (4 items)]				
Incoming traffic on interface eth0	03 Mar 2010 14:39:13	8.98 MBps	+8.98 MBps	Graph
SSH server status	03 Mar 2010 14:38:25	Up (1)	-	Graph
WEB server status	03 Mar 2010 14:39:24	Up (1)	-	Graph
Zabbix agent version	02 Mar 2010 20:31:00	1.8.1	-	History

Wonderful, data is still arriving. Even better, notice how Zabbix now automatically calculates MB where appropriate. Let's look at the network traffic, click **Graph**.



Take a look at y-axis – now units are calculated there as well to make the graph readable, and unit calculations are retroactively applied to the previously gathered values.

One parameter that we set, update interval, could have been smaller, thus resulting in a better looking graph. But it is important to remember that the smaller the intervals you have on your items, the more data Zabbix has to retrieve, and each second more and more data has to be inserted into the database and more calculations have to be performed when displaying this data. While it would have made no notable difference on our test system, you should try to keep intervals as large as possible.

So far we have created items that gathered numeric data – either integers, or decimal. Let's create another one, a bit different this time. As usual, open **Configuration | Hosts** and click on **Items** next to **Another Host**. Before continuing with item creation, let's look at what helpful things are available in configuration section, particularly for items. If we look above the item list, we can see navigation and information bar.

[Hosts list](#) [Applications \(0\)](#) [Triggers \(0\)](#) [Graphs \(0\)](#) **Host:** Another Host **DNS:** - IP: 192.168.3.93 **Port:** 10050 **Status:** Monitored **Availability:** Available

This area provides quick useful information about currently selected host – hostname, DNS, IP, configured Zabbix agent port and whether the host is monitored and available. What's even more important, on the left hand side it provides quick shortcuts back to the host list and other elements, associated with current host – applications, triggers and graphs. This is a handy way to switch between element categories for a single host without going through the host list all the time. But that's not all yet – click on **Filter** link just above this status bar. A sophisticated filter appears.

The screenshot shows a 'Filter' dialog box with various search criteria fields. At the top right are 'Filter' and 'Reset' buttons. The fields include:

- Host group: [text input] Select
- Type: [dropdown menu] all
- Type of information: [dropdown menu] all
- Status: [dropdown menu] all
- Triggers: [dropdown menu] all
- Template: [dropdown menu] all
- Host: [text input] Another Host Select
- Update interval (in sec): [text input]
- Keep history (in days): [text input]
- Keep trends (in days): [text input]
- Application: [text input] Select
- Description like: [text input]
- Key like: [text input]

Using this filter, we can make complex rules on what items to display. Looking at the left top corner of the filter, we can see that we are not limited to viewing items from a single host – we can also choose a hostgroup. When needed, we would make filter choices and click on the **Filter** link below. Currently it has only one condition – field **Host** contains **Another Host**, so **Items** link from the host list we used was the one which set this filter.



Host information and quicklink bar is only available when items are filtered for a single host.

Now look right below the main item filter – that is subfilter, which, as its header informs, only affects data already filtered by the main filter.

Subfilter [affects only filtered data!]	
Types	Zabbix agent (2) , Zabbix agent (active) (1)
Type of information	Numeric (float) (1) , Numeric (unsigned) (2)

The entries in the subfilter work like toggles – if we switch one on, it works as a filter on the data in addition to all other toggled subfilter controls. Let's click on **Zabbix agent** now. Notice how item list now contains two items only – that's what number **2** represented next to this subfilter toggle. But the subfilter itself now also looks different.

Subfilter [affects only filtered data!]	
Types	Zabbix agent (2) , Zabbix agent (active) (+1)
Type of information	Numeric (float) (0) , Numeric (unsigned) (2)

The option we enabled, **Zabbix agent**, is painted in green. **Numeric (float)**, instead, is grey – this means that activating this toggle in addition to already active ones would result in no items displayed at all. While **Numeric (unsigned)** toggle still has **2** listed to it – which shows that enabling it will change amount of items displayed to this value – **Zabbix agent (active)** toggle instead has **+1** next to it. This form represents the fact that activating this toggle will display one more item than currently, and is used for toggles in the same category, listed on the left. Currently subfilter has few entries, as it only shows present values. Once we will have additional and more different items configured, subfilter will expand. We have finished with exploring these filters, so click on **Create Item**.

Once you have many different hosts monitored by Zabbix, it's quite easy to forget what version of Zabbix agent daemon each host has – and even if you have automated software deploying in place, it is nice to be able to see at which version each host is, all in one place.

Enter the following values:

- **Description:** Enter **Zabbix agent version**
- **Type:** Select **Zabbix agent (active)** (we're still creating active items);
- **Key:** Click on **Select**, then choose second entry from the list – **agent.version**

- **Type of information:** Choose **Character**
- **Update interval:** Enter **86400**

When done, click the **Save** button.

There are two notable things we did. First, we set information type to **Character**, which reloaded the form, slightly changing available options. Most notably, fields that are relevant for numeric information were hidden, such as units, multiplier, and trends.

Second, we entered a very large update interval, 86400, which is equivalent to 24 hours. While that might seem excessive, remember what we will be monitoring here – Zabbix agent version, so it probably (hopefully) isn't changing several times per day. Depending on your needs, you might set it to even larger values, like a week.

To check out results of our work, open **Monitoring | Latest data**.

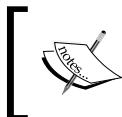
+ Description	Last check	Last value	Change	History
[- other - (4 items)]				
Incoming traffic on interface eth0	11 Feb 2010 10:58:57	192.41 B	+7.15 B	Graph
SSH server status	11 Feb 2010 10:59:25	1	-	Graph
WEB server status	11 Feb 2010 10:59:24	1	-	Graph
Zabbix agent version	11 Feb 2010 10:37:58	1.8.1	-	History

If you don't see the data, wait a while, it should appear eventually. When it does, you should see the version of the Zabbix agent installed on the remote machine listed – and it might be a higher number than displayed here, as new versions of Zabbix have probably been released. Notice one minor difference – while all the items we added previously have links named **Graph** in the **History** column, the last one has **History**. The reason is simple, for textual items graphs can't be drawn so Zabbix does not even attempt to do that.

Now, about that waiting – why did we have to wait for the data to appear? Well, remember how active items work? The agent queries the server for the item list it should report on, then sends in data periodically, but this item list checking is also done periodically. To find out how often, open configuration file `/etc/zabbix/zabbix_agentd.conf` on the remote machine and look for the parameter `RefreshActiveChecks`. The comment says that the default is two minutes, which is configured in seconds, thus listing 120 seconds. So in the worst case you might have had to wait for nearly three minutes to see any data, as opposed to normal or passive items, where the server would query the agent as soon as the configuration change is available in its cache. In a production environment with many agents using active items it might be a good idea to increase this value. Usually item parameters aren't changed that often.

Supported items

We created some items that use the Zabbix agent in both directions and gathered data. But those are hardly the only ones available. You could check out the list, available when creating an item again (**Configuration | Hosts**, click on **Items** for any host and click **Create Item** button, then **Select** button next to the **Key** field), to see what items are built-in for Zabbix items along with a short description for most of them.



Not all Zabbix agent items are available as both passive and active. For example, log and event log items (for gathering log file and Windows event log, respectively) are only available as active items.

Looking at the list, we can find out what categories of items Zabbix agents support natively—system configuration, network traffic, network services, system load and memory usage, file system monitoring, and others. But that does not mean everything you see there will work on any system that the Zabbix agent daemon runs on. As every platform has a different way of exposing this information, and some parameters might even be platform-specific, it isn't guaranteed that every key will work on every host.

For example, when the disk drive statistics report changes to user space, the Zabbix agent has to specifically implement support for the new method, thus older agent versions would support fewer parameters on recent Linux systems. If you are curious whether a specific parameter works on a specific version of a specific operating system, the best way to find out is to test it. While the Zabbix documentation does contain a supported key matrix, it should be considered as a guideline, not the definitive answer.

Simple checks

The previously created items all required the Zabbix agent daemon to be installed, running, and able to make a connection in either direction. But what if you can't or don't want to install the agent on a remote host, and only need to monitor simple things? This is where simple checks can help you. These checks do not require any specialized agent running on the remote end, and only rely on basic network protocols such as ICMP and TCP to query monitored hosts.

Let's create a very basic check now. Open **Configuration | Hosts**, click on **Items** next to **Another Host** and click on **Create Item**. Fill in the values as follows:

- **Description:** Enter **SMTP server status**
- **Type:** Select **Simple check**

- **Key:** Click the **Select** button. The **Type** dropdown at the upper-right corner should already say **Simple check**. If it doesn't, change it to that. In the key list, click on the **smtp<,port>** key, then edit it, deleting everything between the angle brackets, and the brackets themselves
- **Update interval:** Set it to **60**.
- **Keep history:** Change this to **7**

Item 'Another Host:SMTP server status'

Host	Another Host	<input type="button" value="Select"/>
Description	SMTP server status	
Type	Simple check	<input type="button" value="Select"/>
Key	smtp	<input type="button" value="Select"/>
Type of information	Numeric (unsigned)	<input type="button" value="Select"/>
Data type	Decimal	<input type="button" value="Select"/>
Units		
Use multiplier	Do not use	<input type="button" value="Select"/>
Update interval (in sec)	60	
Flexible intervals (sec)	No flexible intervals	
New flexible interval	Delay 50 Period 1-7,00:00-23:59	<input type="button" value="Add"/>
Keep history (in days)	7	
Keep trends (in days)	365	
Status	Active	<input type="button" value="Select"/>
Store value	As is	<input type="button" value="Select"/>
Show value throw map	As is	<input type="button" value="Select"/>
New application		
Applications	None	
<input type="button" value="Save"/> <input type="button" value="Cancel"/>		



When configuring simple checks in Zabbix, beware of paranoid network security configurations that might trigger an alert if you are checking too many services too often.

When done, click **Save**. To check the result, open **Monitoring | Latest data**—our new check should be there and depending on whether you have SMTP server running and accessible for Zabbix server, should list either 1 (if running and accessible) or 0.

Setting up ICMP checks

What if we care only about the basic reachability of a host, like a router or switch that is out of our control? ICMP ping would be an appropriate method for monitor in that, and Zabbix supports such simple checks. Usually these won't work right away, to use them we'll have to set up a separate utility, **fping**, which Zabbix uses for ICMP checks. It should be available for most distributions, so just install it using distribution package management tools. If not, you'll have to download and compile fping manually, available at <http://fping.sourceforge.net/>.

Once fping is properly installed, the Zabbix server must know where to find it and be able to execute it. On the Zabbix server, open `/etc/zabbix/zabbix_server.conf` and look for the parameter `FpingLocation`. It is commented out by default and defaults to `/usr/sbin/fping`. You can quickly find fping binary location with:

```
$ whereis -b fping
```

If one of the results is `/usr/sbin/fping`, you don't have to change this parameter. If it's not, modify the parameter to point to the correct fping location and restart the Zabbix server so that it knows about the configuration changes. That's not it yet. Zabbix also needs to be able to run fping as root, so execute as root:

```
# chgrp zabbix /usr/sbin/fping
# chmod 4710 /usr/sbin/fping
```

As the fping binary should have been owned by root before, this should be enough to allow its use for the Zabbix group as required, let's verify that.

As usual, navigate to **Configuration | Hosts**, click on **Items** next to **Another Host** and click on **Create Item**. Fill in the following:

- **Description:** Type **ICMP ping performance**.
- **Type:** Choose **Simple check**.
- **Key:** Click on the **Select** button, in the list click on **icmppingsec** key, then remove everything inside the square brackets and the brackets themselves.
- **Type of information:** Choose **Numeric (float)**.
- **Units:** Enter **ms**.
- **Use multiplier:** Select **Custom multiplier**.

- **Custom multiplier:** This field will appear after you choose **Custom multiplier** in the above dropdown. Enter 1000.
- **Update interval:** Change to 60.
- **Keep history:** Change to 7.

Item 'Another Host:ICMP ping performance'

Host	Another Host	Select
Description	ICMP ping performance	
Type	Simple check	
Key	icmppingsec	Select
Type of information	Numeric (float)	
Units	ms	
Use multiplier	Custom multiplier	
Custom multiplier	1000	
Update interval (in sec)	60	
Flexible intervals (sec)	No flexible intervals	
New flexible interval	Delay 50 Period 1-7,00:00-23:59	Add
Keep history (in days)	7	
Keep trends (in days)	365	
Status	Active	
Store value	As is	
New application		
Applications	-None-	
Save Cancel		

When all fields are correctly set, click on **Save**. Perform the usual roundtrip to **Monitoring | Latest data** – ICMP ping should be recording data already. If you wait for a few minutes, you can also take a look at the relatively interesting graph to notice any changes in network performance.

ICMP ping performance	11 Feb 2010 11:17:29	0.27 ms	-	Graph
-----------------------	----------------------	---------	---	-----------------------

Here, we set up ICMP ping in seconds, measuring network latency. If you wanted to simply test host connectivity, you would have chosen key **icmpping**, which would only record whether the ping was successful or not. That's a simple way to test connectivity on a large scale, as it puts a small load on the network (unless you use ridiculously small intervals). Of course, there are things to be aware of, like doing something different to test internet connectivity – it wouldn't be enough to test the connection to your router, firewall, or even your provider's routers. The best way would be to choose several remote targets to monitor that are known to have a very good connection and availability.

Tying it all together

So we found out that a normal or passive agent waits for the server to connect, while an active agent connects to the server, grabs list of items to check, and then reconnects to the server periodically to send in the data. This means that using one or another kind of Zabbix agent items can impact performance. In general, active agents would reduce the load on the Zabbix the server, because server doesn't have to keep a list of what to check and when to check it. Instead, the agent picks up that task and reports back to the server. But you should evaluate each case separately – if you have only a few items per host that you monitor very rarely (the update interval is set to a large value), converting all agents to active ones that retrieve the item list more often than the items were checked previously wouldn't improve Zabbix server performance.

It is important to remember that you can use a mixture of various items against single host. As we just saw, single host can have normal or passive Zabbix agent items, active Zabbix agent items, and simple checks assigned. That allows you to choose the best fit for monitoring every characteristic to ensure the best connectivity, and performance and the least impact on the network and the monitored host. And that's not all yet – we'll explore several additional item types, which again can be mixed with the ones we already know for a single configured host.

Positional parameters for item descriptions

While we're working with items, let's explore some more tricks. Open **Configuration** | **Hosts**, click on **Items** next to **Another Host**, then click on **Incoming traffic on interface eth0** in the **Description** column. In the item editing form, click the **Clone** button at the bottom. In the new form, modify **Key** field so that it reads `net.if.in[lo]`, then click on **Save**.

You might notice it right away, or click on **Monitoring | Latest data** and look at the list. Despite the fact that we only modified the key, item name was updated accordingly as well.

Incoming traffic on interface lo	11 Feb 2010 11:35:57	28.76 B	+0.47 B	Graph
Incoming traffic on interface eth0	11 Feb 2010 11:35:57	202.83 B	-17.46 B	Graph

That's what the `$1` part in the item description field was doing. It's like a normal shell positional parameter, taking the first parameter of the item key. If we had more parameters, we could access those for inclusions in description with `$2`, `$3`, and so on. This is mostly useful in cases where you want to create several items that monitor different entities, so that when cloning the items you have to change only a single instance of the identifier. It's easier than it seems to miss some change when there are multiple locations, thus creating items with mismatched configuration.

Now that we have some more items configured, it's worth looking at another monitoring view. While we spent most of our time in **Monitoring | Latest data**, this time navigate to **Monitoring | Overview**. The **Type** dropdown at the upper-right corner currently lists **Triggers**, which does not provide a very exciting view for us—we have only a single trigger created. But we did create several items, so switch this dropdown to **Data**.

OVERVIEW		
Hosts location	Top	
Items	A Test Host	Another Host
CPU Load	0.120000	-
ICMP ping performance	-	0.22 ms
Incoming traffic on interface lo	-	28.28 B
Incoming traffic on interface eth0	-	136.52 B
SMTP server status	-	0
SSH server status	-	1
WEB server status	-	1
Zabbix agent version	-	1.8.1

Using mass update

Now that looks quite good, we can see in a compact form all of the monitored data. Those "1" results that denote the status for various servers, what do they mean? Was "1" for a running state, or was it an error, like with exit codes? They surely aren't

intuitive enough, so let's try to remedy that. Open Configuration | Hosts, and click on **Items for Another Host**. Select all three server status items (SMTP, SSH, and WEB), then look at the action dropdown, available at the bottom of the item list.

The screenshot shows the Zabbix 1.8.1 Configuration | Hosts screen. A table lists various items with columns for Log, Description, Triggers, Key, Interval, History, Trends, Type, Status, Applications, and Error. Several items are selected, including 'SMTP server status', 'SSH server status', and 'WEB server status'. At the bottom of the table, there is a context menu with options: 'Activate selected', 'Disable selected', 'Mass update', 'Copy selected to ...', 'Clear history for selected', and 'Delete selected'. The 'Mass update' option is highlighted. The status bar at the bottom indicates 'Zabbix 1.8.1 Copyright 2001-2009 by SIA Zabbix' and 'Connected as 'Admin''. A 'Go (3)' button is also visible.

This time we will want to make single change for all the selected items, so the third action from the top looks like what we need—it says **Mass update**. Select it, then click the **Go** button, then confirm the pop up.

The screenshot shows the 'Mass update' dialog box. It contains a list of configuration options with checkboxes and their original values. The options include: Type (Original), SNMP community (Original), SNMPv3 security name (Original), SNMPv3 security level (Original), SNMPv3 auth passphrase (Original), SNMPv3 priv passphrase (Original), SNMP port (Original), Type of information (Original), Data type (Original), Units (Original), Custom multiplier (0 - Disabled) (Original), Update interval (in sec) (Original), Flexible intervals (sec) (Original), New flexible interval (Original), Keep history (in days) (Original), Keep trends (in days) (Original), Status (Original), Log time format (Original), Store value (Original), Show value [throw map](#) (As is dropdown set to As is), Allowed hosts (Original), and Applications (Original). At the bottom are 'Update' and 'Cancel' buttons.

Now that's an interesting screen—it allows us to change some parameters for multiple items at once. While doing that, only changes that are marked and specified are performed, thus we can change some common values for otherwise wildly differing items. It allows us to set things such as **Update interval** or any other parameter, unified for the selected items.

Value mapping

This time we are interested in only one value—how the value is displayed to us. Mark the checkbox next to the **Show value throw map** entry to see the available options.

Looks like somebody has already defined entries here but let's find out what it actually means before making a decision. Click the **throw map** link to the left on the same line.

VALUE MAPPING	
Name	Value map
Service state	0 => Down 1 => Up
Host status	0 => Up 2 => Unreachable
Windows service state	0 => Running 1 => Paused 3 => Pause pending 4 => Continue pending 5 => Stop pending 6 => Stopped 7 => Unknown 255 => No such service
APC Battery Replacement Status	
APC Battery Status	
HP Insight System Status	
Dell Open Manage System Status	

Looking at the list, we can see various names, each of them having a list of mapped references. Look at the **Name** column, where the predefined entries have hints at what they are good for. You can see UPS related mappings, generic status/state, and Windows service related mappings. The **Value map** column shows the exact mappings that are assigned to each entry. But what exactly are they? Looking at the entries, you can see things like **0 => Down** or **1 => Up**. It's useful to know that such mappings can only be assigned to items having integer data type, thus such data

arriving for an item that has value mapping assigned will expose the descriptive mappings. You are completely free to create any mapping you desire, as long as incoming data is numeric integer. To create a new category of mapped data, you would use a button at the upper-right corner, named **Create value map**. We won't do that now, because one of the available mappings covers our needs quite well. Look at the entries, and remember about items we were curious about? They were monitoring a service and they used **1** to denote a service that is running and **0** to denote a service that is down. Looking at the list, we can see an entry **Service state**, which defines **0** as **Down** and **1** as **Up**—exactly what we need. Well, that means we don't have to create or modify any entries, so simply close this window.



You can access **VALUE MAPPING** configuration screen at any time by navigating to **Administration | General** and choosing **Value mapping** from the dropdown at the upper-right corner.

<input checked="" type="checkbox"/> Show value throw map	Service state ▾
--	---------------------------------

When back in the mass update screen, think about mapping entries we just saw, remember which entry fit our requirements the best. Hopefully you remember that, so simply choose **Service state** from the dropdown for the only entry we marked the checkbox against.

Let's see how our change affected information display. Configured and assigned value mappings are used in most Zabbix frontend locations where it makes sense. For example, let's look at the old friend of ours, **Monitoring | Latest data**. Take a close look at the various server status entries—Zabbix still shows numeric values for the reference, but each has conveniently listed appropriate "friendly name" mapped value:

SMTP server status	11 Feb 2010 12:12:28	Down (0)	-	Graph
SSH server status	11 Feb 2010 12:12:25	Up (1)	-	Graph
WEB server status	11 Feb 2010 12:12:24	Up (1)	-	Graph

I have currently stopped the SMTP server to verify whether both **1 => Up** and **0 => Down** mappings work – as we can see, they do. Value mapping isn't available for any data type, it's possible for integers only, so it will be useful for returned data that works like "codes" – service states, hardware states (like batteries), and other similar monitored data. We saw some predefined examples in the value mapping configuration screen before, and we are free to modify or create new mappings according to our needs.

Navigate back to **Monitoring | Overview** and again, look at the various server status entries for **Another Host**.

Items	A Test Host	Another Host
CPU Load	0.070000	-
ICMP ping performance	-	0.27 ms
Incoming traffic on interface lo	-	28.28 B
Incoming traffic on interface eth0	-	159.35 B
SMTP server status	-	Down (0)
SSH server status	-	Up (1)
WEB server status	-	Up (1)
Zabbix agent version	-	1.8.1

While value mapping doesn't seem too useful when you have to remember a single monitored characteristic with only two possible states, it becomes useful when there are many different possible states and many possible mappings, so that in most locations you will have a quick hint on what each numeric value means, and you are always free to invent your own mappings for custom developed solutions.

Copying items

Looking at the same overview screen, the data seems easier to understand with textual hints provided for previously cryptic numeric values, but there's still a bit not-so-perfect displaying. Notice the dashes, displayed for **CPU Load** item for **Another Host** and all other values for **A Test Host**. We didn't create corresponding items on both hosts, and item data is displayed here, which means missing items should be created for each host to gather the data. But recreating all items would be very boring. Luckily, there's a simple and straightforward solution to this problem.

Open **Configuration | Hosts** and click on **Items** next to **A Test Host**. We had only single item configured for this host, so mark the checkbox next to this item. Let's look at the available action dropdown at the bottom of the list again:

	Log	Description	Triggers	Key	Interval	History	Trends	Type	Status	Applications	Error
<input checked="" type="checkbox"/>		CPU Load	Triggers (1)	system.cpu.load	30	90	365	Zabbix agent	Active	-	<input checked="" type="checkbox"/>

Activate selected

Activate selected

Disable selected

Mass update

Copy selected to ...

Clear history for selected

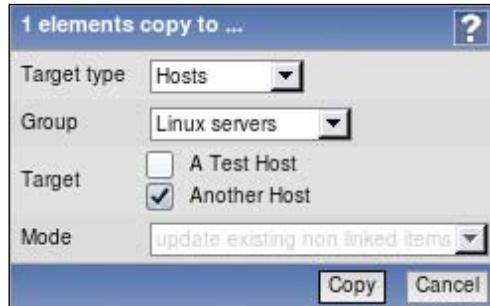
Delete selected

Go (1)

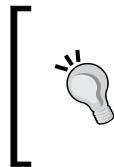
Copyright 2001-2009 by SIA Zabbix

Connected as 'Admin'

This time we don't want to update selected items but copy them to another host, so choose **Copy selected to ...**, click on the **Go** button, and confirm the pop up. By default, this opens a large list, showing all hosts configured, including templates. To simplify the list, select **Linux servers** in the **Group** dropdown, which should leave us with a much shorter list. As we are copying from "A Test Host" to the "Another Host", mark checkbox next to **Another Host** entry and click **Copy**.



When the operation completes, click **Select** next to the **Host** filter field (expand the filter if it is closed), choose **Linux servers** in the **Group** dropdown and click on **Another Host**, then click **Filter**. Notice how **CPU Load** item has appeared in the list. This time, mark all items except **CPU Load**, because that's the only item "A Test Host" has. Here's a little trick to make this easier – hold down *Ctrl* on the keyboard and mark checkbox next to the item **ICMP ping performance** (first item in the range we want to select). Then, still holding down *Ctrl*, click **Zabbix agent version** (last item in the range we want to select). This should select all items between the two checkboxes we clicked in.



Ctrl + click works to both select and unselect arbitrary entry ranges, including items, hosts, triggers, and other entries in the Zabbix frontend. It works both upwards and downwards. The result of the action depends on the first checkbox marked – if you select it, whole range will be selected and vice versa.

With those items selected, choose **Copy selected to ...** from the action dropdown and click on the **Go** button, then confirm the pop up. This time choose **Linux servers** in the **Group** dropdown, mark only the checkbox next to **A Test Host** and click **Copy**. After that, click the **Details** link; notice how this time we changed more entities with single operation, thus expanded details box has a vertical scrollbar instead of pushing other content down.

Details	Items copied
	Added new item A Test Host:icmppingsec
	Added new item A Test Host:net.if.in[lo]
	Added new item A Test Host:net.if.in[eth1]
	Added new item A Test Host:smtp
	Added new item A Test Host:net.tcp.service[ssh]
	...

Let's take another look at **Monitoring | Overview**.

Items	A Test Host	Another Host
CPU Load	0.180000	0.060000
ICMP ping performance	0.03 ms	0.26 ms
Incoming traffic on interface lo	106.98 B	28.76 B
Incoming traffic on interface eth0	2.38 KB	168.12 B
SMTP server status	Down (0)	Down (0)
SSH server status	Up (1)	Up (1)
WEB server status	Up (1)	Up (1)
Zabbix agent version	1.8.1	1.8.1

Great, that's much better. We can see all the data for two hosts with the numeric status nicely explained. Basically, we just cross-copied items that did not exist on one host from the other one.

But it only gets better – mouseover displayed values. Notice how chosen row is highlighted, and selected cell gains a dotted border. Let's click on one of the **CPU Load** values.

Items	A Test Host	Another Host
CPU Load Graphs	1000 0.020000	
ICMP packet loss Last hour graph	ms 0.30 ms	
Incoming traffic Last week graph	B 28.28 B	
Incoming traffic Last month graph	B 168.77 B	
SMTP server status Values	Down (0)	Up (0)
SSH service status Values) Up (1)	Up (1)
WEB server status Values) Up (1)	Up (1)
Zabbix agent version	1.8.1	1.8.1

As you can see, the overview screen not only shows you data in a tabular form, it also allows quick access to common time scale graphs and the **500 latest values** for the item. Clicking on any of the links will open a new browser window with the chosen data displayed. Feel free to try that out.

When you have looked at the data, click on one of the **Zabbix agent version** values.

SMTP server status	Down (0)	Down (0)
SSH server status	Up (1)	Up (1)
WEB server status	Up (1)	Up (1)
Zabbix agent version Values	1.8.1	1.8.1
500 latest values		

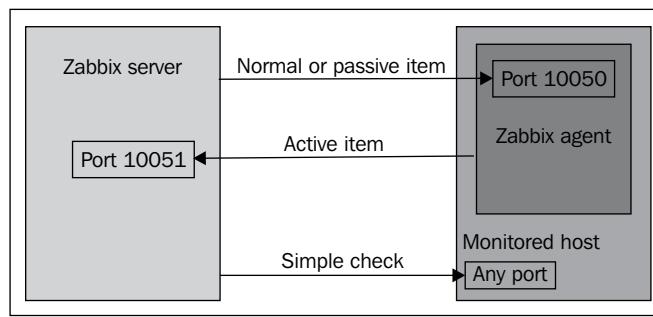
Notice how this time there are no entries for graphs. Remember, graphs were only available for numeric data, so **Monitoring | Latest data** and these overview screen pop up menus offer the value history only.

Summary

This time we created a new host, added several normal or passive items, active items, and simple checks to two different hosts. We also explored many tricks and mechanisms to ease managing in the frontend, like item cloning, copying, and value mapping.

It might be worth remembering how connections are made for each of these item types—that's important when you have to decide on monitoring mechanisms based on existing network topology and configuration.

Let's look at a simplistic diagram. The arrow direction denotes how connections are made.



Listed are default ports that can be changed, if necessary.

- Normal or passive item: A Zabbix server connects to a Zabbix agent, which in turn gathers the data
- Active item: A Zabbix agent connects to a Zabbix server, retrieves list of things to monitor, gathers the data, and then periodically reports back to the server
- Simple check: A Zabbix server directly queries exposed network interfaces of the monitored host, no agent required

This covers the two basic, most commonly used check types—a Zabbix agent with bidirectional connection support and simple checks that are performed directly from the server.

4

Monitoring SNMP and IPMI Devices

Now that we are familiar with monitoring using Zabbix agents and one agentless method, let's explore two additional methods that do not require Zabbix agent installation, even though they need an agent of some kind anyway—SNMP and IPMI. We'll learn to configure and use SNMP and IPMI items within Zabbix; including SNMP checks and trap receiving, as well as introducing IPMI support in Zabbix.

Simple Network Management Protocol

Our first protocol in this chapter, **Simple Network Management Protocol** or simply **SNMP**, is a well established and popular network monitoring solution.

Being more than two decades old, SNMP has had the time to become widespread across a whole range of networked devices. Although the name implies management functionality, it's mostly used for monitoring. As the first versions had security drawbacks, the ability to modify configuration over SNMP did not become as popular as its read-only counterpart.

SNMP as the primary monitoring solution is especially popular in embedded devices, where running a complete operating system and installing separate monitoring agents would be overkill. Two of the most popular device categories implementing SNMP out of the box are printers and network switches, which allows easy monitoring of these otherwise quite closed devices. Other devices with SNMP agents provided include routers, UPSes, NAS devices, computer rack temperature/humidity sensors and so on. Of course, SNMP is in no way restricted to devices of limited processing power—it's perfectly fine to run a generic SNMP agent instead of specialized monitoring agent on common servers. Reasons to use SNMP agents instead of Zabbix agents might include already installed and set up SNMP agents, no access to monitored hosts to install Zabbix agents, and desire to keep systems relatively free from dependencies on monitoring software.

Given the prevalence of SNMP, it's no wonder Zabbix supports it out of the box—it would be weird if it didn't. SNMP support in Zabbix builds upon another open source quality product—Net-SNMP (<http://net-snmp.sourceforge.net/>).



While Zabbix also supports UCD-SNMP, Net-SNMP is the successor of UCD-SNMP, thus UCD-SNMP is unlikely to be found in any recent distribution. You are strongly suggested to use Net-SNMP.

Using Net-SNMP

As we installed the dependencies and compiled Zabbix server with SNMP support, all that's left to do is setting up SNMP monitoring configuration. Before we do that, we'll need some device that has an SNMP agent installed. This is where you can choose between various options—you can use any networked device that you have access to, like manageable switch, network printer, UPS with SNMP interface, and so on. As SNMP agents usually listen on port 161, you will need the ability to connect to such a device on this port over UDP. Although TCP is also supported, UDP is much more widely used.

If you don't have access to such a device, you can also start up an SNMP daemon on a computer. For example, you could easily use "Another Host" as a testbed for SNMP querying. Many distributions ship with the SNMP daemon from the Net-SNMP package, and often it is enough to simply start `snmpd` service. If that's not the case for your chosen distribution, you'll either have to find one of those networked devices with an SNMP agent already available, or configure `snmpd` manually.

Whichever way you choose, you will have to find out what data the device actually provides and how to get it. This is where Net-SNMP comes in, providing many useful tools to work with SNMP-enabled devices. We will use several of these tools to discover information that is needed to configure SNMP items in Zabbix.

Let's start by verifying whether our SNMP device is reachable and whether it responds to our queries.

While SNMPv3 is the current version of SNMP since 2004, it is still not as widespread as versions 1 and 2. There's a whole lot of old devices in use that only support older protocol versions, and many vendors do not hurry with SNMPv3 implementations.

To complicate things further, SNMPv2 also isn't widely used. Instead, a variation of it, the community-based SNMPv2 or SNMPv2c, is used. While devices can support both v1 and v2c, some only support one of these. Both use so called "community" authentication, though user authentication is performed based on single community string. Therefore, to query a device you would have to know which protocol version it supports, and the community string to use. Though it's not as hard as it sounds; most devices use common string for access – "public", as does Net-SNMP daemon. Unless you explicitly change this string, you can just assume that's what is needed to query any host.

If you have installed and started Net-SNMP daemon on "Another Host", you can do a simple query to verify SNMP connectivity:

```
$ snmpstatus -v 2c -c public <IP address>
```

If the daemon is started correctly and network connectivity is fine, you should get some output, depending on what system you have:

```
[UDP: <IP address>:161] => [Linux zab 2.6.16.60-0.21-default #1 Tue May  
6 12:41:02 UTC 2008 i686] Up: 20:17:40.28  
Interfaces: 3, Recv/Trans packets: 108105509/35678987 | IP:  
48381196/35649135  
1 interface is down!
```

We can see here that it worked, and by default a connection was made over UDP to port 161. We can see the target system's operating system, hostname, kernel version, when was it compiled and what hardware architecture it was compiled for, as well as current uptime. There's also some network statistics information tucked on.

If you are trying to query a network device, it might have restrictions on who is allowed to use SNMP agent. Some devices allow free access to SNMP data, some restrict it by default and every connecting host has to be allowed explicitly. If a device does not respond, check its configuration – you might have to add IP address of the querying machine to the SNMP permission list.

Looking at the `snmpstatus` command itself, we passed two parameters to it – SNMP version (2c in this case) and community (which is as discussed before, "public").

If you have other SNMP-enabled hosts, you can try the same command on them. Let's look at some different devices:

```
$ snmpstatus -v 2c -c public <IP address>  
[UDP: [<IP address>]:161] => [IBM Infoprint 1532 version NS.NP.N118  
kernel 2.6.6 All-N-1] Up: 5 days, 0:29:53.22  
Interfaces: 0, Recv/Trans packets: 63/63 | IP: 1080193/103316
```

As we can see, this has to be an IBM printer. And hey, it seems to be using a Linux kernel.

While many systems will respond to version 2c queries, sometimes you might see the following:

```
$ snmpstatus -v 2c -c public <IP address>
Timeout: No Response from <IP address>
```

This could of course mean network problems, but sometimes SNMP agents ignore requests coming in with protocol version they do not support or an incorrect community string. If the community string is incorrect, you would have to find out what it has been set to; this is usually easily available in the device or SNMP daemon configuration (for example, Net-SNMP usually has it set in the configuration file `/etc/snmp/snmp.conf`). If you believe a device might not support a particular protocol version, you can try another command:

```
$ snmpstatus -v 1 -c public <IP address>
[UDP: [<IP address>]:161=>[HP ETHERNET MULTI-ENVIRONMENT, SN:
CNBW71B06G, FN:JK227AB, SVCID:00000, PID:HP LaserJet P2015 Series] Up:
3:33:44.22
Interfaces: 2, Recv/Trans packets: 135108/70066 | IP: 78239/70054
```

So this HP LaserJet printer did not support SNMPv2c, only v1. Still, when queried using SNMPv1, it divulged information like the serial number and series name.

Let's look at another SNMPv1 only device:

```
$ snmpstatus -v 1 -c public <IP address>
[UDP: [<IP address>]:161=>[APC Web/SNMP Management Card (MB:v3.6.8
PF:v2.6.4 PN:apc_hw02_aos_264.bin AF1:v2.6.1 AN1:apc_hw02_sumx_261.bin
MN:AP9617 HR:A10 SN: ZA0542025896 MD:10/17/2005) (Embedded PowerNet
SNMP Agent SW v2.2 compatible)] Up: 157 days, 20:42:55.19
Interfaces: 1, Recv/Trans packets: 2770626/2972781 | IP:
2300062/2388450
```

This seems to be an APC UPS, and it's providing a lot of information stuffed in this output, including serial number and even firmware versions. It also has considerably longer uptime than the previous systems – 157 days.

But surely there must be more information obtainable through SNMP and also this looks a bit messy. Let's try another command from Net-SNMP arsenal, `snmpwalk`. This command tries to return all the values that are available from a particular SNMP agent, so the output could be very large – we'd better restrict it to few lines at first.

```
$ snmpwalk -v 2c -c public 10.1.1.100 | head -n 6
SNMPv2-MIB::sysDescr.0 = STRING: Linux zab 2.6.16.60-0.21-default #1
Tue May 6 12:41:02 UTC 2008 i686
SNMPv2-MIB::sysObjectID.0 = OID: NET-SNMP-MIB::netSnmpAgentOIDs.10
```

```
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (8411956) 23:21:59.56
SNMPv2-MIB::sysContact.0 = STRING: Sysadmin (root@localhost)
SNMPv2-MIB::sysName.0 = STRING: zab
SNMPv2-MIB::sysLocation.0 = STRING: Server Room
```

As we can see, this command outputs various values; with a name or identifier displayed on the left, and value itself on the right. Indeed, the identifier is called **object identifier** or **OID**, and it is a unique string, identifying single value.

Looking at the output, we can also identify some of the data we saw in the output of `snmpstatus -SNMPv2-MIB::sysDescr.0` and `DISMAN-EVENT-MIB::sysUpTimeInstance`. Two other values, `SNMPv2-MIB::sysContact.0` and `SNMPv2-MIB::sysLocation.0` haven't been changed from the defaults, and thus aren't too useful right now. While we are at it, let's compare this output to the one from APC UPS:

```
$ snmpwalk -v 1 -c <IP address> | head -n 6
SNMPv2-MIB::sysDescr.0 = STRING: APC Web/SNMP Management Card (MB:
v3.6.8 PF:v2.6.4 PN:apc_hw02_aos_264.bin AF1:v2.6.1 AN1:apc_hw02_sumx_
261.bin MN:AP9617 HR:A10 SN: ZA0542025896 MD:10/17/2005) (Embedded
PowerNet SNMP Agent SW v2.2 compatible)
SNMPv2-MIB::sysObjectID.0 = OID: PowerNet-MIB::smartUPS450
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (1364829916) 157
days, 23:11:39.16
SNMPv2-MIB::sysContact.0 = STRING: Unknown
SNMPv2-MIB::sysName.0 = STRING: Unknown
SNMPv2-MIB::sysLocation.0 = STRING: Unknown
```

The output is quite similar, containing the same OIDs, and the system contact and location values aren't set as well. But to monitor some things, we have to retrieve single value per item and we can verify that it works with another command, `snmpget`.

```
$ snmpget -v 2c -c public 10.1.1.100 DISMAN-EVENT-MIB::sysUpTimeInstance
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (8913849) 1 day,
0:45:38.49
```

We can add any valid OID, like `DISMAN-EVENT-MIB::sysUpTimeInstance` in the above example, after the host to get whatever value it holds. Let's look closer at the OID itself. In its current form, it consists of two parts, separated by two colons. The first part is the name of **management information base** or **MIB**. MIBs are collections of item descriptions, mapping numeric OIDs to textual ones. The second part is uniquely identifiable descriptor from this MIB, so such a pair can be used to refer to single, specific value. We can see full OID by adding a the flag `-Of` to modify the output:

```
$ snmpget -v 2c -c public -Of 10.1.1.100 DISMAN-EVENT-MIB::
sysUpTimeInstance
.iso.org.dod.internet.mgmt.mib-2.system.sysUpTime.sysUpTimeInstance =
Timeticks: (8972788) 1 day, 0:55:27.88
```

That's a considerably long name, showing the tree-like structure. It starts with a no-name root object, and goes further with all the values attached at some location to this tree. Well, we mentioned numeric form, and we can make `snmpget` output numeric names as well with the flag `-On`:

```
$ snmpget -v 2c -c public -On 10.1.1.100 DISMAN-EVENT-MIB::  
sysUpTimeInstance  
.1.3.6.1.2.1.1.3.0 = Timeticks: (9048942) 1 day, 1:08:09.42
```

So each OID can be referred to in one of three notations—short, long, or numeric. In this case, `DISMAN-EVENT-MIB::sysUpTimeInstance`, `.iso.org.dod.internet.mgmt.mib-2.system.sysUpTime.sysUpTimeInstance`, and `.1.3.6.1.2.1.1.3.0` all refer to the same value.



You can look at `man snmpcmd` for other supported output formatting options.



But how does this fit into Zabbix SNMP items? Well, to create an SNMP item in Zabbix, you have to enter an OID. How do you know what OID to use? Often you might have the following choices:

- Just know it
- Ask somebody
- Find it out yourself

More often than not first two options don't work, so finding it out yourself will be the only way. As we have learned, Net-SNMP tools are fairly good at supporting such a discovery process.

Using SNMPv3 with Net-SNMP

The latest version of SNMP, version 3, is not that common yet, and it is somewhat more complex than the previous versions. Device implementations also can vary in quality, so it might be useful to test your configuration of Zabbix against a known solution—Net-SNMP daemon. Let's add a SNMPv3 user to it and get some value. Make sure Net-SNMP is installed and that `snmpd` starts up successfully.

To configure SNMPv3, first stop `snmpd`, then as root, run:

```
# net-snmp-create-v3-user -ro zabbix
```

This utility will prompt for a password. Enter a password of at least eight characters. Start snmpd again, and test retrieval of values using version 3:

```
$ snmpget -u zabbix -A zabbixzabbix -v 3 localhost SNMPv2-MIB::sysDescr.0
```

This should return data successfully as follows:

```
SNMPv2-MIB::sysDescr.0 = STRING: Linux rich 2.6.29.6 #2 Mon Aug 17  
01:31:42 CDT 2009 i686
```

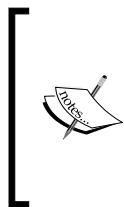
We don't need to configure versions 1 or 2c separately, so now we have a general SNMP agent, providing all common versions for testing or exploring.

Adding new MIBs

One way to discover usable OIDs is to redirect full SNMP tree output to a file and find out what interesting and useful information the device exposes and determine what the OIDs are from that. It's all good as long as the MIB files shipped with Net-SNMP provide the required descriptors, but SNMP MIBs are extensible – anybody can add new information, and many vendors do. In such a case your file might be filled with lines like:

```
SNMPv2-SMI::enterprises.318.1.1.1.2.3.0 = STRING: "QS0547120198"
```

That's quite cryptic. While the output is in the short, textual form, part of it is numeric. This means that there is no MIB definition for this part of the SNMP tree. Luckily, APC offers a MIB for download from their site, so it can be added to Net-SNMP configured MIBs. But how?



Getting SNMP MIBs isn't always easy. A certain large printer manufacturer representative claimed that they do not provide SNMP MIBs and everybody should use their proprietary printer management application. Most manufacturers do provide MIBs, though, and for some when getting them is hard, freely accessible MIB collection sites can help.

After downloading a new MIB, you have to place it in a location where Net-SNMP will search for MIB files and configure it as well. Net-SNMP searches for MIBs in two locations – `.snmp/mibs` in the user's home directory, or `/usr/share/snmp/mibs`, which one to use is your decision. If you want something for the current user only or don't have access to the `/usr` directory, you can use `.snmp/mibs`, otherwise use `/usr/share/snmp/mibs`. Whichever you choose, that's not enough – you also have to instruct tools to include this MIB.

The first method is to pass MIB names directly to the called command. But hey, we don't know the MIB name yet. To find out what a particular name in some file is, open the file in a text editor and look for `MIB DEFINITIONS ::= BEGIN` near the beginning of the file. String before this text will be the MIB name we are looking for. For example:

```
PowerNet-MIB DEFINITIONS ::= BEGIN
```

So APC have chosen to name their MIB `PowerNet-MIB`. Armed with this knowledge, we can instruct any command to include this file:

```
$ snmpget -m +PowerNet-MIB -v 1 -c public <IP address> SNMPv2-SMI::enterprises.318.1.1.1.1.2.3.0
```

```
PowerNet-MIB::upsAdvIdentSerialNumber.0 = STRING: "QS0547120198"
```

Excellent, `snmpget` included the correct MIB and obtained full textual string, which confirms our suspicion that this might be a serial number. You can now use the same flag for `snmpwalk` and obtain a file with much better value names. Quite often you will be able to search such a file for interesting strings like "serial number" and find the correct OID.



The + sign means to include specified MIBs in addition to otherwise configured ones. If you omit the +, the MIB list will be replaced with the one you specified.

Feel free to look at MIB files in the `/usr/share/snmp/mibs` directory. As you can see, most files here have their filename the same as their MIB name without the extension, which is not required. Actually, the filename has nothing to do with the MIB name, thus sometimes you might have to resort to tools like `grep` to find out which file contains which MIB.

While passing individual MIB names on the command line is nice for a quick one time query, it gets very tedious once you have to perform these actions more often and the MIB list grows. There's another method, somewhat more durable – environment variable `MIBS`. In this case, variable could be set like:

```
$ export MIBS=+PowerNet-MIB
```

In the current shell, individual commands do not need the MIB names passed to them anymore. All the MIBs specified in the variable will be included upon every invocation.

Of course, that's also not that permanent. While you can specify this variable in profile scripts, that can get tedious to manage for all users on a machine. This is where a third method comes in – configuration files.

Again, you can use per-user configuration files, located in `.snmp/snmp.conf` in their home directories, or you can use global `/etc/snmp/snmp.conf` file.



The location of global configuration file and MIB directory can be different if you have compiled Net-SNMP from source. They might reside in `/usr/local`.

The syntax to add MIBs is similar to the one used in the environment variable—you only have to prefix each line with "mibs", for example:

```
mibs +PowerNet-MIB
```

If you want to specify multiple MIB names in any of these locations, you have to separate them with a colon. Let's say you also need generic UPS MIB, in that case MIB name string would be:

```
+PowerNet-MIB:UPS-MIB
```

And if you feel lazy, you can make Net-SNMP include all MIB files, located in those directories by setting `mibs` to `ALL`—this works in all three locations. Beware that this might impact performance and also lead to some problems if some parts are declared in multiple locations; including warnings from Net-SNMP tools and incorrect definitions being used.

Working with SNMP items in Zabbix

Armed with knowledge about SNMP OIDs, let's get to the real deal—getting SNMP data into Zabbix. To ease the following steps, you should choose an entry that returns string data. We could use a UPS serial number, like the one discovered above to be `PowerNet-MIB::upsAdvIdentSerialNumber.0`, do the same for some network printer or manageable switch, or if you don't have access to such a device, you can choose a simple entry from the Net-SNMP enabled host, like already mentioned system description—`SNMPv2-MIB::sysDescr.0`.

Now is the time to return to the Zabbix interface.

Go to **Configuration | Hosts**, click **Create Host**. Then fill in the following values:

- **Name:** Enter **SNMP Device**.
- **Groups:** If in the **In Groups** listbox there's any group, select it and click on the **>>** button.
- **New group:** Enter **SNMP Devices**.

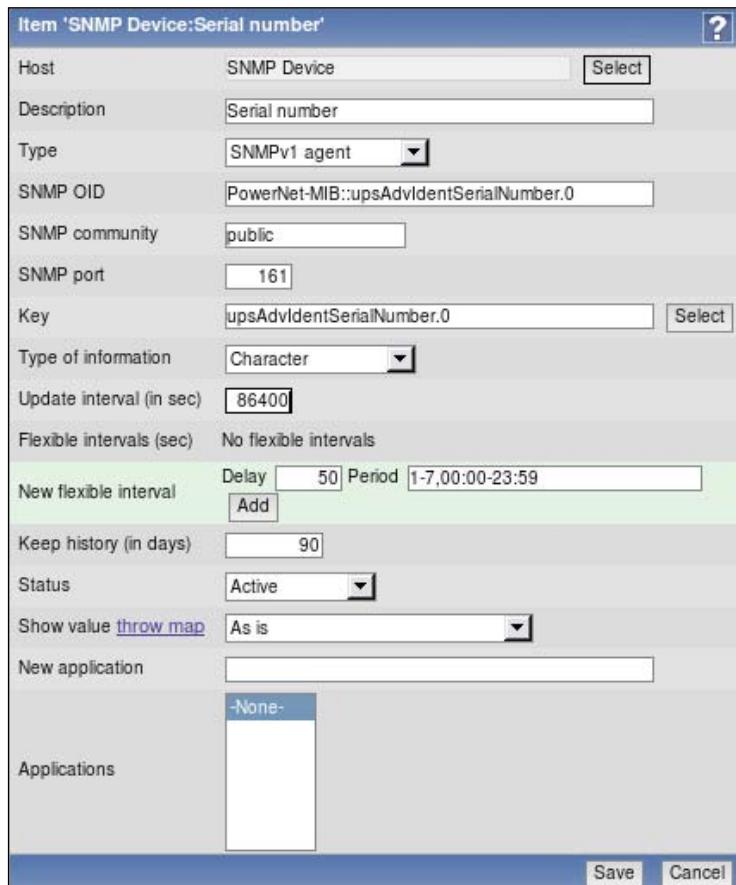
- **DNS name or IP address:** Enter the correct DNS name or IP address. If you have chosen to use an SNMP-enabled device, input its IP or DNS name here. If you don't have access to such a device, use Another Host's IP address or DNS name.
- **Connect to:** Choose DNS or IP, according to which field you populated.

The screenshot shows the 'Host' configuration dialog in Zabbix. The 'Name' field is set to 'SNMP Device'. In the 'Groups' section, there is a list of available groups: 'Discovered Hosts', 'Linux servers' (which is currently selected), 'Templates', 'Windows servers', and 'Zabbix Servers'. The 'In Groups' list is currently empty. Below the groups, there are several configuration fields: 'New group' (set to 'SNMP Devices'), 'DNS name' (empty), 'IP address' (set to '10.1.1.200'), 'Connect to' (set to 'IP address'), 'Zabbix agent port' (set to '10050'), 'Monitored by proxy' (set to '(no proxy)'), 'Status' (set to 'Monitored'), and a 'Use IPMI' checkbox (unchecked). At the bottom right are 'Save' and 'Cancel' buttons.

When you are done, click **Save**. It's likely that you won't see newly created host in the host list – the reason being the **Group** dropdown in the upper-right corner, which probably says **Linux servers**. You can change the selection to **all** to see all configured hosts, or to **SNMP Devices** to only see our new device. Now is the time to create an item, so click on **Items** next to **SNMP Device** and click the **Create Item** button. Fill in the following values:

- **Description:** Enter something sensible like **Serial number** if you are using an OID from SNMP agent, or "System description" if you are using the Net-SNMP daemon.
- **Type:** Change to the appropriate version of your SNMP agent. In the displayed example, **SNMPv1 agent** is chosen because that's the only version our device supports.

- **SNMP OID:** This is where our knowledge comes in. Paste the SNMP OID you have found out and chosen here. In the example, `PowerNet-MIB::upsAdvIdentSerialNumber.0` is entered. If you are using the Net-SNMP daemon, enter `SNMPv2-MIB::sysDescr.0`.
- **Key:** Not restricted or too important for SNMP items, but needed for references from triggers and other locations. You can choose to enter the last part of the textual OID, like `upsAdvIdentSerialNumber.0` or `sysDescr.0`.
- **SNMP community:** Unless you have changed it, keep the default **public**.
- **Type of information:** Select **Character**.
- **Update interval:** This information really isn't changing that often, so use some large value like **86400** again.



When you are done, click **Save**.

Now, the outcome will depend on several factors. If you are lucky, you will already see the incoming data in **Monitoring | Latest data**. If you have chosen some vendor-specific OID, like in our example, it is possible that you will have to go back to **Configuration | Hosts**, click **Items** next to **SNMP Device**, and observe the status of this item.

Status	Applications	Error
Active	-	
SNMP error [1]		

Now what's that? How could it be? We saw in our tests with Net-SNMP command line tools that there actually is such an OID. Well, one possible situation when this error message appears is when specified MIB is not available—which could happen if you tried SNMP queries previously from a different host.

Zabbix server works as if **ALL** is set for MIB contents, thus you don't have to do anything besides specific MIB copying in correct directory (usually `/usr/share/snmp/mibs`) on the Zabbix server.

If you did not copy the OID, instead deciding to retype it, you might have made a mistake if you see an error like this:

	Interval	History	Trends	Type	Status	Applications	Error
.0	86400	90	0	SNMPv1 agent	Active	-	
SNMP error [(noSuchName) There is no such variable name in this MIB.]							

Verify that the entered OID is correct. After fixing any problems, wait until Zabbix server refreshes the item configuration and rechecks the item. With the item configured, let's see what data we can get in Zabbix from it. Navigate to **Monitoring | Latest data**, select **SNMP Devices** from the **Group** drop-down and **SNMP Device** from the **Host** dropdown. Expand the **-other-** category if needed and look for the serial number.

Serial number	-	X A123PO	-	History
---------------	---	----------	---	-------------------------

The serial number has been successfully retrieved and is visible in the item listing. This allows us to automatically retrieve data that, while not directly tied to actual availability or performance monitoring, is still quite useful. For example, if a remote device dies and has to be replaced, you can easily find the serial number to supply in a servicing request even if you neglected to write it down beforehand.

Translating SNMP OIDs

In case you can't or don't want to copy vendor specific MIB files on the Zabbix server, you can always use numeric OIDs, like we saw before. While not being as descriptive, they are guaranteed to work even if copied MIBs are not available for some reason or are removed during a system upgrade.

But how do we derive corresponding numeric OIDs from a textual one? While we could use `snmpget` to retrieve the particular value and output it in numeric form, that requires availability of the device and network roundtrip. Fortunately, there's an easier way – `snmptranslate` command. To find out numeric form for key we use, `PowerNet-MIB::upsAdvIdentSerialNumber.0`:

```
$ snmptranslate -On PowerNet-MIB::upsAdvIdentSerialNumber.0
.1.3.6.1.4.1.318.1.1.1.2.3.0
```



You must have MIBs placed correctly and pass their names to Net-SNMP tools for translation to work.

The default output format for Net-SNMP tools is a short textual one, which only outputs the MIB name and object name. If you would like to find out the corresponding textual name use the following:

```
$ snmptranslate .1.3.6.1.2.1.1.1.0
SNMPv2-MIB::sysDescr.0
```

You can also use the `-Of` flag to output an OID in full notation:

```
$ snmptranslate -Of PowerNet-MIB::upsAdvIdentSerialNumber.0
.iso.org.dod.internet.private.enterprises.apc.products.hardware.ups.
upsIdent.upsAdvIdent.upsAdvIdentSerialNumber.0
```

Dynamic indexes

Previously, we monitored incoming traffic on `eth0` device using an active Zabbix agent daemon item. If we have `snmpd` set up and running, we can also try retrieving outgoing traffic but this time, let's try to use SNMP for that.

Monitoring network traffic using Zabbix agent daemon is usually easier, but SNMP monitoring is the only way to obtain this information for many network devices such as switches and routers. If you have such a device available, you can try monitoring it instead, though the network interface name most likely will differ.

One way to find the item that we are interested in would be to redirect output of `snmpwalk` to a file and then examine this file. Looking at the output, there are lines like:

```
IF-MIB::ifDescr.1 = STRING: lo
IF-MIB::ifDescr.2 = STRING: eth0
```

Great, so the desired interface, `eth0` in this case, has an index of "2". Nearby we can find actual information we are interested in – traffic values:

```
IF-MIB::ifOutOctets.1 = Counter32: 1825596052
IF-MIB::ifOutOctets.2 = Counter32: 1533857263
```

So theoretically we could add an item with the OID `IF-MIB::ifOutOctets.2` and name it appropriately. Unfortunately, there are devices which change interface index now and then. Also, index for a particular interface is likely to differ between devices, thus potentially creating a configuration nightmare. This is where dynamic index support in Zabbix comes in use.

Let's look at what a dynamic index item OID would look like in this case:

IF-MIB::ifOutOctets["index",	"ifDescr",	"eth0"]
Data base OID	Literal string "index"	Index base OID	Index string

- Data base OID: The base part of the OID that holds the data we are interested in, that is, without the actual index. In this case, the OID leading to `ifOutOctets`, in any notation.
- Literal string index: This is the same for all dynamic index items.
- Index base OID: The base part of the OID that holds the index we are interested in. In this case, the OID leading to `ifDescr`, in any notation.
- Index string: The string that the index part of the tree is searched for. This is an exact, case-sensitive match of all OIDs from the previous base OID. Here, the name of the interface we are interested in, `eth0`, will be searched for.

The index that this search will return will be added to the database OID and the next query will gather values from the resulting OID.

You can easily view the index to determine the correct string for search to enter here with Net-SNMP tools.

```
$ snmpwalk -v 2c -c public localhost .iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable.ifEntry.ifDescr  
IF-MIB::ifDescr.1 = STRING: lo  
IF-MIB::ifDescr.2 = STRING: eth0  
IF-MIB::ifDescr.3 = STRING: sit0
```

As can be seen, this machine has three interfaces—loopback, ethernet, and tunnel. The picture will be very different for some other devices; for example, a HP ProCurve Switch would return (with the output shortened) the following:

```
$ snmpwalk -v 2c -c public 10.196.2.233 .iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable.ifEntry.ifDescr  
IF-MIB::ifDescr.1 = STRING: 1  
IF-MIB::ifDescr.2 = STRING: 2  
...  
IF-MIB::ifDescr.49 = STRING: 49  
IF-MIB::ifDescr.50 = STRING: 50  
IF-MIB::ifDescr.63 = STRING: DEFAULT_VLAN  
IF-MIB::ifDescr.4158 = STRING: HP ProCurve Switch software loopback interface
```

Now that we know the OID to use for dynamic index items, let's create that item in Zabbix. Navigate to **Configuration | Hosts**, click on **Items** next to the correct host you want to create item for and click **Create Item**.

- **Description:** Enter **Outgoing traffic on interface \$1**
- **Type:** Choose **SNMPv2 agent**
- **SNMP OID:** Enter **IF-MIB::ifOutOctets["index","ifDescr","eth0"]**
- **Key:** Enter **ifOutOctets[eth0]**
- **Type of information:** Select **Numeric (float)**
- **Units:** Enter **Bps**
- **Update interval:** Change it to **60**

- **Keep history:** Change to 7
- **Store value:** Choose Delta (speed per second).

Item 'A Test Host:Outgoing traffic on interface \$1'

Host	A Test Host	Select
Description	Outgoing traffic on interface \$1	
Type	SNMPv2 agent	
SNMP OID	[F-MIB::ifOutOctets["index","ifDescr","eth0"]]	
SNMP community	public	
SNMP port	161	
Key	ifOutOctets[eth0]	Select
Type of information	Numeric (float)	
Units	Bps	
Use multiplier	Do not use	
Update interval (in sec)	60	
Flexible intervals (sec)	No flexible intervals	
New flexible interval	Delay 50 Period 1-7:00:00-23:59	Add
Keep history (in days)	7	
Keep trends (in days)	365	
Status	Active	
Store value	Delta (speed per second)	
New application		
Applications	-None-	

Save Cancel

Make sure that the compound OID is entered correctly, paying close attention to quotes and spelling. When you are done, click **Save**.

The newly added item should start gathering data, so let's look at **Monitoring** | **Latest data**. If you don't see this item or the data for it, navigate back to **Configuration** | **Hosts**, click **Items** next to the corresponding host – there should be error message displayed that should help with fixing the issue. If you have correctly added the item, you'll see nicely displayed traffic data:

Outgoing traffic on interface eth0	11 Feb 2010 17:35:40	2.86 KBps	+2.15 KBps	Graph
------------------------------------	----------------------	-----------	------------	-----------------------

Dynamic index items are quite common. Many network devices have fixed port names but varying index. Host-based SNMP agents place things like disk usage and memory statistics in dynamic indexes, thus if you have such devices to monitor, Zabbix support for them will be handy.

Using dynamic index items can slightly increase overall load, as two SNMP values are required to get the final data. Zabbix caches retrieved index information and uses a single connection to the SNMP agent, so the load increase should not be noticeable.

Receiving SNMP traps

While querying SNMP-capable devices is a nice method that requires little or no configuration of each device on itself, in some situations reverse direction information flow is desired. For SNMP, those are called traps. Usually traps are sent upon some condition change and the agent connects to the server or management station to port 162 (as opposed to port 161 on the agent side that is used for queries). You can think of SNMP traps as being similar to Zabbix active items—as with those, all connections are made from monitored machines to the monitoring server.

The direction of the connections isn't the only difference—SNMP traps have some other pros and cons when compared to queries. For example, SNMP traps are usually more capable of detecting short problems that might have been missed by queries. Let's say you are monitoring incoming voltages on a UPS. You have decided on a reasonable item interval that would give you useful data and wouldn't overload network and Zabbix server, let's say some 120 seconds or two minutes. If the input voltage suddenly peaks or drops for a minute, your checks might easily miss this event, thus making it impossible to correlate with problems to other devices that are not connected to the UPS. Another benefit that traps provide is reduced network and Zabbix server load as the information is only sent upon an event, there is no constant querying by the server. A drawback is partial decentralization of the configuration. SNMP trap sending conditions and parameters have to be set for each device or device group individually.

As such, SNMP traps aren't used to replace SNMP queries. Instead they supplement them, by leaving statistical information gathering to the queries, and providing notifications of various events happening in the devices, usually notifying of emergencies.

In Zabbix, SNMP traps are received by `snmptrapd`, a daemon from Net-SNMP suite again. These traps then have to be passed to Zabbix daemon with some method. Zabbix includes an example shell script to do just that, so we'll use that for simplicity.

If you don't have one already, create home directory for the zabbix user as root, along with a location to place scripts for it.

```
# mkdir -p /home/zabbix/bin; chown zabbix /home/zabbix
```

Let's place the SNMP trap parsing script in this directory now.

```
# cp misc/snmptrap/snmptrap.sh /home/zabbix/bin
```

Let's take a look at that script now. Open the file we just copied to `/home/zabbix/bin/snmptrap.sh`. As you can see, this is a very simplistic script, which gets passed trap information, then sends it to the Zabbix server, using both host `snmptraps` and key `snmptraps`. If you are reading it carefully enough, you probably already noticed one problem. We didn't install any software as `~zabbix/bin/zabbix_sender`, so that probably is wrong.

First, let's find out where `zabbix_sender` is actually located.

```
$ whereis zabbix_sender
zabbix_sender: /usr/local/bin/zabbix_sender
```

On this system it's `/usr/local/bin/zabbix_sender`. It might be a good idea to look at its syntax by running:

```
$ zabbix_sender --help
```

This allows you to send a value to the Zabbix server, specifying the server with `-z` flag, port with `-p` flag, and so on. Now let's return to the script. With our new knowledge, look at the last line, the one that invokes `zabbix_sender`. The script seems to pass values retrieved from SNMP trap as parameters to `zabbix_sender`, thus we can make any decisions and information transformation between `snmptrapd` and Zabbix. Now let's fix the problem we noticed:

- Change `ZABBIX_SENDER` to read `/usr/local/bin/zabbix_sender` (or another path if that's different for you);
- Additionally, change the last line to read `$ZABBIX_SENDER -z $ZABBIX_SERVER -p $ZABBIX_PORT -s "$HOST" -k "$KEY" -o "$str"`—this way we are also quoting host and key names, just in case they might include spaces or other characters that might break command execution.

Save the file. Let's prepare the Zabbix side now for trap receiving. In the frontend, Navigate to **Configuration | Hosts** and click **Create Host**. Fill in the following values:

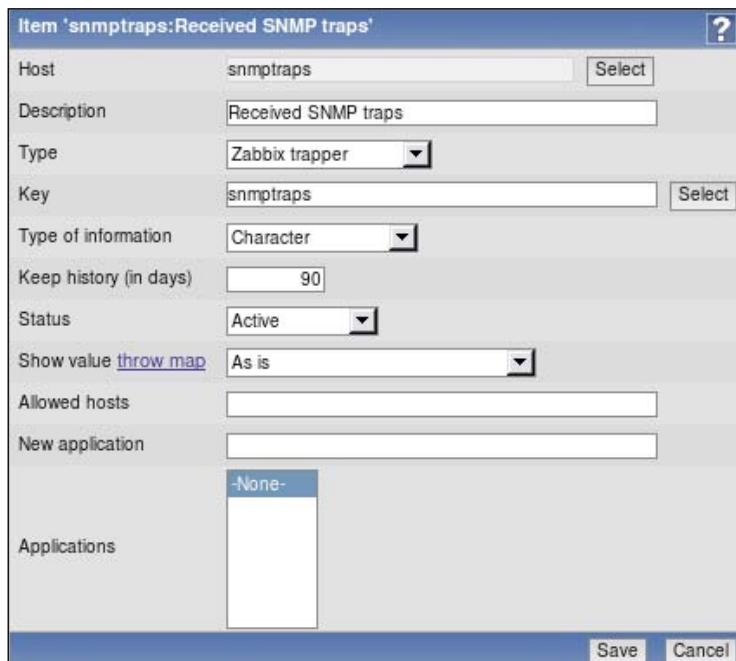
- **Name:** Enter `snmptraps`
- **Groups:** Click on **SNMP Devices** in the **Other Groups** box, then click the **<<** button; if there's any other group in the **In Groups** listbox, remove it

The screenshot shows the 'Host' configuration dialog in Zabbix. The 'Name' field is set to 'snmptraps'. In the 'In Groups' section, 'SNMP Devices' is selected. In the 'Other Groups' section, several groups are listed: Discovered Hosts, Linux servers, Templates, Windows servers, and Zabbix Servers. Below these sections are input fields for 'DNS name', 'IP address' (set to 0.0.0.0), 'Connect to' (set to 'IP address'), 'Zabbix agent port' (set to 10050), 'Monitored by proxy' (set to '(no proxy)'), 'Status' (set to 'Monitored'), and 'Use IPMI' (unchecked). At the bottom right of the dialog are 'Save' and 'Cancel' buttons.

Click the **Save** button. Note that the hostname used here, `snmptraps`, must be the same as the one we configured in the `snmptrap.sh` script, otherwise the traps won't be received in Zabbix.

Now, click on **Items** next to **snmptraps** host, then click **Create Item**. Enter these values:

- **Description:** Enter Received SNMP traps
- **Type:** Select Zabbix trapper
- **Key:** Enter snmptraps
- **Type of information:** Select Character.



When you are done, click **Save**. Again, notice how we used the exactly same key spelling as in the `snmptrap.sh` script.

We're done with configuring Zabbix for SNMP trap receiving but how will the traps get to the script we edited, and in turn, to Zabbix? This is where `snmptrapd` steps in.

Let's create a simplistic configuration that will pass all the received traps to our script. To do this, you'll have to find the location where your distribution places Net-SNMP configuration files – usually that's `/etc/snmp/`. In this directory, look for a file named `snmptrapd.conf`. If it's there, edit it (create a backup copy before you do anything), or if it's missing, create the file. Edit it as root and make it looks as follows:

```
authCommunity execute public
authCommunity execute PUBLIC
traphandle default /bin/bash /home/zabbix/bin/snmptrap.sh
```

This will do two things. Firstly, all received traps with communities `public` and `PUBLIC` will be passed for local execution. As the community string is case sensitive, we've been proactive here and entered both upper and lower case versions. Secondly, the `traphandle` line will execute the specified script, and `default` will make sure that all received traps will go to this script (that is, unless we would have other `traphandle` statements with OIDs specified, then only those received traps would get to this script that wouldn't match any other `traphandle` statement). Save this file, then start or restart the `snmptrapd` daemon as appropriate for your distribution.

 If you receive traps with various and not known in advance community strings then you could disable authorization with the `disableAuthorization yes` option in `snmptrapd.conf`.

Now we should be able to receive SNMP traps through all the chain links. Let's test that—for simplicity's sake, execute the following from the Zabbix server:

```
$ snmptrap -Ci -v 2c -c public localhost "" "NET-SNMP-MIB::netSnmpExperimental" NET-SNMP-MIB:::netSnmpExperimental s "test"
```

This slightly non-optimal Net-SNMP syntax will attempt to send an SNMP trap to `localhost` using community `public` and some nonsense OID. It will also wait for a response to verify that `snmptrapd` has received the trap successfully—that's achieved by the `-ci` flag. It uses the default port 162, so make sure that port 162 is open in your firewall configuration on the Zabbix server to receive traps.

 Confirmation waiting also makes `snmptrap` retransmit the trap. If the receiving host is slow to respond, the trap might be received multiple times before the sender receives confirmation.

If the command is successful, it will finish without any output. If it fails with the error message `snmpinform: Timeout` then several things might have gone wrong. As well as double-checking that UDP port 162 is open for incoming data, verify that the community in `/etc/snmp/snmptrapd.conf` file matches one used in the `snmptrap` command and that the `snmptrapd` daemon is actually running.

Once the command completes successfully, check out the frontend for the results. Go to **Monitoring | Latest data** and select **SNMP Devices** in the **Group** dropdown. Once the data is received in Zabbix, **snmptraps** should be available in the **Host** dropdown, so select it.

+ Description	Last check	Last value	Change	History
- other - (1 items)				
Received SNMP traps	12 Feb 2010 09:11:30	localhost "test" N ... -		History

Great, data from our test trap is received here. It's trimmed in the table view, though, so click on **History** to view all of it.

Timestamp	Value
2010.Feb.12 09:11:30	localhost "test" NET-SNMP-MIB::netSnmpExperimental

Excellent, we can see our trap in its entirety. Let's check what it looks like with several traps received one after another. From the console again, execute:

```
$ snmptrap -Ci -v 2c -c public localhost "" "NET-SNMP-MIB::netSnmpExperimental" NET-SNMP-MIB::netSnmpExperimental s "another test"
```

Refresh the history screen we had open in the browser and check whether the result is satisfactory.

Timestamp	Value
2010.Feb.12 09:14:53	localhost "another" NET-SNMP-MIB::netSnmpExperimental
2010.Feb.12 09:11:30	localhost "test" NET-SNMP-MIB::netSnmpExperimental

Our latest trap is nicely listed, with entries being ordered in descending order; but wait, everything after the first space is missing from the informative text. That's not desirable, so let's try to fix this problem. As root, open `/home/zabbix/bin/snmptrap.sh` file and look for the line that strips out address from the received information:

```
oid=`echo $oid|cut -f2 -d' '`
address=`echo $address|cut -f2 -d' '`
community=`echo $community|cut -f2 -d' '`
enterprise=`echo $enterprise|cut -f2 -d' '`
```

As can be seen here, when using a space as the separator only the second field is grabbed. We would want full details captured instead, as otherwise "A Very Important Failure" would simply turn out as "A" for us. Let's add a dash to the field parameter so that all trailing fields are captured as well:

```
address=`echo $address|cut -f2- -d' '`
```

This should solve the problem, so let's test it again:

```
$ snmptrap -Ci -v 2c -c public localhost "" "NET-SNMP-MIB::netSnmpExperimental" NET-SNMP-MIB::netSnmpExperimental s "A Very Important Failure"
```

Return to the frontend, and refresh the history listing again.

Timestamp	Value
2010.Feb.12 09:24:16	localhost "A Very Important Failure" NET-SNMP-MIB::netSnmpExperimental
2010.Feb.12 09:14:53	localhost "another" NET-SNMP-MIB::netSnmpExperimental
2010.Feb.12 09:11:30	localhost "test" NET-SNMP-MIB::netSnmpExperimental

Finally, the data from our important traps won't be lost anymore.

Trap handling schemes

While that is great for receiving all traps in a single location, it also makes traps harder to correlate to particular hosts, and especially hard to observe if you have lots and lots of trap sending hosts. In that case, it becomes very desirable to split incoming traps in some sort of a logical structure. At the very least, a split by already existing hosts can be performed. In this case all received traps would be placed in a single item for that host. If there are some particular traps or trap groups that are received very often or are very important, these can be further split in individual items.

For example, if a network switch is sending various traps, including link up and down ones, we'll probably want to place these in a single item so that they do not obscure other traps that much. If the switch has many workstations connected that are constantly switched on and off, we might even want to drop these traps before they reach Zabbix. On the other hand, if this switch has very important connections that should never go down, we might even go as far as creating an individual item for each port's notification.

All the methods work by either replacing, improving, or hooking into the handler script, `snmptrap.sh`.

Custom mapping

One way to approach trap distribution is to create custom mappings that choose appropriate destination for the trap depending on any parameters, including source host, OID, trap details, and so on. Such mapping, while being relatively cumbersome to set up is also the most flexible, as you can do all kinds of specific case handling. It also requires double configuration – most changes have to be made both to Zabbix configuration and to these mappings.

Custom mapping can use file-based lookup, a separate database, or any other kind of information storage.

Database lookups

Another method is to tap into already existing knowledge, through the Zabbix database. As the database already holds information on host/IP address relation, we can simply look up the corresponding hostname. Let's modify `snmptraps.sh` so that all traps coming from hosts defined in Zabbix end up in an `snmptraps` item for that specific host, but other traps are collected in the generic `snmptraps` host instead.

Start by modifying `/home/zabbix/bin/snmptraps.sh` and adding three lines:

```
oid=`echo $oid|cut -f11 -d'.'`  
community=`echo $community|cut -f2 -d'"'`  
export HOME=/root  
ZABBIXHOST=$(echo "select host from zabbix.hosts where ip=\\"$hostname\" order by 'hostid' limit 1;" | mysql -N 2> /dev/null)  
[[ "$ZABBIXHOST" ]] && HOST=$ZABBIXHOST  
  
str="$hostname $address $community $enterprise $oid"  
$ZABBIX_SENDER $ZABBIX_SERVER $ZABBIX_PORT -s "$HOST" -k "$KEY" -o  
"$str"
```

So what do these do? The second line queries MySQL database and checks whether a host is defined with the same IP as the trap source. If it is, `ZABBIXHOST` variable gets host name, as defined in Zabbix, assigned. Returned results are sorted by host ID and only first match is taken. Thus, if there are multiple hosts with the same IP address (which is perfectly fine configuration in Zabbix), only the oldest entry is selected. Any error output is discarded (redirected to `/dev/null`), so in case of some database misconfiguration traps are not lost but end up in the generic trap handling host.

The third line simply sets the host used for sending data to Zabbix to the entry returned from the database, if such exists.

But what about the first line? We have to look at the second line again—the `mysql` command used there does not specify user, password, or any other connection information, so for the command to succeed it would have to get this information from somewhere. For MySQL this information can be placed in `.my.cnf` file located in the user's home directory. Given that `snmptrapd` runs as root, but services often do not get all the environment variables normal logins do, we are directing further commands to look in `/root` for that file.

Which means we're not done yet; we have to create the `/root/.my.cnf` file and fill it with the required information. As root, create `/root/.my.cnf` and place the following content in it:

```
[client]  
user=zabbix  
password=mycreativepassword
```

For the password, use the same password as you used for the Zabbix server and frontend (if you don't remember this password, you can look it up in `/etc/zabbix/zabbix_server.conf`).

Now we should prepare for trap receiving on the Zabbix side. Open **Configuration | Hosts** and click on **Items** next to **Another Host**, then click the **Create Item** button. Enter these values:

- **Description:** Enter `snmptraps`
- **Type:** Select **Zabbix trapper**
- **Key:** Enter `snmptraps`
- **Type of information:** Select **Character**

Item 'Another Host:snmptraps'	
Host	Another Host
Description	snmptraps
Type	Zabbix trapper
Key	snmptraps
Type of information	Character
Keep history (in days)	90
Status	Active
Show value throw map	As is
Allowed hosts	
New application	
Applications	-None-
<input type="button" value="Save"/> <input type="button" value="Cancel"/>	

When you are done, click **Save**.

Before we send a test trap let's do one more thing, make sure that `snmptrapd` does not perform reverse lookup on received traps. While that might slightly decrease the prettiness of data, we want to keep this script simple for now and that will also improve performance a bit. To do this, add the `-n` flag for `snmptrapd` in the startup scripts, then restart it. This procedure is distribution specific.

Finally, we are ready to test our tricky setup. From "Another Host", execute:

```
$ snmptrap -Ci -v 2c -c public <Zabbix server> "" "NET-SNMP-MIB::netSnmpExperimental" NET-SNMP-MIB::netSnmpExperimental s "test"
```

Replace <Zabbix server> with the Zabbix server's IP or DNS name. This command should complete without any error messages.

Back in the frontend, navigate to **Monitoring | Latest data**.

Another Host	[- other - (9 items)]					
	CPU Load	12 Feb 2010 10:10:01	0.000000	-	Graph	
	ICMP ping performance	12 Feb 2010 10:09:29	0.47 ms	+0.22 ms	Graph	
	Incoming traffic on interface lo	12 Feb 2010 10:09:11	28.28 B	+0.46 B	Graph	
	Incoming traffic on interface eth0	12 Feb 2010 10:10:11	295.93 B	-0.63 B	Graph	
	SMTP server status	12 Feb 2010 10:09:28	Down (0)	-	Graph	
	snmptraps	12 Feb 2010 10:07:35	10.1.1.100 "test" ...	-	History	
	SSH server status	12 Feb 2010 10:09:25	Up (1)	-	Graph	
	WEB server status	12 Feb 2010 10:09:24	Up (1)	-	Graph	
	Zabbix agent version	11 Feb 2010 10:37:58	1.8.1	-	History	

Great, along with all the other monitored items, SNMP traps now are successfully sorted by host, if present. That took us some time to set up, but now it's very simple. If we want traps from some host to be handled by a specific host, we create that host and `snmptraps` item for it. All other traps go to the generic `snmptraps` host and `snmptraps` item.

But what about item lookup? The database holds information on item keys as well, so maybe we can try using that.

We need to retrieve item key from any database field based on information received in the trap. As traps include SNMP OID, that is the best candidate to map trap against item. Now, the OID can be in numeric or textual form. In Zabbix configuration, we have two fields that could be used:

- **Description.** While pretty much a free form field, it is a "friendly name", so we'd better keep it human-readable.
- **Key.** This field has more strict rules on characters it accepts, but OIDs should be fine. While not used by humans much, this field is still referred to in trigger expressions.

That means we will use the **Key** field. To keep it both short enough and somewhat human-readable, we'll set it to the last part of the received textual form OID. As the trap will be received by `snmptraps.sh`, it will try to match received OID to item key and based on that decide where to send the data.



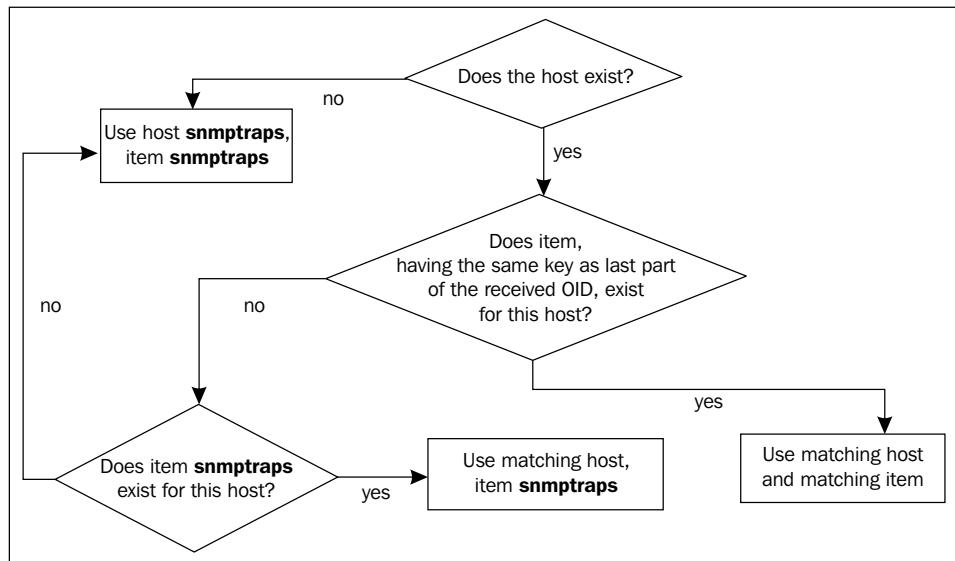
Remember that specific MIBs might have to be added to `/etc/snmp/snmp.conf` so that they are found by `snmptrapd`.

Again, as root edit the script `/home/zabbix/bin/snmptraps.sh`. Replace last two lines we just added so that it looks like this:

```
community=`echo $community|cut -f2 -d'"'`  
export HOME=/root  
ZABBIXHOSTID=$(echo "select hostid,host from zabbix.hosts where ip=\\"$hostname\" order by 'hostid' limit 1;" | mysql -N 2> /dev/null)  
ZABBIXID=$(echo $ZABBIXHOSTID | cut -d" " -f1)  
ZABBIXHOST=$(echo $ZABBIXHOSTID | cut -d" " -f2-)  
[[ "$ZABBIXHOST" ]] && {  
    TRAPOID=$(echo $oid | cut -d: -f3)  
    if [ "$TRAPOID" ]; then  
        ZABBIXITEM=$(echo "select key_ from zabbix.items where  
key_=\"$TRAPOID\" and hostid=\"$ZABBIXID\";" | mysql -N 2>  
/dev/null)  
        if [ "$ZABBIXITEM" ]; then  
            KEY=$ZABBIXITEM  
            HOST=$ZABBIXHOST  
        elif [ "$(echo "select key_ from zabbix.items where  
key_='snmptraps' and hostid=\"$ZABBIXID\";" | mysql -N 2>  
/dev/null)" ]; then  
            HOST=$ZABBIXHOST  
        fi  
    elif [ "$(echo "select key_ from zabbix.items where  
key_='snmptraps' and hostid=\"$ZABBIXID\";" | mysql -N 2>  
/dev/null)" ]; then  
        HOST=$ZABBIXHOST  
    fi  
}  
  
str="$hostname $address $community $enterprise $oid"
```

Save the file. Functionally, for our current configuration it will work exactly the same as previous version with one minor improvement. If you look at the previous version carefully, it would only check for host availability, so if you created a host but forgot to create an item with key `snmptraps` for it, sent trap would be lost. This version will check whether an item with such a key exists for that host. If not, the generic host, "snmptraps", will receive the trap.

Additionally, it will now check whether the host also has an item with key, matching last part of the OID received. A simple decision flow representation is shown in the following figure:



To test this, send an SNMP trap from "Another Host" (there is no need to restart `snmptrapd`).

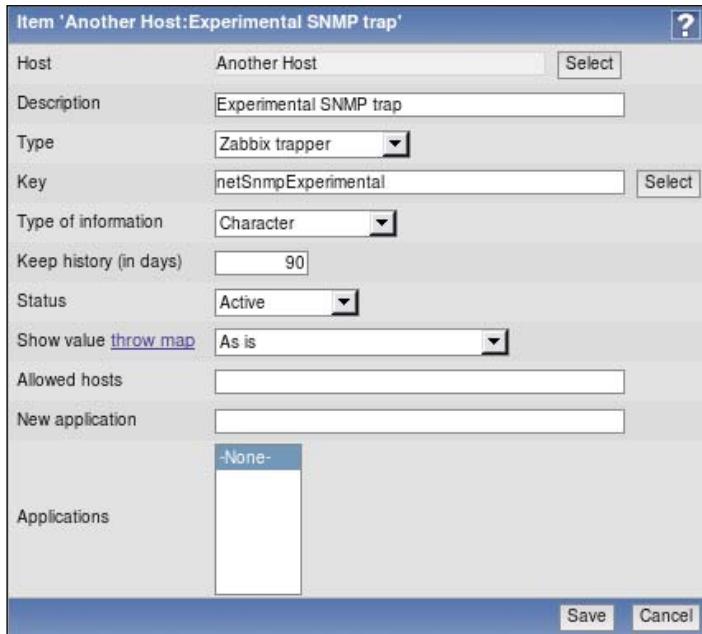
```
$ snmptrap -Ci -v 2c -c public <Zabbix server> "" "NET-SNMP-MIB::netSnmpExperimental" NET-SNMP-MIB::netSnmpExperimental s "test"
```

Replace <Zabbix server> with the Zabbix server's IP or DNS name. If you now check out **Monitoring | Latest data** for **Another Host**, the trap should correctly be placed in `snmptraps` item for this host. A trap sent from any other host, including the Zabbix server, should be placed in `snmptraps` host and `snmptraps` item – feel free to try this out. Previously, a trap sent from the Zabbix server would be lost, because the script did not check for `snmptraps` item existence – it would find the host and then push the data to this non-existent item.

Let's try out our item mapping now. Go to the Zabbix interface, **Configuration | Hosts**, click on **Items** next to **Another Host** and click the **Create Item** button. Fill in these values:

- **Description:** Enter Experimental SNMP trap
- **Type:** Select Zabbix trapper
- **Key:** Enter netSnmpExperimental

- **Type of information:** Select **Character**



When you are done, click **Save**.

Again, send a trap from "Another Host":

```
$ snmptrap -Ci -v 2c -c public <Zabbix server> "" "NET-SNMP-MIB::netSnmpExperimental" NET-SNMP-MIB::netSnmpExperimental s "test"
```

In the frontend, look at **Monitoring | Latest data**. If all went right, this time the trap data should be placed in yet another item—the one we just created.

Experimental SNMP trap	12 Feb 2010 14:02:32	10.1.1.100 "test"	...	-	History
------------------------	----------------------	-------------------	-----	---	-------------------------

Now, whenever we will have a host that will be sending us traps, we will have to decide where we want it's traps to go. Depending on that, we'll decide whether it needs it's own host with `snmptraps` item, or maybe even individual items for each trap type.

Note that this script is not optimized. While it tries to query the database only when necessary, that's still at least one, but up to three queries to the database per every trap received. If you have many devices that send traps often, it might become a performance concern.

SNMP Trap Translator

While not strictly a trap to item mapping solution, **SNMP Trap Translator (SNMPTT, <http://www.snmptt.org/>)** can be useful for making received data human-readable and also performing the mapping.

Remember that it changes passed data, so, depending on how things are set up, adding SNMPTT might require changes to item mapping, triggers or other configuration.

Of course, you can use all these methods together – simple host mapping in `snmptraps.sh`, trap translating in SNMPTT, and a thin layer of custom item mappings for some corner cases. It all depends on how complex an environment you have to monitor and how you want to see and manipulate data.

Intelligent Platform Management Interface

While SNMP is very popular and available on the majority of network-attached devices, there's a relatively new protocol that also provides monitoring data – **Intelligent Platform Management Interface or IPMI**. IPMI is usually implemented as a separate host operating system independent management and monitoring module that can also provide information when the machine is powered down. Recently IPMI is becoming more and more popular, and Zabbix has direct IPMI support.

IPMI is especially popular on so-called "lights out" or "out-of-band" management cards, available for most server offers today. As such, it might be desirable to monitor hardware status directly from these cards, as that does not depend on operating system type or even whether it's running at all.

Dell Remote Access Controller

For this section, you will need an IPMI-enabled device; usually a server with remote management card. Examples here will concentrate on the **Dell Remote Access Controller (DRAC)**, but it should be possible to apply the general principles to any product from any vendor.

Preparing Zabbix for IPMI querying

To gather data using IPMI, Zabbix must first be configured accordingly. As we already compiled Zabbix server with OpenIPMI support, the server binary should be just fine.



Check your installed OpenIPMI version. It is known that version 2.0.7 is broken and thus does not work with Zabbix. Earliest known working version is 2.0.14.

By default, the Zabbix server is configured to not start any IPMI pollers, thus any added IPMI items won't work. To change this, open `/etc/zabbix/zabbix_server.conf` as root and look for the following line:

```
# StartIPMIPollers=0
```

Uncomment it and set poller count to 3, so that it reads:

```
StartIPMIPollers=3
```

Save the file and restart `zabbix_server` afterwards.

Configuring DRAC IPMI access

An example will be provided here for setting up Dell Remote Access Controller for IPMI monitoring. For other vendor products, you will have to configure it using the available interface, usually either command line tools or web interface to allow access for the Zabbix user from the Zabbix server.

If you have access to DRAC Controller and the machine has Dell server tools installed, save the following to a file, edit the network details as appropriate, then run it as root to set up a user for Zabbix to access IPMI on the management controller. If the card already has network and IPMI access configured, there is no need to repeat that, so remove the first three lines that set the network address, enable the network, and IPMI. You should also set the password to something reasonable, it is set to `someipmipassword` in this example.



Check that the user with ID 13 is unused before resetting its username and password. You can use any other user ID, if desired.

```
#!/bin/bash
racadm setniccfg -s <IP address> <netmask> <gateway>
racadm config -g cfgLanNetworking -o cfgNicEnable 1
racadm config -g cfgIpmiLan -o cfgIpmlanEnable 1
USERID=13
racadm config -g cfgUserAdmin -o cfgUserAdminUserName -i $USERID
zabbix
racadm config -g cfgUserAdmin -o cfgUserAdminPassword -i $USERID
someipmipassword
racadm config -g cfgUserAdmin -o cfgUserAdminEnable -i $USERID 1
racadm config -g cfgUserAdmin -o cfgUserAdminIpmlanPrivilege -i
$USERID 2
```

All operations should succeed, as confirmed by the console output:

```
Static IP address enabled and modified successfully
Object value modified successfully
```

Setting up IPMI items

Before we can add IPMI items in Zabbix, we should test IPMI access. By default, IPMI uses UDP port 623, so make sure it is not blocked by some firewall. Check whether your Zabbix server has the `ipmitool` package installed—if not, install it and then execute:

```
$ ipmitool -U zabbix -H <IP address of the IPMI host> -I lanplus -L user
sensor
Password:
```

Provide the password as set in the previous step or that you have set in the IPMI configuration. We are using user-level access, as specified by the `-L user` flag so that administrative privileges should not be required for Zabbix user. The flag `-I lanplus` instructs `ipmitool` to use IPMI v2.0 LAN interface, and the `sensor` command queries the host for available sensors. If your device has IPMI running on non-default port, you can specify port with `-p` flag.

The output will contain a bunch of sensors, possibly including some like:

Processor Vcc	1.245	Volts	ok	na	0.774	0.804	1.460	1.509	na
BB Ambient Temp	26.000	degrees C	ok	na	5.000	10.000	61.000	66.000	na
System Fan 3	3402.000	RPM	ok	na	na	na	na	na	na

That looks like useful data, so let's try to monitor fan RPM in Zabbix. In the frontend, navigate to **Configuration | Hosts**. Depending on where the management card is attached to, you have two options.

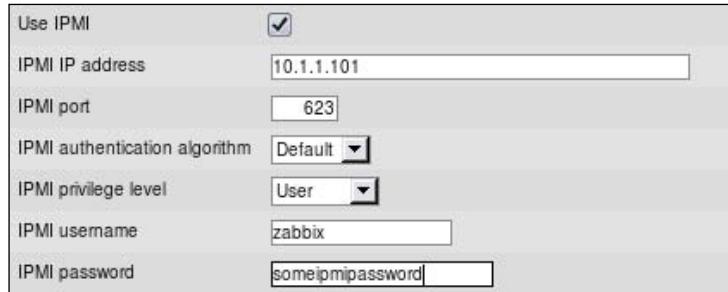
Card attached to one of the already monitored hosts

If the management card is attached to one of the already monitored hosts ("A Test Host" or "Another Host"), click on the corresponding host to open the host editing view. Make the following changes:

- **Use IPMI:** Mark the checkbox
- **IPMI IP address:** Enter the IP address, assigned to the management card
- **IPMI username:** Enter **zabbix**
- **IPMI password:** Enter the password you have set for IPMI access

If you have different configuration for the IPMI, such as a different privilege level, port, or other parameters, set them appropriately.

[ Some IPMI solutions work on the primary network interface, intercepting IPMI requests. In such a case simply set the same IP address to be used for IPMI.]



The screenshot shows a configuration dialog box for IPMI settings. The fields are as follows:

Use IPMI	<input checked="" type="checkbox"/>
IPMI IP address	10.1.1.101
IPMI port	623
IPMI authentication algorithm	Default
IPMI privilege level	User
IPMI username	zabbix
IPMI password	someipmipassword

When you are done, click **Save**.

Card attached to a different host

If the management card is attached to another server, click **Create Host**. Enter the following values:

- **Name:** Enter **IPMI Host**
- **Groups:** Click on **Linux servers** in the **Other Groups** box, then click the << button
- **Use IPMI:** Mark the checkbox

- **IPMI IP address:** Enter the IP address, assigned to the management card
- **IPMI username:** Enter zabbix
- **IPMI password:** Enter the password you have set for IPMI access

If you have different configuration for the IPMI, like different privilege level, port, or other parameters then set them appropriately.

The screenshot shows the 'Host' configuration dialog in Zabbix. The 'Name' field is set to 'IPMI Host'. In the 'Groups' section, 'Linux servers' is selected under 'In Groups'. On the right, under 'Other Groups', there are links to 'Discovered Hosts', 'SNMP Devices', 'Templates', 'Windows servers', and 'Zabbix Servers'. Below the groups, there are fields for 'New group', 'DNS name', 'IP address' (set to '0.0.0.0'), 'Connect to' (set to 'IP address'), 'Zabbix agent port' (set to '10050'), 'Monitored by proxy' (set to '(no proxy)'), 'Status' (set to 'Monitored'), 'Use IPMI' (checkbox checked), 'IPMI IP address' (set to '10.1.1.201'), 'IPMI port' (set to '623'), 'IPMI authentication algorithm' (set to 'Default'), 'IPMI privilege level' (set to 'User'), 'IPMI username' (set to 'zabbix'), and 'IPMI password' (set to 'someipmipassword'). At the bottom right are 'Save' and 'Cancel' buttons.

When you are done, click **Save**.

Notice that we did not enter the IP address or DNS name for the host itself—we will be using IPMI items for this host only.

Creating IPMI item

Now that we have the host part of IPMI connectivity sorted out, it's time to create actual items. Open **Configuration | Hosts** and click on **Items** next to the host you configured IPMI access for (one of the "A Test Host", "Another Host" or "IPMI Host"), then click on **Create Item**. Enter these values:

- **Description:** Enter **System Fan 3** (or, if your IPMI-capable device that does not provide such sensor, choose another useful sensor)
- **Type:** Select **IPMI agent**
- **IPMI sensor:** Enter **System Fan 3**
- **Key:** Enter **System_Fan_3**
- **Units:** Enter **RPM**
- **Update interval:** Change to **60**
- **Keep history:** Change to **7**

Item 'IPMI Host:System Fan 3'

Host	IPMI Host	<input type="button" value="Select"/>
Description	System Fan 3	
Type	IPMI agent	<input type="button" value="Select"/>
IPMI sensor	System Fan 3	
Key	System_Fan_3	
Type of information	Numeric (unsigned)	
Data type	Decimal	
Units	RPM	
Use multiplier	Do not use	
Update interval (in sec)	60	
Flexible intervals (sec)	No flexible intervals	
New flexible interval	Delay	50
	Period	1-7,00:00-23:59
	<input type="button" value="Add"/>	
Keep history (in days)	7	
Keep trends (in days)	365	
Status	Active	
Store value	As is	
Show value throw map	As is	
New application		
Applications	-None-	
<input type="button" value="Save"/> <input type="button" value="Cancel"/>		

When you are done, click **Save**.

If your IPMI-monitored host does not provide such a sensor, check the output of sensor listing command.

```
$ ipmitool -U zabbix -H <IP address of the IPMI host> -I lanplus -L user  
sensor
```

Choose some useful sensor and use the value from the first column for the description, sensor, and key fields. Replace spaces with underscores for the key field.



If you have set a long IPMI password and revisit host editing screen, you'll see it being trimmed. This is normal, as maximum password length for IPMI v2.0 is 20 characters.



Let's check out the results of our work, open **Monitoring | Latest data**, then select the host you configured an IPMI item for.

+ Description ▲	Last check	Last value	Change	History
- other - (1 Items)				
FAN 1 RPM	20 Apr 10:52:54	6.08 KRPM	-75 RPM	Graph

Great, the hardware's state information is gathered correctly. What's even better, this information is retrieved independently of the installed operating system or specific agents and it is retrieved even if there is no operating system running or even installed.

So IPMI, while not as widespread as SNMP, can provide software-independent hardware monitoring for some devices, usually servers.

Summary

Having explored basic monitoring with Zabbix agent before, we looked at two major agentless monitoring solutions in this chapter, especially SNMP, and to a somewhat lesser extent, IPMI. Given the wide array of devices supporting SNMP, this knowledge should help us with both retrieving information from devices like printers, switches, UPSes, and others, while also listening and managing incoming SNMP traps from those.

Beware of starting to monitor large amount of network devices, especially if they have many interfaces. For example, adding 10 switches with 48 ports, even if you monitor single item per switch once a minute only, will make Zabbix poll eight new values per second (480 ports once a minute results in $480/60=8$ new values per second). Usually more values per port are monitored, so such an increase can bring a Zabbix server down, and severely impact network performance.

IPMI, being more popular as the out-of-band monitoring and management solution for servers, should help us to watch over hardware states for the compliant devices.

5

Managing Hosts, Users, and Permissions

We created some hosts and host groups before, thus exploring the way items can be grouped and attached to hosts. Now is the time to take a closer look at these concepts and see what benefits they provide.

Additionally, we will find out about the permission system in Zabbix, so you will be able to allow partial access to other users as desired.

Host and host groups

A host can be considered as a basic grouping unit in Zabbix's configuration. As you might remember, hosts are used to group items, which in turn are basic data acquiring structures. Each host can have any number of items assigned spanning all item types—Zabbix agents, simple checks, SNMP, IPMI, and so on. An item can't exist on its own, so hosts are mandatory.

Managing Hosts, Users, and Permissions

Zabbix does not allow a host to be "left alone", that is, not belonging to any group, since version 1.8. Let's look at what host groups we have currently defined—in the frontend, open **Configuration | Host groups**.

CONFIGURATION OF HOST GROUPS			
HOST GROUPS			
Displaying 1 to 6 of 6 found			
<input type="checkbox"/>	Name ▲	#	Members
<input type="checkbox"/>	Discovered Hosts	Hosts (0) Templates (0)	-
<input type="checkbox"/>	Linux servers	Hosts (3) Templates (0)	Another Host , A Test Host , IPMI Host
<input type="checkbox"/>	SNMP Devices	Hosts (2) Templates (0)	SNMP Device , snmptraps
<input type="checkbox"/>	Templates	Hosts (0) Templates (42)	Template 3COM 3824, Template 3COM 4400, Template AIX, Template APC Automatic Transfer Switch, Template APC Battery, Template App MySQL, Template C3750-48TS, Template Cisco 837, Template Cisco 877, Template Cisco 2960, Template Cisco PIX, Template Cisco PIX515E, Template Cisco PIX 525, Template Dell OpenManage, Template Dell PowerConnect 5224, Template Dell PowerConnect 5324, Template Dell PowerConnect 6248, Template Dell PowerEdge, Template FreeBSD, Template Hibernate, Template HPUX, Template HP ColorLaserJet, Template HP InsightManager, Template HP Procurve, Template IPMI Sun Fire X4100 M2, Template Java, Template Linux, Template MacOS X, Template Microsoft Exchange 2003, Template Microsoft Exchange 2007, Template Microsoft SQLServer 2005, Template NetScreen 25, Template Netware, Template OpenBSD, Template pfSense, Template SNMPv1 Device, Template SNMPv2 Device, Template Solaris, Template Standalone, Template Tomcat, Template Tru64, Template Windows
<input type="checkbox"/>	Windows servers	Hosts (0) Templates (0)	-
<input type="checkbox"/>	Zabbix Servers	Hosts (1) Templates (0)	Zabbix Server

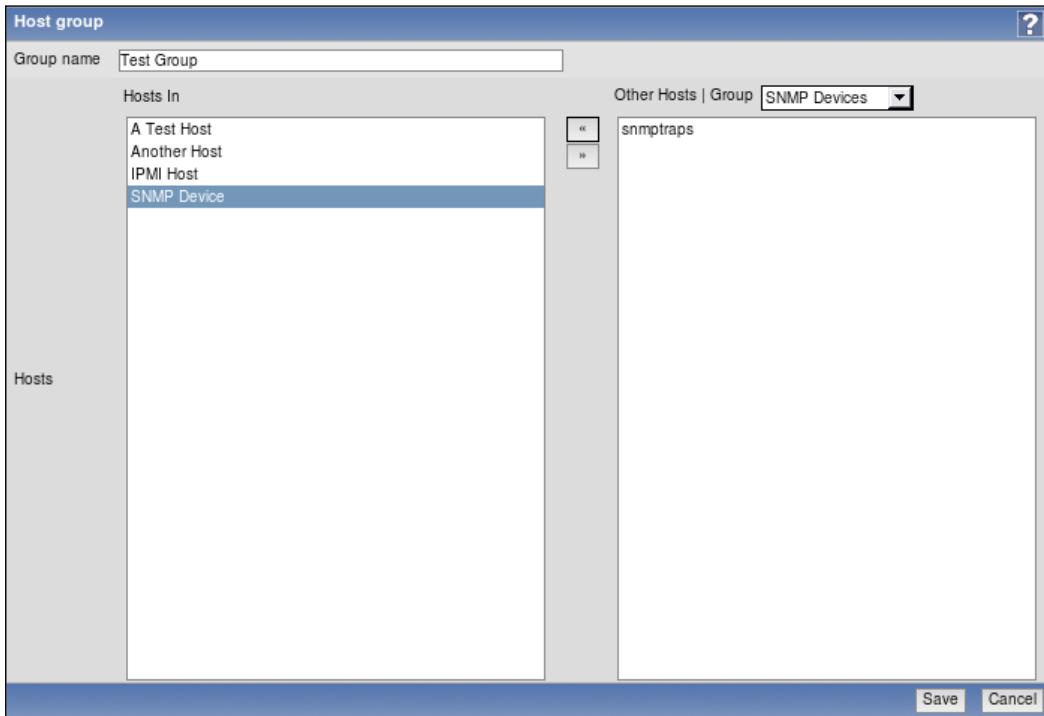
The first thing that catches the eye is that group **Templates** seems to have a large number of templates already. These are provided as examples, so that you can later quickly reference them for some hints on items. We'll ignore these for now. We can also see an empty **Windows servers** group, and the **Zabbix Servers** group which contains a single example host. The interesting part is at the top of the table—we can see both groups we created along the way with all the corresponding members. Note, you might be missing **IPMI Host** if you set up IPMI for one of the existing hosts. This table is fairly simple with just a group name, a count of the number of group members (individually denoting hosts and templates contained in the group), and individual members being listed.

As can be seen, individual members are color coded.

- Black: Normal, enabled host
- Red: Normal, disabled host
- Grey: Host template

Let's create another host group and assign some hosts to it. Click the **Create Group** button. Enter **Test Group** in the **Group name** field, then select **Linux servers** in the **Group** dropdown above the **Other Hosts** list box. In the filtered list, select our custom created hosts—**A Test Host**, **Another Host**, and **IPMI Host** (if applicable). You can use the *Ctrl* and *Shift* keys to select multiple entries. When you have these hosts selected, click the **<<** button.

Now select **SNMP Devices** in the **Group** dropdown and select **SNMP Device**, then click on the **<<** button again.



This form allows easy selection of any number of hosts to be added when a new group is created. You can freely move hosts from one box to another until you are satisfied with the result. When you are done, click **Save**.

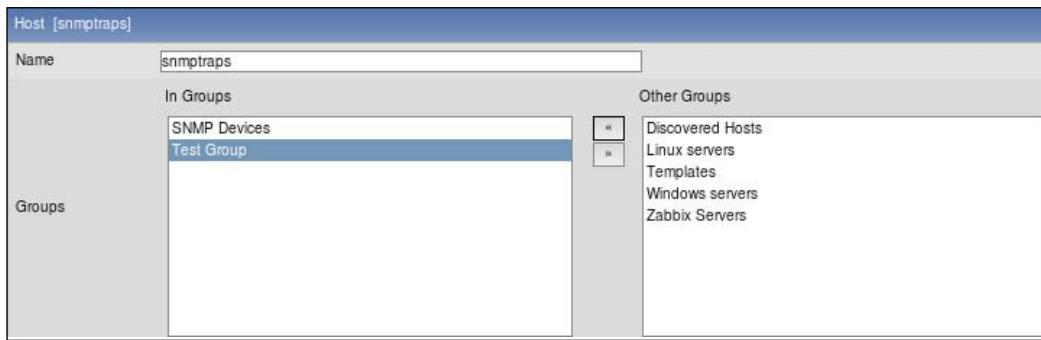
A new group now appears in the list. As we can see, it contains the four hosts we just added.

<input type="checkbox"/>	Test Group	Hosts (4)	Templates (0)	Another Host , A Test Host , IPMI Host , SNMP Device
--------------------------	----------------------------	---------------------------	-------------------------------	--

But wait, the two first groups have 3 and 2 hosts correspondingly.

<input type="checkbox"/>	Linux servers	Hosts (3) Templates (0)	Another Host, A Test Host, IPMI Host
<input type="checkbox"/>	SNMP Devices	Hosts (2) Templates (0)	SNMP Device, snmptraps

Right, we forgot to add the **snmptraps** host. Move your mouse cursor over it—notice how this (and every other host on this page) is a link. Clicking it will take you to host details, so do that now. As we can see on the host editing form, it is already in one group, **SNMP Devices**. Click on **Test Group** in the **Other Groups** list box, then click the << button.



When you are done, click **Save**.

You have probably guessed by now that a host can belong to any number of groups. That allows you to choose grouping based on any arbitrary decisions, like having a single host in groups "Linux servers", "Europe servers", and "DB servers".

Now you are back in the host list, so return to host group list by navigating to **Configuration | Host groups**. **Test Group** contains five hosts, as it should. Let's say you would like to disable a whole group of hosts, or even several host groups. Maybe you have a group of hosts that are retired, but that you don't want to delete just yet, or maybe you want to disable hosts created for testing when creating an actual production configuration on the Zabbix server. The group listing provides an easy way to do that. Mark the checkboxes next to **Linux servers** and **SNMP Devices** entries, then choose **Disable selected hosts** action in the dropdown at the bottom of the list, then click the **Go** button and confirm the pop up.

After this operation, all black hosts should be gone—they should be red now, indicating they are in the disabled state.

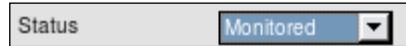
This time you could also have marked checkbox next to **Test Group** only as **Linux servers** and **SNMP Devices** are subsets of **Test Group**, the final effect would be the same. After doing this, we remember that **snmptraps** is a generic SNMP trap receiving host which probably should be left enabled. Again, click on it to open the host editing details page.

While we have the host details page open, we can take a quick look at DNS name and IP address fields. As you can see, only a single IP and DNS field is available, and **Connect to** controls which is used for server-side initiated checks. This means that Zabbix currently does not natively support multihomed hosts, with the only exception being the ability to query IPMI on another IP address



It is a common practice to use a separate network for management interface access so that application caused problems on the network do not impact access to the management interfaces. If that is the case, Zabbix server will need access to the management network, and the host operating system should provide correct routing.

Change **Status** dropdown to **Monitored** and click **Save**.



You should now see a host list with one disabled host (indicated by red text saying **Not monitored** in the **Status** column), and one enabled host (indicated by green text **Monitored** in the **Status** column). Let's enable back the **SNMP Device** – click on the **Not monitored** text next to it. That leaves us with two enabled devices on the list. Select **Linux servers** in the **Group** dropdown and mark checkboxes next to the still disabled ones (if you have three of them, you can use range select feature again by holding down *Ctrl*, clicking first entry, then last entry in the list that you want to see selected) and choose **Activate selected** in the dropdown at the bottom of the list, then click the **Go** button. Click on **OK** in the confirmation dialog. Finally, we are back to having all hosts enabled again. We used four methods to change host state here:

- Changing state for whole groups in **Configuration | Host groups** area
- Changing state for a single host by changing the **Status** dropdown value in that host's properties page
- Changing state for a single host by using controls for each host in the **Status** column in host configuration list
- Changing state for a single host or multiple hosts by marking the relevant checkboxes in the host configuration list and using the controls at the bottom of the list

We created the last host group by going through the group configuration screen. As you might remember, another way is to use the **New group** field when creating or editing the host—this created the group and simultaneously added the host to that group.

The host list in the configuration screen also is useful in another way. It provides a quite nice and quick way to see which hosts are down. While monitoring section gives quite extensive information on state of specific services and conditions on each device, sometimes you will want a quick peek at device status—for example, determining availability for all devices in a particular group such as printers, routers, or switches. Configuration divulges this information in a list that contains almost no other information to distract you. If we would now select **all** in the **Group** dropdown, we would see all the hosts this installation has:

<input type="checkbox"/>	Name	Applications	Items	Triggers	Graphs	DNS	IP	Port	Templates	Status	Availability
<input type="checkbox"/>	Another Host	Applications (0)	Items (10)	Triggers (0)	Graphs (0)	-	192.168.3.93	10050	-	Monitored	  
<input type="checkbox"/>	A Test Host	Applications (0)	Items (9)	Triggers (1)	Graphs (0)	-	127.0.0.1	10050	-	Monitored	  
<input type="checkbox"/>	IPMI Host	Applications (0)	Items (1)	Triggers (0)	Graphs (0)	-	0.0.0.0	10050	-	Monitored	  
<input type="checkbox"/>	SNMP Device	Applications (0)	Items (1)	Triggers (0)	Graphs (0)	-	10.1.1.200	10050	-	Monitored	  
<input type="checkbox"/>	snmptraps	Applications (0)	Items (1)	Triggers (0)	Graphs (0)	-	0.0.0.0	10050	-	Monitored	  
<input type="checkbox"/>	Zabbix Server	Applications (12)	Items (102)	Triggers (44)	Graphs (4)	-	127.0.0.1	10050	Template_Linux	Not monitored	  

This time we are interested in two columns—**Status** and **Availability**. As can be seen, we have one host that is not monitored, and this information is easily noticeable—printed in red, it stands out from the usual green entries. The availability column shows the internal state, as determined by Zabbix, for each host and for each polled item type. If Zabbix tries to get data from a host but fails, the availability of that host for this specific type of information is determined to be unavailable, as has happened here with **Another Host**. Both the availability status and error message are displayed and preserved for three separate types of items polled by the Zabbix server:

- Zabbix agent (passive)
- SNMP
- IPMI

On the other hand, the availability of the **snmptraps** host is unknown for all polled item types, as Zabbix has never tried to retrieve any data from it (that is, there are no items configured for it that Zabbix server polls). Again, both unknown and unavailability visually differ from the available hosts, providing a quick overview.

That about wraps it up for host and host group management in Zabbix. Host group usefulness extends a bit past frontend management, though—we'll see how exactly in the next section.

Users, user groups, and permissions

Hosts manage monitored entity (host) information and are used to group basic information gathering units – items. User accounts in Zabbix control access to the monitored information.

Authentication methods

Before we look at more detailed user configuration, it might be helpful to know that Zabbix supports three authentication methods. Navigate to **Administration | Authentication** to see authentication configuration.

The screenshot shows the 'LDAP Authentication' configuration page. The 'Default authentication' dropdown is set to 'Internal'. Other fields include 'LDAP Host' (empty), 'Port' (LDAP selected), 'Base DN' (empty), 'Search attribute' (uid), 'Bind DN*' (empty), 'Bind Password*' (empty), 'LDAP Authentication Enabled' (unchecked), 'Test Authentication' (must be valid LDAP User), 'Login' (Admin), and 'User Password' (empty). Buttons at the bottom are 'Save' and 'Test'.

As can be seen here, the three authentication methods are:

- **HTTP:** Users are authenticated against web server HTTP authentication mechanisms. One of the uses is scripted, automated data retrieval through a web interface. Support for HTTP authentication basically allows the use of any of the authentication methods for Zabbix that the web server supports, and in the case of Apache, that's many.
- **LDAP:** Users are authenticated against an LDAP server. This can be handy if all enterprise users that need access to Zabbix are already defined in an LDAP structure.
- **Internal:** With this method, users are authenticated against Zabbix's internal store of users and passwords. We will use this method.

Creating a user

The initial Zabbix installation does not contain many predefined users—let's look at the user list. Navigate to **Administration | Users** and choose the **Users** in the first dropdown.

USERS											User group	All	
Displaying 1 to 2 of 2 found													
	Alias	Name	Surname	User type	Groups	Is online?	Login	GUI access	API access	Debug mode	Status		
<input type="checkbox"/>	Admin	Zabbix	Administrator	Zabbix Super Admin	Zabbix administrators	Yes (Tue, 16 Feb 2010 10:13:38 +0200)	Ok	System default	Disabled	Disabled	Enabled		
<input type="checkbox"/>	guest	Default	User	Zabbix User	Guests	No (Mon, 15 Feb 2010 09:24:06 +0200)	Ok	System default	Disabled	Disabled	Enabled		

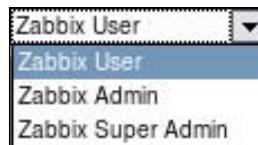
That's right, only two user accounts are defined—**Admin** and **guest**. We have been logged in as **Admin** all the time. On the other hand, the **guest** user is used for unauthenticated users. Before we logged in as **Admin**, we were **guest**. The user list shows some basic information about the users, like what groups they belong to, if they are logged in, when their last login was, and whether their account is enabled.



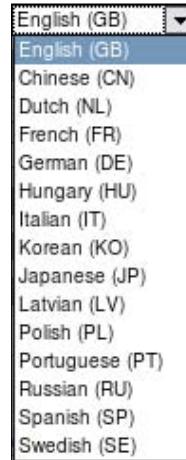
By granting access permissions to the **guest** user it is possible to allow anonymous access to resources.

Let's create another user for ourselves. Click the **Create User** button located in the upper-right corner. We'll look at all available options for a user account while filling in the appropriate ones.

- **Alias:** Enter **monitoring-user**. This is essentially a username.
- **Name:** Enter **monitoring**. In this field you would normally enter the user's real name.
- **Surname:** Enter **user**. Obviously, this field normally contains the user's real surname.
- **Password:** Choose and enter a password, then retype it in the next field.
- **User type:** This dropdown offers three available user types. We'll leave it at **Zabbix User** for this user. For reference, these types mean:



- **Zabbix User:** These are plain users that only have access to the Monitoring section in the main menu
 - **Zabbix Admin:** These users in addition to the Monitoring section have access to the Configuration section, so they are able to reconfigure parts of Zabbix itself
 - **Zabbix Super Admin:** These users have full access to Zabbix; including the Monitoring, Configuration, and Administration sections
- **Groups:** Just like hosts, user accounts can be grouped. We won't assign our new user to any groups for now.
 - **Language:** The frontend has translations of various maturity and each user can choose their own preference. We'll leave this at **English (GB)**.

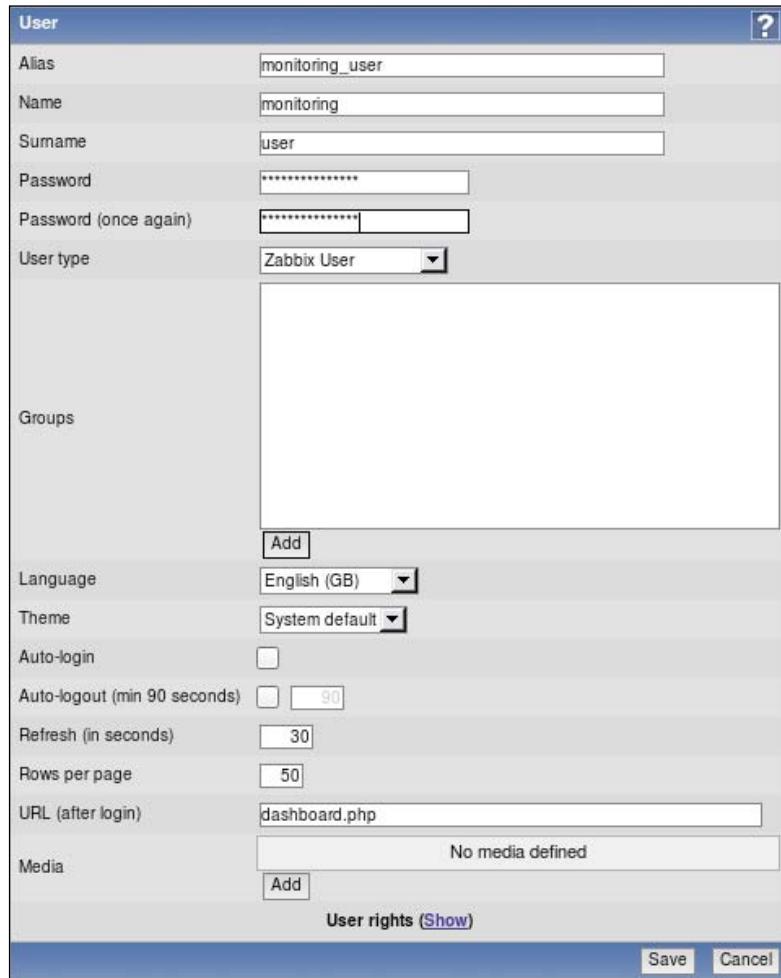


- **Theme:** The zabbix frontend supports theming. Currently, there are only two themes included, though. We'll leave theme to be **System default** (which is **Original blue**)



- **Auto-login:** Marking this option will automatically login the user after they have logged in at least once manually. Automatic login is performed with browser cookies. We won't be using automatic login for this user.
- **Auto-logout:** You can make a particular account automatically log out after a specific time of inactivity. The minimum time period that can be set is 90 seconds, and the maximum is about 166 minutes. There is no need to set automatic logout here.
- **Refresh:** The time in seconds between page refreshes when in the **Monitoring** section. While smaller values might be nice to look at when first setting up and having items with short check intervals, they somewhat increase the load on the server, and if the page contains a lot of information then sometimes it might not even finish loading before the next refresh kicks in. Let's set this to **60** for this user—after all, he can always refresh manually when testing something.
- **Rows per page:** Each user can have an individual maximum rows per page setting. If the returned data exceeds this parameter, the interface splits the data into multiple pages. We won't change this parameter.
- **URL (after login):** A user might wish to always see the same screen after logging in—be it Overview, trigger list, or any other page. This option allows the user to customize that. The URL entered is relative to the Zabbix directory, so let's make this user always see **Monitoring | Dashboard** when he logs in by entering **dashboard.php** here.
- **Media:** Here users can have various media assigned so that Zabbix knows how to alert them. Media types include e-mail addresses or numbers to send SMS to.

The final result should look as follows:



The screenshot shows the 'User' configuration dialog box. The 'Alias' field is set to 'monitoring_user'. The 'Name' field is set to 'monitoring'. The 'Surname' field is set to 'user'. The 'Password' and 'Password (once again)' fields both contain masked text. The 'User type' dropdown is set to 'Zabbix User'. Below this, there is a large empty box labeled 'Groups' with an 'Add' button. Under 'Language', 'English (GB)' is selected. Under 'Theme', 'System default' is selected. The 'Auto-login' checkbox is unchecked. The 'Auto-logout (min 90 seconds)' field has a value of '90'. The 'Refresh (in seconds)' field has a value of '30'. The 'Rows per page' field has a value of '50'. The 'URL (after login)' field contains 'dashboard.php'. Under 'Media', it says 'No media defined' with an 'Add' button. At the bottom, there is a link 'User rights (Show)' and a blue footer bar with 'Save' and 'Cancel' buttons.

If it does, click **Save**.

Now it would be nice to test this new user. It is suggested that you launch another browser for this test so that any changes are easy to observe. Let's call the browser where you have the administrative user logged in "Browser 1" and the other one "Browser 2". In Browser 2, open the Zabbix page and log in as "monitoring_user", supplying whatever password you entered before.

The screenshot shows the Zabbix 1.8.1 Personal Dashboard. At the top, there is a navigation bar with links for Monitoring, Inventory, Reports, Dashboard, Overview, Web, Latest data, Triggers, Events, Graphs, Screens, Maps, and IT services. A search bar labeled "SEARCH:" is also present. Below the navigation bar, the history is shown as "History: Dashboard". The main area is titled "PERSONAL DASHBOARD" and contains several sections:

- Favourite Graphs**: A list with a "Graphs »" link.
- Favourite Screens**: A list with a "Screens »" link.
- Favourite Maps**: A list with a "Maps »" link.
- System status**: A table showing system status by host group. The columns are Host group, Disaster, High, Average, Warning, Information, and Not classified. The table is currently empty.
- Host status**: A table showing host status by host group. The columns are Host group, Without problems, With problems, and Total. The table is currently empty.
- Last 20 issues**: A table showing the last 20 issues. The columns are Host, Issue, Last change, Age, Ack, and Actions. The table is currently empty.
- Web monitoring**: A table showing web monitoring status by host group. The columns are Host group, Ok, Failed, In progress, and Unknown. The table is currently empty.

At the bottom of the dashboard, it says "Zabbix 1.8.1 Copyright 2001-2009 by SIA Zabbix" and "Connected as 'monitoring_user'".

As expected, the user is directed to the dashboard directly after logging in. Also, the screen is notably different—the main menu entries "Configuration" and "Administration" are missing here. It also looks quite empty in **System status** and **Last 20 issues** sections, no systems or issues are visible. Open **Monitoring | Overview**. Despite the **Group** dropdown saying **all**, detail view claims that there are **No triggers defined**. Why so?

By default, users don't have access to any systems. When our new user logs in, nothing is displayed in the monitoring section, because we did not grant any privileges, including read only. Back in Browser 1, click on **monitoring_user** in the **Alias** column. One minor thing to notice – instead of input fields for the password, this time a button that says **Change password** is visible. If you ever have to reset a password for some user, clicking this button will reveal the password input fields again, allowing a password update along with any other changes that might have been made.



But there's one link at the bottom we still haven't used – **Show** next to the **User rights**.



User rights sounds just like what we want, so click on this link.

	Read-write	Read only	Deny
Host groups			Discovered Hosts Linux servers SNMP Devices Templates Test Group Windows servers Zabbix Servers
Hosts			A Test Host Another Host IPMI Host SNMP Device snmptraps Template_3COM_3824 Template_3COM_4400 Template_AIX Template_AP_C_Automatic_Transfer_Switch Template_APC_Battery Template_App_SQL Template_C3750-48TS Template_Cisco_2960 Template_Cisco_837 Template_Cisco_877

Now that looks very close. There are hosts and host groups, split in read-write, read only, and deny permissions. There's just one problem – there is no way to change these permissions.

It's time to find out the truth – **access in Zabbix can be granted to user groups only**. We conveniently skipped adding our user to any groups, so now is a good time to fix that.

Creating user groups

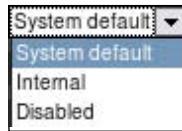
Navigate to **Administration | Users** and take a look at the list of current user groups.

USER GROUPS							
Displaying 1 to 10 of 10 found							
#	Name ▲	Members	Users status	GUI access	API access	Debug mode	
	API access	Users (0)	Enabled	System default	Enabled	Disabled	
	Database administrators	Users (0)	Enabled	System default	Disabled	Disabled	
	Disabled	Users (0)	Disabled	System default	Disabled	Disabled	
	Guests	Users (1) guest	Enabled	System default	Disabled	Disabled	
	Head of IT department	Users (0)	Enabled	System default	Disabled	Disabled	
	Network administrators	Users (0)	Enabled	System default	Disabled	Disabled	
	Security specialists	Users (0)	Enabled	System default	Disabled	Disabled	
	UNIX administrators	Users (0)	Enabled	System default	Disabled	Disabled	
	WEB administrators	Users (0)	Enabled	System default	Disabled	Disabled	
	Zabbix administrators	Users (1) Admin	Enabled	System default	Disabled	Disabled	
Enable selected		Go (0)					

As can be seen, there are already ten predefined groups, giving you some idea of how users could be organized. That organization can be based on system categories, systems, management roles, physical locations, and so on. For example, you might have a group of administrators in headquarters and some in a branch location. Each group might not be interested in the UPS status in the other location, so you could group them as "HQ admins" and "Branch admins". A user can belong to any amount of groups, so you can create various schemes as real world conditions require.

Let's create a new group for our user. Click on **Create Group** in the upper-right corner. Now that's more like it, we can finally see controls for various permission levels. Let's fill in the form and find out what each control does.

- **Group name:** Enter **Our Users**.
- **Users:** Here we can add users to the group we are creating. Our current installation has very few users, so finding the correct username with all users displayed is easy. If we had many users, we could use the **Other Groups** dropdown to filter user list and more easily find what we are looking for. Select **monitoring_user** and click the << button.
- **GUI access:** This option allows us to choose the authentication method for a specific group. This allows for a configuration where most users are authenticated against LDAP, but some users are authenticated against the internal user database. It also allows us to set no GUI access for some groups, which can then be used for users that only need to receive notifications. We'll leave this option at **System default**.



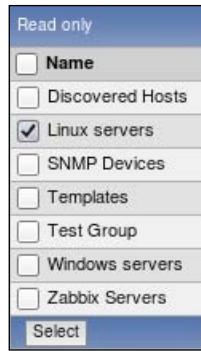
- **Users status:** With a single option, the whole group can be disabled or enabled. As predefined groups might hint, this is a nice way to easily disable individual user accounts by simply adding them to a group that has this option set to **Disabled**. We want our user to be able to log in, so this option will stay at **Enabled**.
- **API access:** Users in this group would have access to the API, used to communicate with Zabbix programmatically.
- **Debug mode:** This option gives users access to frontend debug information. It is mostly useful for Zabbix developers.
- **Rights:** Now this is the section we were initially after. There are three boxes, labeled **Read-write**, **Read only**, and **Deny**. Each has buttons below it, named **Add** and **Delete selected**, which seem to modify respective permission. Our user already has no permissions, so we will want to add some kind of permissions in the first two boxes. Click on **Add** below the **Read-write** box. This opens a new window with some options.

It also provides us with another valuable bit of information. If you look at the window contents carefully, you'll notice something common for all of these entries—they are all host groups. We finally have gotten the essential information together—in Zabbix, **permissions can be set for user groups on host groups only**.

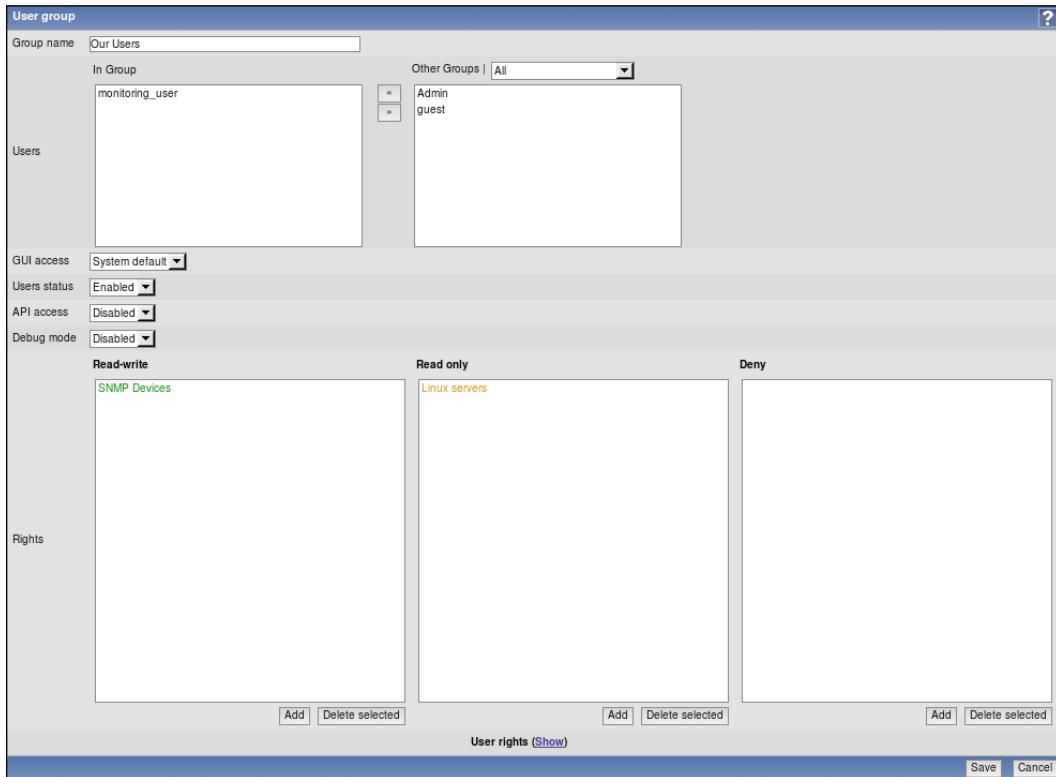
Mark the checkbox next to **SNMP Devices** and click the **Select** button.



We can now see **SNMP Devices** added to the **Read-write** box, in green font. Next, click on **Add** below the **Read only** box. An identical pop up to the previous one is opened. This time, mark the checkbox next to the **Linux servers** entry, then click on **Select**.



Now the **Read only** box has **Linux servers** listed in a somewhat orange-ish color. The final form should look like this:



But again you can see the **User rights (Show)** link at the bottom. Weird, we just set up user rights – let's click on it to find out what's hiding there.

User rights (Hide)		
	Read-write	Read only
Host groups	SNMP Devices	Linux servers
Hosts	SNMP Device snmptraps	A Test Host Another Host IPMI Host
		Deny Discovered Hosts Templates Test Group Windows servers Zabbix Servers
		Template_3COM_3824 Template_3COM_4400 Template_AIX Template_APP_Automatic_Transfer_Switch Template_APP_Battery Template_App_MySQL Template_C3750-48TS Template_Cisco_2960 Template_Cisco_837 Template_Cisco_877 Template_Cisco_PIX Template_Cisco_PIX_525 Template_Cisco_PIX515E Template_Dell_OpenManage Template_Dell_PowerConnect_5224

This view shows effective user rights which now look more interesting. We can see what the exact access permissions will look like – which hosts will be allowed read and write access, which will have read only, and for which there will be no access at all. This looks about right, so click on **Save**. Group will be successfully added, and we will be able to see it in the group list:

<input type="checkbox"/>	Our Users	Users (1)	monitoring_user	Enabled	System default	Disabled	Disabled
--------------------------	---------------------------	---------------------------	---------------------------------	-------------------------	--------------------------------	--------------------------	--------------------------

Let's get back to Browser 2. Navigate to **Monitoring | Latest data**. Great, our new user can view data from all the hosts. But we also added write permissions to one group for this user, what's up with the "Configuration" menu? Let's recall the user creation process – wasn't there something about user types? Right, we were able to choose between three user types, and we chose "Zabbix user", which, as discussed, was not allowed to access configuration.

We'll create another, more powerful user, so in Browser 1 go to **Administration | Users** and select **Users** from the dropdown, then click on **Create User** button. Fill in these values:

- **Alias:** Enter **advanced_user**.
- **Name:** Enter **advanced**.
- **Surname:** Enter **user**.

- **Password:** Enter some password in both fields. You can use same password as for the "monitoring_user" to make it easier to remember.
- **User type:** Select **Zabbix Admin**. This is will make quite a large difference, as we will see soon.
- **Refresh:** Enter **60**.
- **URL (after login):** Let's have this user view a different page right after logging in. Currently active triggers might do. Enter **tr_status.php** here.

The screenshot shows the 'User' configuration dialog box. It contains the following fields:

- Alias:** advanced_user
- Name:** advanced
- Surname:** user
- Password:** [REDACTED]
- Password (once again):** [REDACTED]
- User type:** Zabbix Admin
- Groups:** (Empty list with an 'Add' button)
- Language:** English (GB)
- Theme:** System default
- Auto-login:** (checkbox)
- Auto-logout (min 90 seconds):** (checkbox) 90
- Refresh (in seconds):** 60
- Rows per page:** 50
- URL (after login):** tr_status.php
- Media:** (Empty list with an 'Add' button)

At the bottom, there are 'User rights (Show)' and 'Save' / 'Cancel' buttons.

When you are done, click the **Save** button.

Let's use Browser 2 now. In the upper-right corner, click on the **Logout** link, then log in as "advanced_user". This time we can see **Configuration** menu and only because we set user type to "Zabbix Admin". Let's check out what we have available there—open **Configuration | Hosts**.

<input type="checkbox"/>	Alias	Name	Surname	User type	Groups
<input type="checkbox"/>	Admin	Zabbix	Administrator	Zabbix Super Admin	Zabbix administrators
<input type="checkbox"/>	advanced_user	advanced	user	Zabbix Admin	
<input type="checkbox"/>	guest	Default	User	Zabbix User	Guests
<input type="checkbox"/>	monitoring_user	monitoring	user	Zabbix User	Our Users

Wait, **No hosts defined**? How is that possible, we set this user as "Zabbix Admin" type. We probably should look at the user list back in the Browser 1.

<input type="checkbox"/>	Alias	Name	Surname	User type	Groups
<input type="checkbox"/>	Admin	Zabbix	Administrator	Zabbix Super Admin	Zabbix administrators
<input type="checkbox"/>	advanced_user	advanced	user	Zabbix Admin	
<input type="checkbox"/>	guest	Default	User	Zabbix User	Guests
<input type="checkbox"/>	monitoring_user	monitoring	user	Zabbix User	Our Users

Here, we can easily spot our mistake – somebody forgot to add `advanced_user` to any groups. We'll fix that now, but this time, we'll use user properties form. Click on **advanced_user** in the **Alias** column and in the upcoming form click on **Add** next to the **Groups** field. Mark checkbox next to **Our Users**, then click **Select**.

User "advanced_user"

Alias	advanced_user
Name	advanced
Surname	user
Password	Change password
User type	Zabbix Admin
Groups	<p>Our Users</p> <p>Add Delete selected</p>
Language	English (GB)
Theme	System default
Auto-login	<input type="checkbox"/>
Auto-logout (min 90 seconds)	<input type="checkbox"/> 90
Refresh (in seconds)	60
Rows per page	50
URL (after login)	tr_status.php
Media	No media defined Add

[User rights \(Show\)](#)

[Save](#) [Delete](#) [Cancel](#)

When you are done, click **Save**. In Browser 2, simply refresh host configuration tab—that should reveal two hosts, **SNMP Device** and **snmptraps** that "advanced_user" can configure.

Suddenly we notice that we have granted configuration access to the host **snmptraps** this way, which we consider an important host that should not be messed with and that none of our two users should have access to anyway. How can we easily restrict access to this host, while still keeping it in the **SNMP Devices** group?

In Browser 1, navigate to **Configuration | Host groups** and click on **Create Group**.

- **Group name:** Enter **Important SNMP Hosts**
- **Hosts:** Filter **Other Hosts** list box with **Group** dropdown to display **SNMP Devices**, select **snmptraps**, then click on << button

When done, click on **Save**.

Open **Administration | Users**, click on **Our Users** in the **Name** column. In the group details, click the **Add** button below the **Deny** box. In the resulting window, mark the checkbox next to **Important SNMP Hosts** and click on **Select**, then click on **Save** button.

Now is the time to look at Browser 2. It should still show host configuration with two hosts. Refresh the list and "snmptraps" host disappears. Now "advanced_user" has configuration access only to the host **SNMP Device**, and there would be no access to monitoring of "snmptraps" host as well, because we used "deny". For "monitoring_user", nothing changed—there was no access to the "SNMP Devices" group at all.

Summary

It's important to remember the main rules about permissions in Zabbix:

- Permissions can be **assigned to user groups only**
- Permissions can be **granted on host groups only**

This means that for fancy permission schemes you might have to do some planning before starting to click around. We can also safely say that, to avoid mysterious problems in future, every host should be in at least one host group and every user should be in at least one user group. Additionally, there were two factors that combined to determine effective permissions—permissions set for groups and user type. We can try summarizing the interaction of these two factors:

Zabbix user type \ Permissions	Zabbix User	Zabbix Admin	Zabbix Super Admin
Read-write	Read only	Full	Full
Read only	Read only	Read only	Full
Deny	None	None	Full

Looking at this table, we can see that the "Zabbix Super Admin" user type cannot be denied any permissions. On the other hand, the "Zabbix User" cannot be given write permissions.

With this knowledge, you should be able to group hosts, create, and group users along with assigning fine-grained permissions.

6

Acting Upon Monitored Conditions

We have gained quite a wide knowledge of what information we can gather using items. Though, so far we have only a single thing that we are actively monitoring—we have only a single trigger created (we did that in *Chapter 2*). We also have a single action to perform when this trigger fires. Both of these mechanisms provide more power than explored so far—let's try to recall what those triggers and actions did:

- Triggers define when a condition is considered worthy of attention. They "fire" (that is, become active) when item data matches a particular condition, such as too high system load or too low free disk space.
- Actions make sure something is done about a trigger firing. While it's good to see active triggers in the frontend, we'll want something done on most triggers, usually that being a notification sent in one way or other, but actions allow also other operations to be performed.

Let's explore both of these concepts in more detail now.

Triggers

Triggers are things that "fire". They are the ones that look at item data and raise a flag when the data does not fit whatever condition is defined. As we discussed before, simply gathering data is nice, but awfully inadequate. If you want anything past historical data gathering, including notifications—there would have to be a person looking at all the data all the time, so we have to define thresholds at which we want the condition to be considered worth looking into. Triggers provide a way to define what those conditions are.

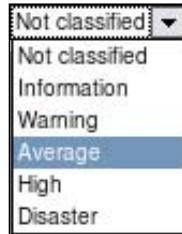
Earlier, we created a single trigger that was checking the system load on "A Test Host". It checks whether the returned value is larger than a defined threshold. Now, let's check for some other possible problems with a server—for example, when a service is down. The SMTP service going down can be significant, so we will try to look for such an event now. Navigate to **Configuration | Hosts**, choose **Triggers** in the first dropdown and click on the **Create Trigger** button. In the form that opens, we will fill in some values.

- **Name:** The contents of this field will be used to identify the trigger in most places, so it should be human-readable. This time, enter **SMTP service is down**. Notice how we are describing what the problem actually is. As opposed to an item, which is gathering statuses, a trigger has a specific condition to check, thus the name reflects it. If we had a host that should not ever have a running SMTP service, we could create a trigger named "SMTP service should not be running".
- **Expression:** This is probably the most important factor of a trigger. What is being checked, and for what conditions, will be specified here. Trigger expressions can vary from very simple to complex ones. This time we will create a simple one, and we will also use some help with that. Click the **Select** button next to the **Expression** field to open expression building dialog. It has several fields to fill as well, so let's look at what those are.
 - **Item:** Here, we can specify which item data should be checked. To do that click on the **Select** button. Another pop up opens. Select **Linux servers** in the **Group** dropdown, then select **Another Host** in the **Host** dropdown. We are interested in the SMTP service, so click on **SMTP server status** in the **Description** column. Pop up will close, and **Item** field will be populated with the chosen name.
 - **Function:** Here we can choose the actual check to be performed. Maybe we can try remembering what the SMTP server status item values were—right, "1" was for server running, and "0" was for server down. If we want to check whenever the last value is 0, the default function seems to fit quite nicely, so we won't change it.
 - **N:** This field allows us to set the constant used in the function above. We want to find out whenever server goes down (or status is "0"), so here the default fits as well.



With the values set as above, click the **Insert** button. **Expression** field is now populated with a trigger expression `{Another Host:smtp.last(0)}=0`.

- **Severity:** There are five severity levels in Zabbix, and an additional "Not classified" severity.



We will consider this problem to be of an average severity, so choose **Average** from the dropdown.

Name	SMTP service is down
Expression (Toggle input method)	<code>{Another Host:smtp.last(0)}=0</code>
The trigger depends on	No dependencies defined
New dependency	Add
Event generation	Normal
Severity	Average
Comments	(empty text area)
URL	(empty text area)
Disabled	<input type="checkbox"/>
<input type="button" value="Save"/> <input type="button" value="Cancel"/>	

Before continuing, make sure the SMTP server is running on "Another Host", then click **Save**. Let's find out how it looks in the overview now – open **Monitoring | Overview** and make sure **Type** dropdown has **Triggers** selected.

Triggers	
CPU Load too high on Test Host for last 3 minutes	A Test Host
SMTP service is down	Another Host

Great, we can see both hosts now have a trigger defined. As the triggers differ, we also have two unused cells. A newly added trigger will be flashing, thus indicating a recent change.

Let's look at the trigger expression in more detail. It starts with an opening curly brace, and the first parameter is the hostname. Separated with a colon is the item key – `smtp` here. After the dot comes the more interesting and trigger specific thing – the trigger function. Used here is one of the most common functions, `last`. It always returns single value from the item history. Here it also has a parameter passed, `0`, enclosed in parenthesis. For this particular function, a parameter passed with such syntax is ignored, while it could mean seconds, that would not make much sense (the function returns a single value only). Still, a parameter has to be provided even for functions that ignore it.

But that's not the only parameter syntax this function supports – if the value is prefixed with a hash, it is not ignored. In that case it works like an *Nth* value specifier. For example, `last (#9)` would retrieve the 9th most recent value. As we can see, `last (#1)` is equal to `last (0)`. Another overlapping function is `prev`. As the name might suggest, it returns the previous value, thus `prev (0)` is the same as `last (#2)`.

Continuing with the trigger expression, curly braces close to represent a string that retrieves some value. Then we have an operator, which in this case is a simple equal sign. Comparison is done with a constant number, zero.

Trigger dependencies

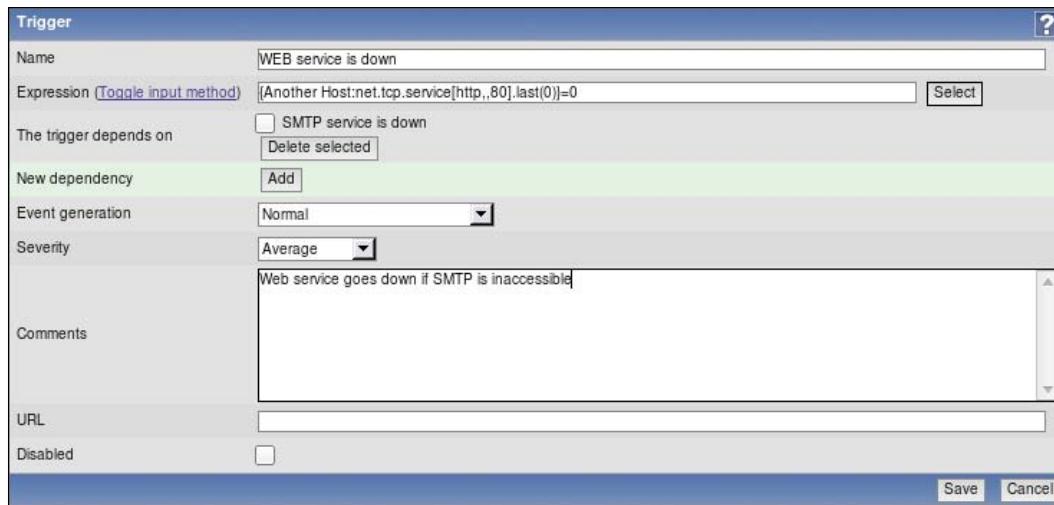
We now have one service being watched. Though there are some more monitored and now we can try to create a trigger for a HTTP server. Go to **Configuration | Hosts**, click on **Triggers** next to **Another Host**, then click on **Create Trigger**. Fill in the following values:

- **Name:** Enter **WEB service is down**.
- **Expression:** Click on **Select**, then again on **Select** next to the **Item** field. Make sure **Linux servers** is selected in the **Group** field and **Another Host** in the host field, then click on **WEB server status** in the **Description** column. Both function and its parameter are fine, so click on **Insert**.



That inserts the expression `{Another Host:net.tcp.service[http,,80].last(0)}=0`.

- **The trigger depends on:** Our host runs software that is a bit weird—the web service is a web e-mail frontend, and it goes down whenever the SMTP server is unavailable. This means the web service depends on SMTP service. To configure that, click on **Add** next to the **New dependency**. In the resulting window, make sure **Linux servers** is selected in the **Group** dropdown and **Another Host** is selected in the **Host** dropdown, then click on the only entry in the **Description** column—**SMTP service is down**.
- **Severity:** Select **Average**.
- **Comments:** Trigger expressions can get very complex. Sometimes the complexity can make it impossible to understand what a trigger is supposed to do without serious dissection. Comments provide a way to help somebody else, or yourself, to understand the thinking behind such complex triggers later. While our trigger still is very simple, we might want to explain the reason for the dependency, so enter something like **Web service goes down if SMTP is inaccessible**.



When you are done, click **Save**. Notice how, in the trigger list, trigger dependencies are listed in the **Name** column. This allows for a quick overview of any dependent triggers without opening the details of each trigger individually.

	Severity	Status	Name	Expression	Error
<input type="checkbox"/>	Average	Enabled	SMTP service is down	<code>{Another Host:smtp.last(0)}=0</code>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	Average	Enabled	WEB service is down Depends on : Another Host : SMTP service is down	<code>{Another Host:net.tcp.service[http,,80].last(0)}=0</code>	<input checked="" type="checkbox"/>



Item name colors in the **Expression** column match their state – green for OK, red for disabled, and grey for unsupported.

With the dependency set up, let's find out whether it changes anything in the frontend. Navigate to **Monitoring | Overview**.

Triggers	A Test Host	Another Host
CPU Load too high on Test Host for last 3 minutes	green	
SMTP service is down		↑
WEB service is down		↓

Indeed, the difference is visible immediately. Triggers involved in the dependency have arrows drawn over them. So an upwards arrow means something depends on this trigger or was it the other way around? Luckily, you don't have to memorize that. Move the mouse cursor over the **SMTP service is down** trigger for **Another Host**, the upper cell with the arrow.



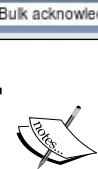
A pop up appears, informing you that there are other triggers dependent on this one. Dependent triggers are listed in the pop up. Now move the mouse cursor one cell below, over the downwards pointing arrow.



Let's see what effect other than the arrows does this provide. Open **Monitoring | Triggers** and make sure both **Host** and **Group** dropdowns say **all**, then bring down web server on "Another Host". Wait for the trigger to fire, look at the entry. Notice how an arrow indicating dependency is displayed here as well. Move the mouse cursor over it again, and the dependency details are displayed in a pop up.

Severity	Status	Last change	Age	Acknowledged	Host	Name	Comments
Average	PROBLEM	17 Feb 2010 11:57:24	1h 39s	Acknowledge (1)	Another Host	WEB service is down	Show

Hey, what's up with the **Show** link in the **Comments** column? Let's find out—click on it. As can be seen, the comment we provided when creating the trigger is displayed. This allows for easy access to comments from the trigger list both for finding out more information about the trigger and updating the comment as well. Click on **Cancel** to return to the trigger list. Now, stop the SMTP service on the "Another Host". Wait for the trigger list to update and look at it again. The web server trigger has disappeared from the list, and is replaced by the SMTP server one. That's because Zabbix does not show dependent triggers if the dependency upstream trigger is active. This helps to keep the list short and concentrate on the problems that actually cause other downtime.

<input type="checkbox"/>	Severity	Status	Last change	Age	Acknowledged	Host	Name	Comments
<input type="checkbox"/>	Average	PROBLEM	17 Feb 2010 13:03:28	1h 30m 36s	Acknowledge (1)	Another Host	 SMTP service is down	Add
Dependent:								
 - WEB service is down								
Bulk acknowledge ▾ Go (0)								

[ There's actually a minor bug with dependent trigger displaying in Zabbix 1.8.1 – most likely, you will see both triggers instead. This problem is fixed in version 1.8.2.]

Trigger dependencies are not limited to a single level. We will now add another trigger to the mix. Before we do that, we'll also create an item that will allow an easy manual condition change without affecting system services. In the frontend, navigate to **Configuration | Hosts**, click on **Items** next to **Another Host**, then click on **Create Item**. Fill in the following values:

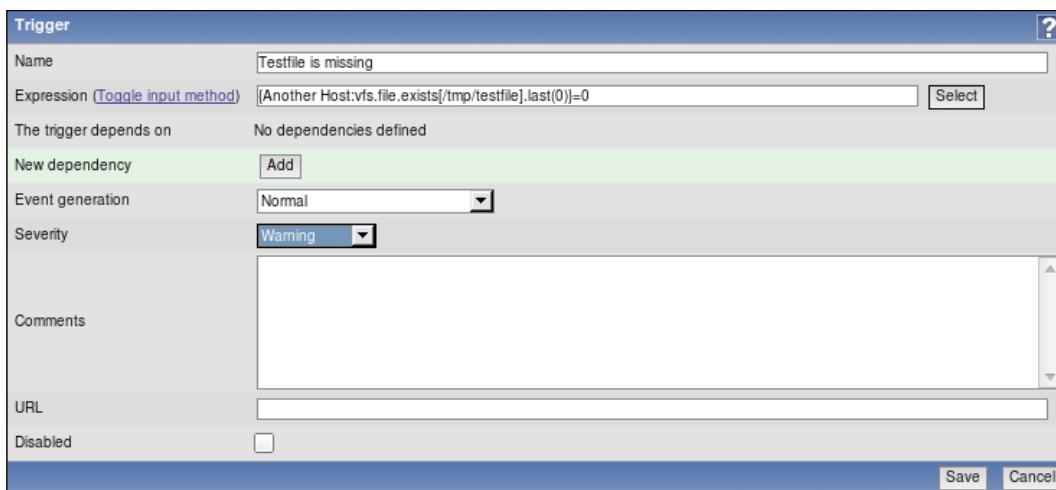
- **Description:** Enter **Testfile exists**
- **Key:** Enter `vfs.file.exists[/tmp/testfile]`.

When you are done, click **Save**. As the key might reveal, this item simply checks whether a particular file exists and returns 1 if it does, 0 if it does not.

[ Using a constant filename in /tmp in real life situations might not be desirable, as any user could create such a file.]

In the bar above the item list, click on **Triggers**, then click on **Create Trigger** button. Enter these values:

- **Name:** Enter **Testfile is missing**.
- **Expression:** Click on **Select**, then again **Select** next to the **Item** field. In the item list for **Another Host**, click on **Testfile exists** in the **Description** column, then click on **Insert** (again, the default condition works for us). The **Expression** field is filled with the following expression: `{Another Host: vfs.file.exists[/tmp/testfile].last(0)}=0`.
- **Severity:** Select **Warning**.



When you are done, click **Save**. Let's complicate the trigger chain a bit now. Click on the **SMTP service is down** trigger in the **Name** column, then click on **Add** next to the **New dependency** entry. In the upcoming dialog, click on the **Testfile is missing** entry in the **Name** column. This creates a new dependency for the SMTP service trigger.

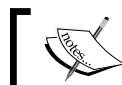


Click **Save**. Now we have created a dependency chain, consisting of three triggers. "WEB service is down" depends on "SMTP service is down", which in turn depends on "Testfile is missing". Zabbix calculates chained dependencies, so all upstream dependencies are also taken into account when determining the state of a particular trigger – in this case, "WEB service is down" depends on both the other triggers. With Zabbix versions 1.8.2 and latter, this will mean only single trigger being displayed

in **Monitoring | Triggers** section. Now we should get to fixing the problems the monitoring system has identified. Let's start with the one at the top of the dependency chain—the missing file problem. On "Another Host", execute:

```
$ touch /tmp/testfile
```

This should deal with the only trigger currently on the trigger list. Wait for the trigger list to update. You will see two triggers, with their status flashing.



Again, with version 1.8.1 you will see all triggers. This problem has been corrected in 1.8.2.



Remember, by default Zabbix shows triggers that have recently changed state flashing, and that includes also triggers in the "OK" state.

	Severity	Status	Last change	Age	Acknowledged	Host	Name	Comments
<input type="checkbox"/>	Average	PROBLEM	17 Feb 2010 14:48:28	5m 47s	Acknowledge (1)	Another Host	SMTP service is down	Add
<input type="checkbox"/>	Warning	OK	17 Feb 2010 14:47:45	6m 30s	Acknowledge (1)	Another Host	Testfile is missing	Add

Looking at the list, we see one large difference this time—the SMTP trigger now has two arrows, one pointing up, and the other pointing down. Moving your mouse cursor over them you will discover that they denote the same thing as before—the triggers that this particular trigger depends on or that depend on this trigger. If a trigger is in the middle of the dependency chain, two arrows will appear.

Our testfile trigger worked as expected for the chained dependencies, so we can remove it now. Open **Configuration | Hosts**, click on **Triggers** next to **Another Host** and click on the **SMTP service is down** trigger in the **Name** column. Mark the checkbox next to the **Testfile is missing** entry in the dependencies list, then click **Delete selected** button. Now click the **Save** button. Note that you always have to save your changes for the editing form of any entity. In this case, simply removing the dependency would not be enough. If we navigate to some other section without explicitly saving the changes, any modifications will be lost. Now you can also restart any stopped services on "Another Host".

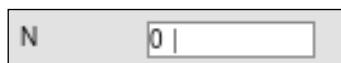
Constructing trigger expressions

So far we have used only very simple trigger expressions, comparing the last value to some constant. Fortunately, that's not all trigger expressions can do. We will now try to create a slightly more complex check.

Let's say we have two servers, "A Test Host" and "Another Host", providing a redundant SFTP service. We would be interested in any one of the services going down. Navigate to **Configuration | Hosts** and click on **Triggers** next to either **A Test Host** or **Another Host**, then click on the **Create Trigger** button. Enter these values:

- **Name:** Enter **One SSH service is down**.
- **Expression:** Click on the **Select** button. In the resulting pop up, click **Select** next to the **Item** field. Make sure **Another Host** is selected in the **Host** dropdown, click on **SSH server status** item in the **Description** column, then click **Insert**.

Now, position the cursor at the end of the inserted expression and enter "**|**" without quotes (that's space, the vertical pipe character, space). Again, click on the **Select** button. In the resulting pop up, click **Select** next to the **Item** field. Select **A Test Host** in the **Host** dropdown, click on **SSH server status** item in the **Description** column. Before inserting the expression, take a look at the **N** field, though.



Notice how there's a space and vertical pipe added. Zabbix tried to fill in values as in our first expression and interpreted the previous change as the value to compare to. That's surely incorrect, so remove these additions so there is only a **0** in that field, then click **Insert**.

- **Severity:** Select **Average** (remember, these are redundant services).

The screenshot shows the 'Trigger' configuration dialog box. The 'Name' field is set to 'One SSH service is down'. The 'Expression' field contains the formula '[Another Host:net.tcp.service[ssh].last(0)=0] | [A Test Host:net.tcp.service[ssh].last(0)=0]'. The 'Severity' dropdown is set to 'Average'. The 'Save' and 'Cancel' buttons are visible at the bottom right.

When you are done, click **Save**.

The process we did with the expression (the insertion of two expressions) allowed us to create a more complex expression than simply comparing the value of a single item. Instead, two values are compared, and the trigger fires if any one of them matches the condition. That's what the | (vertical pipe character) is for. It works as a logical OR, allowing a single matched condition to fire the trigger. Another logical operator is &, which works like a logical AND. Using the SSH server example trigger, we could create another trigger that would fire whenever both SSH instances go down. Getting the expression is simple, as we just have to modify single symbol, that is change | to &, so that expression looks like this:

```
{Another Host:net.tcp.service[ssh].last(0)}=0 & {A Test Host:net.tcp.service[ssh].last(0)}=0
```

Trigger expressions also support other operators. In all the triggers we created, we used the most common one—the equality operator =. We could also be using a non-equality check -#. That would allow us to reverse the expression like this:

```
{A Test Host:net.tcp.service[ssh].last(0)}#1
```

While not useful in these cases, such a check is helpful when the item can have many values and we want it to fire whenever the value isn't the expected one.

Trigger expressions also support the standard mathematical operators +, -, *, /, <, and > so complex calculations and comparisons can be used between item data and constants.

With the service checks we wrote, triggers would fire right away as soon as the service goes down for a single check. That can be undesirable if we know some software will be down for a moment during an upgrade, because of log rotation or backup requirements. We can use a different function to achieve delayed reaction in such cases. Replacing function last with max allows to specify a parameter, and thus react only when the problem has been active for some time. For the trigger to fire only when service has not responded for 5 minutes, we could use an expression like this:

```
{A Test Host:net.tcp.service[ssh].max(300)}=0
```

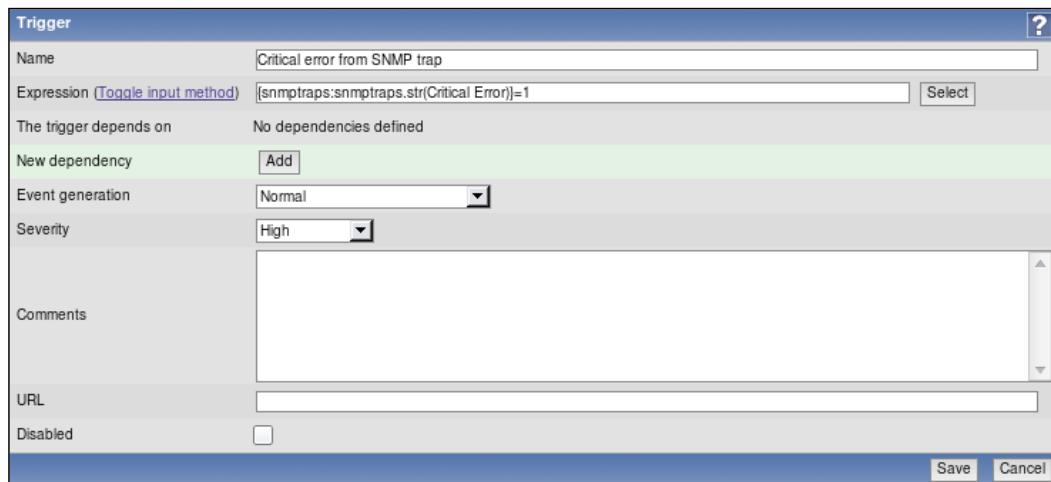
Remember, for functions that accept seconds as a parameter, we can also use the count of returned values by prefixing the number with #:

```
{A Test Host:net.tcp.service[ssh].max(#5)}=0
```

In this case trigger would always check last five returned values. Such an approach allows the trigger period to scale along in case item interval is changed, but it should not be used for items that can stop sending in data.

Let's create another trigger using a different function. In frontend section **Configuration | Hosts** choose **SNMP Devices** in the **Group** dropdown, click on **Triggers** next to **snmptraps** and click on the **Create Trigger** button, then enter these values:

- **Name:** Enter **Critical error from SNMP trap**
- **Expression:** Enter `{snmptraps:snmptraps.str(Critical Error)}=1`
- **Severity:** Choose **High**



When you are done, click **Save**.

This time we used another function, `str`. It searches for the specified string in the item data, and returns 1 if found. The match is case sensitive.

This trigger will change into the OK state whenever the last entry for the item does not contain the string specified as the parameter. If we would want to force this trigger to the OK state manually, we could just send a trap that would not contain string the trigger is looking for. A trick like that can also be useful when some other system is sending SNMP traps. In a case when the enabling trap is received successfully, but the disabling trap is lost (because of network connectivity issues, or for any other reason), you might want to use such a fake trap to disable the trigger in question, though in that case you might have to use `zabbix_sender` so that you can fake the host.

Triggers that time out

There are systems that send a trap upon failure, but no recovery trap. In such a case, manually resetting isn't an option. Fortunately, we can construct a trigger expression that times out by using another function—`nodata`. This function kicks in when item has received no data for the time specified as the parameter, so the expression that would time out after 10 minutes looks like this:

```
{snmptraps:snmptraps.str(Critical Error)}=1 & {snmptraps:snmptraps.nodata(600)}=0
```

For now we will want to have a more precise control over how this trigger fires, so we won't change the trigger expression.

Human-readable constants

Using plain numeric constants is fine while we deal with small values. When an item collects data that is larger, like disk space or network traffic, such an approach becomes very tedious. You have to calculate desired value, and from looking at it later it is usually not obvious how large it really is. To help here, Zabbix supports so-called suffix multipliers in expressions—abbreviations `K`, `M`, and `G` are supported. That results in shorter and way more easier-to-read trigger expressions. For example, checking a non-existent host disk space goes from:

```
{host:vfs.fs.size[/,free].last(0)}<16106127360
```

to:

```
{host:vfs.fs.size[/,free].last(0)}<15G
```

That's surely easier to read and modify if such a need arises.

We have now covered the basics of triggers in Zabbix. There are many more functions, allowing evaluation of various conditions, that you will want to use later on. Frontend function selector does not contain all of them, so sometimes you will have to look them up and construct expression manually. For a full and up to date function list, refer to the official documentation.

Event details

After we have configured triggers, they generate events, which in turn are acted upon by actions.



We looked at a high-level schema of information flow inside Zabbix, including item, trigger and, event relationship in *Chapter 2*.

But can we see more details about them somewhere? In the frontend, open **Monitoring | Events** and click on date and time in the **Time** column for the latest entry with **PROBLEM** status.



If you see no events listed, expand filter, set **Events since** to some time in the past, then click **Filter**.



This opens up the event details window, which allows to determine with more confidence event flow. It includes things such as event and trigger details and action history. The event list that includes previous 20 events itself acts as a control, allowing you to click any of these events and see previous 20 events from the chosen event on. As this list only shows events for a single trigger, it is very handy if one needs to figure out the timeline of one, isolated problem.

Event Source Details		Acknowledges													
Host	Another Host	Time	User	Comments											
Trigger	SMTP service is down	...													
Severity	Average														
Expression	(Another Host:smtp.last(0)=0)														
Event generation	Normal														
Disabled	No														
Event details		Message actions													
Event	SMTP service is down	Time	Type	Status	Retries left	Recipient(s)	Message	Error							
Time	2010.Feb.17 19:08:28	Email	sent			user@company.tld	Subject: SMTP service is down: PROBLEM Message: SMTP service is down: PROBLEM								
Status	PROBLEM														
Duration	3m														
Acknowledged	-														
Command actions															
Time	Status	Command				Error									
No actions found															
Events List [Previous 20]															
Time	Status	Duration	Age	Ack	Actions										
2010.Feb.17 19:08:28	PROBLEM		12m 30s	No	1										
2010.Feb.17 19:03:28	OK	5m	17m 30s	No	-										

Event generation and hysteresis

Events are generated whenever a trigger changes state. That's simple, right? But not all trigger state changes can be trapped by an action. A trigger can be in one of the following states:

- **OK** – Normal state when trigger expression evaluates to false. Can be acted upon by actions.
- **PROBLEM** – A problem state when trigger expression evaluates to true. Can be acted upon by actions.
- **UNKNOWN** – A state when Zabbix cannot evaluate trigger expression, usually when there is missing data or trigger has been just modified. Can not be acted upon.

No matter whether the trigger goes from OK to PROBLEM, or UNKNOWN or any other direction, an event is generated.

We found out before that we can use some trigger functions to avoid changing trigger state upon every change in data. By accepting a time period as a parameter these functions allow us to only react if a problem has been going on for a while. But what if we would like to be notified as soon as possible, while still avoiding trigger flapping if values fluctuate near our threshold? Here a specific Zabbix macro helps and allows to construct trigger expressions that have some sort of hysteresis—remembering of a state.

A common case is measuring temperatures. For example, a very simple trigger expression would read:

```
A Test Host:temp.last(0)>20
```

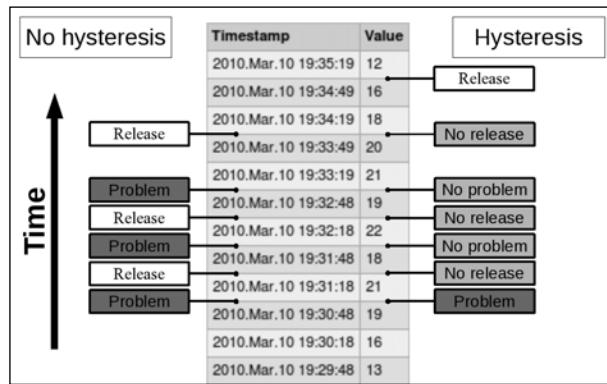
It would fire when the temperature was 21, and go to the OK state when it's 20 and so on. Sometimes temperature fluctuates around the set threshold value, thus trigger goes on and off all the time. That is undesirable, so an improved expression would look like:

```
({{TRIGGER.VALUE}}=0&{{server:temp.last(0)}}>20) | ({{TRIGGER.VALUE}}=1&{{server:temp.last(0)}}>15)
```

A new macro, TRIGGER.VALUE is used. If the trigger is in the OK state it is 0, if trigger is in the PROBLEM state it is 1. Using the logical operator | (OR), we are stating that this trigger should change to (or remain at) PROBLEM state if:

- Trigger is currently in the OK state and the temperature exceeds 20 or when
- Trigger is currently in PROBLEM state and temperature exceeds 15

How does that change the situation when compared to the simple expression that only checked for temperatures over 20 degrees?



In this example case we have avoided two unnecessary PROBLEM states, and usually that means at least two notifications as well.

Actions

The trigger list would be fine to look at, way better than looking at individual items but that would still be an awful lot of manual work. That's where actions come in, providing notifications and other methods to react upon condition change.

The most common method is e-mail sending. If you had an action set up properly when we first configured a fully working chain of item-trigger-action, you received an e-mail whenever we started or stopped some service, created the test file, and so on. Let's look at what actions can do in more detail.

Limiting conditions when actions are sent

Our previous action, created in *Chapter 2*, matched any event as we had not limited its scope in any way. Now we could try matching only some specific condition. Navigate to **Configuration | Actions**, then click on **Create Action**.

In the **Name** field, enter **SNMP Action**.

In the **Action conditions** block at the bottom of the page, click on the **New** button. This will open a new block, **New condition**. Here, select **Host group** in the first dropdown and click on the **Select** button. In the pop up, click on **Important SNMP Hosts**, then click **Add**.

Now we have to define operations. Operations are actual actions that are performed. Click on **New** in the **Action operations** block, then click on **Select** in the **Edit operation** block that appears. In the upcoming window click **Our Users**, then click **Add**. The result should look like this:

The screenshot shows the Zabbix Action configuration interface. On the left, the 'Action' tab is active, displaying fields for Name (SNMP Action), Event source (Triggers), Enable escalations (unchecked), Default subject ([TRIGGER.NAME]: [STATUS]), Default message ([TRIGGER.NAME]: [STATUS]), Recovery message (unchecked), and Status (Enabled). At the bottom of this tab is the 'Action conditions' section with a condition: 'Conditions (A) Host group = "Important SNMP Hosts"'. On the right, the 'Action operations' tab is active, showing a list of operations: 'Send message to Group "Our Users"' with an 'Edit' button. Below this are 'New' and 'Delete selected' buttons. The overall interface has a blue and white color scheme with a sidebar on the left.

Finally, click **Save** in the **Action** block. As we want to properly test how e-mails are sent, we should now disable our previously added action. Mark the checkbox next to **Test Action**, choose **Disable selected** in the **Action** dropdown and click **Go**, then confirm disabling in the pop up.



The following activities rely on correctly configured e-mail setup (done in *Chapter 2*) and group "Our Users" existing (added in *Chapter 5*).



There's still one missing link – none of the two users in the "Our Users" group has media defined. To add media, navigate to **Administration | Users**, select **Users** in the first dropdown and click on **monitoring_user** in the **Alias** column. Click **Add** next to the **Media** section, enter e-mail address in the **Send to** field, then close the pop up by clicking **Add**. We now have to properly save this change as well, so click **Save**.

Now we have to make trigger fire, so execute from the Zabbix server following:

```
$ snmptrap -Ci -v 2c -c public localhost "" "NET-SNMP-MIB::netSnmpExperimental" NET-SNMP-MIB::netSnmpExperimental s "Critical Error"
```

Notice how we have entered the matching string with proper capitalization. You can verify that trigger fires in the **Monitoring | Triggers** section.



We had to send this SNMP trap from Zabbix server or any other machine but "Another Host" – that one had `snmptraps` item created, thus trap would not have landed in the `snmptraps` host as we wanted it to.



Now we can wait for the e-mail to arrive. While we do that, let's check out another view – event view. Open **Monitoring | Events**, take a look at the last few entries.

Time	Host	Description	Status	Severity	Duration	Ack	Actions
2010.Feb.18 14:23:07	snmptraps	Critical error from SNMP trap	PROBLEM	High	1m 42s	No	-
2010.Feb.17 19:11:28	Another Host	SMTP service is down	OK	Average	19h 13m 21s	No	Ok



If you don't see the SNMP event, make sure that both **Group** and **Host** dropdowns have **all** selected.



We can see that two events have been successfully registered – the SMTP service returning to a normal state, and our SNMP trap reporting an error. But the last column, titled **Actions**, is notably different. While the SMTP event has **Ok** listed, the SNMP one has just a dash. So here's why:

In Zabbix, **only users that have at least read-only access to at least one of the systems, referenced in the trigger, receive notifications.**

That allows to overlap host group permissions with action conditions to create quite sophisticated notification scenarios.

Additional action conditions

So far we have used only single action condition. Actually, Zabbix provides quite a lot of different conditions that determine when action is invoked. We added a condition based on **host group**. Let's look at some examples of what other conditions are available:

- **Host:** Similar to the host group condition, this one allows to single out some important (or unimportant) host for action invocation
- **Trigger:** This condition allows to match individual problems
- **Trigger description:** A bit more flexible than the previous one, with this condition we can limit invocation based on trigger name – for example, only acting upon triggers that have string "database" in their names
- **Trigger severity:** We can limit the action to only happen for the highest two trigger severities or maybe only for a couple of lowest severities
- **Time period:** Operations can be carried out only if a problem has happened in a specified time period or they can be suppressed for a specified time period instead

There are more action conditions that are useful in specific use cases.

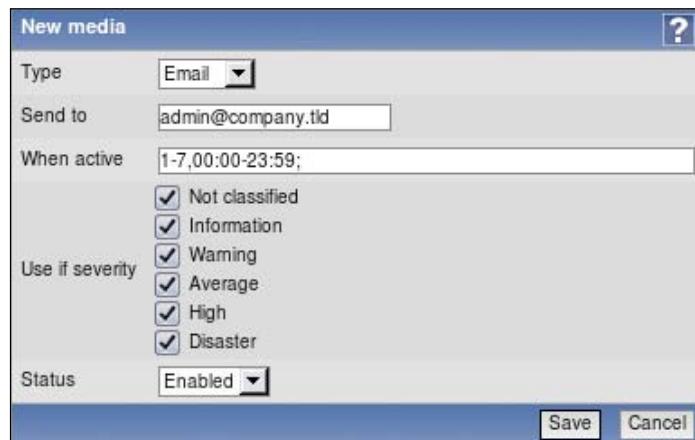
Dependencies and actions

Another way to limit notifications sent is trigger dependencies, which comes real handy here. If a trigger that is dependent on an already active trigger fires, we have seen the effect on the frontend – dependent trigger did not appear in the list of active triggers. This is even better with actions – no action is performed in such a case. If you know that a website relies on NFS server, and have set a corresponding dependency, NFS server going down would not notify you about website problem. When there's a problem to solve, not being flooded with e-mails is a good thing.

There's a possible race condition if the dependent item is checked more often. In such a case the dependent trigger might fire first, and the other one a short time later, thus still producing two alerts. While not a huge problem for the trigger displaying in the frontend, this can be undesirable to happen with actions involved. If you see such false positives often, change the item intervals so that the dependent one always has a slightly longer interval.

Per media limits

We looked at what limits an action can impose, but there are also possible limits per media. Navigate to **Administration | Users**, then find **Admin** in the **Members** column and click on **Edit** next to the only media we have created here.



When considering limits, we are mostly interested in two sections here – **When active** and **Use if severity**.

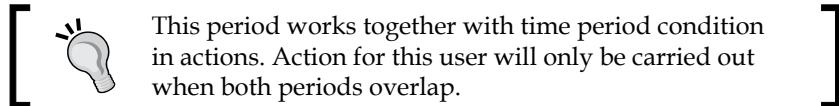
As the name indicates, the first of these allows to set a period when media is used. Days are represented with numbers 1-7 and 24-hour clock notation of HH:MM-HH:MM is used. Several periods can be combined, separated by semicolons. This way it is possible to send an SMS to a technician during weekends and nights, e-mail during workdays, and e-mail to helpdesk during working hours.



In case you are wondering, week starts with Monday.

For example, a media active period like this might be useful for an employee who has different working time during a week:

1-3,09:00-13:00;4-5,13:00-17:00



This period works together with time period condition in actions. Action for this user will only be carried out when both periods overlap.

Use if severity is very useful as well, for that poor technician might not want to receive informational SMS messages during the night, only disaster ones.

Click **Cancel** to close this window.

Sending out notifications

As both of the users specified in the action operations, had explicitly been denied the access to `snmptraps` host, they were not considered valid for action operations.

Let's give them access to this host now. Go to **Administration | Users** and click on **Our Users** in the **Name** column. Mark **Important SNMP Hosts** in the **Deny** box, click **Delete selected** below, then click **Save**. Both users should have access to the desired host now, but our trigger stays in the active state—that's because the last entry contains string `Critical Error`. We want to make this trigger fire again, so we have to deactivate it. As discussed before, we can do that by sending another trap without the string trigger is looking for. From Zabbix server, execute:

```
$ snmptrap -Ci -v 2c -c public localhost "" "NET-SNMP-MIB::netSnmpExperimental" NET-SNMP-MIB::netSnmpExperimental s "some other string"
```

The trigger will now deactivate. Activate it again by executing on the Zabbix server:

```
$ snmptrap -Ci -v 2c -c public localhost "" "NET-SNMP-MIB::netSnmpExperimental" NET-SNMP-MIB::netSnmpExperimental s "Critical Error"
```

Wait for a while so that trigger fires. Check your e-mail, and now you should have received one notification. Let's see the event list again—open **Monitoring | Events**, look at the latest entry.

Time ▾	Host	Description	Status	Severity	Duration	Ack	Actions
2010.Feb.18 14:50:56	snmptraps	Critical error from SNMP trap	PROBLEM	High	1m 55s	No	1 1

Oh, but what's up with the weird entry in the **Actions** column? Those two differently colored numbers look quite cryptic. Let's try to find out what they could mean—open **Administration | Actions**, then select **Audit actions** from the dropdown and look at the last few entries.

Time	Type	Status	Retries left	Recipient(s)	Message	Error
18 Feb 2010 19:34:28	Email	sent		monitoring_user@company.tld	Subject: Critical error from SNMP trap on snmptraps: PROBLEM Message: Critical error from SNMP trap: PROBLEM Last value: 127.0.0.1 "Critical Error" NET-SNMP-MIB::netSnmpExperimental	
18 Feb 2010 19:34:28		not sent	0		Subject: Critical error from SNMP trap on snmptraps: PROBLEM Message: Critical error from SNMP trap: PROBLEM Last value: 127.0.0.1 "Critical Error" NET-SNMP-MIB::netSnmpExperimental	No media defined for user "advanced user (advanced_user)"*



If you don't see any entries, expand the filter and change **Actions since** to some time in the past, then click **Filter**.

Excellent, the audit log clearly explains what the error is, and we can easily deduct that number in the event list represent notification counts—green for successful ones and red for unsuccessful ones. It also shows us that actions should not be configured to send messages for users that do not have media correctly set as such entries pollute the action log and make it harder to review interesting entries.

Using macros

Let's take a careful look at the e-mails we received (if you have already deleted them, just send a couple more SNMP traps). The subject and body both have content like Critical error from SNMP trap: OK. There's one huge problem here—we have no idea which host generated the alarm. While there's another solution we will explore right now, a general suggestion is to **always include host name in trigger name**. Doing so will avoid cases when you receive an alert, but have no idea which host has the problem.

Another solution is possible for the aforementioned problem—we can use something called **macro** to help in this particular case. To proceed, navigate to **Configuration | Actions**, click on **SNMP Action** in the **Name** column, then change **Default subject** field contents to:

```
{TRIGGER.NAME} on {HOSTNAME}: {STATUS}
```

The field already contained two macros—`{TRIGGER.NAME}` and `{STATUS}`. The benefit from a macro is evident when we have a single action covering many cases. We don't have to create a myriad of actions to cover every possible situation, instead we use macros to have desired information, related to the particular event, replaced. You can think of Zabbix macros as variables. Macro names usually provide a good idea on what a macro does. In this case we improved existing the subject line which already contained trigger name and status macros by adding the host name macro, though it is still recommended to include the host name in trigger names.

While we are here, let's improve message itself a bit. Change **Default message** field to read:

```
{TRIGGER.NAME}: {STATUS}  
Last value: {ITEM.LASTVALUE}
```

Note that we are adding that on a new line—multiline mail bodies are supported. Yet another macro is used here, `{ITEM.LASTVALUE}`. Again, the name is very descriptive and this macro should insert last value of the item in question.



In case when there are multiple items used in trigger expression, this macro will return last value of the first item.



To confirm your changes, click **Save**. Make the trigger change state a couple of times by sending SNMP traps like before, then check your e-mail. Not only does the subject include the hostname, now the body also shows the last value that the item had, for example:

```
Last value: 127.0.0.1 "some other string"    NET-SNMP-MIB::  
netSnmpExperimental
```

Sometimes a receiver might benefit from additional information, not directly obtainable from event related macros. Here additional class of macros helps—the ones used in trigger expressions also works for macro contents. Imagine a person, managing two servers that both rely on an NFS server, which is known to have performance problems. If the system load on one of these two servers increases to fire a trigger, alert receiver would want to know load on the second server as well, and also whether NFS service is running correctly. That would allow to do a quick evaluation where the problem most likely lies—if NFS service is down or is having performance problems of its own, then system load on these two servers most likely has risen because of that, and NFS server admin will have to take care of that. For this person to receive such information we can add lines like these to the e-mail body:

```
CPU load on Another Host: {Another Host:system.cpu.load.last(0)}  
NFS service is: {NFS Server:nfs.service.last(0)}
```

Note, there is no built-in NFS service item.

As can be seen in the example, same syntax is used as in trigger expressions, including functions supported. This also allows the receiver to be immediately informed about average load over some period of time by adding a macro like:

```
Average CPU load on Another Host for last 10 minutes: {Another Host:  
system.cpu.load.avg(600)}
```

You can find a full list of supported macros in the official Zabbix documentation.

Escalating things

We know how to perform some action if a condition is reached, like the temperature being too high, the available disk space being too low or web server not working. We can send a message, open a ticket in some tracker, run custom script, or execute a command on the remote machine. But all these are simple if-then sequences – if it's this problem, do this, and so on. Quite often the severity of the problem depends on how long the problem persists. For example, a couple of minute's connection loss to a branch office might not be critical, but still worth noting down and e-mailing IT staff. The inability to reach branch office for ten minutes would be quite important, and at this point we would like to open a ticket in the helpdesk system and send SMS to IT staff. After 20 minutes of silence we would e-mail IT boss. Let's look at what tools Zabbix provides to enable such gradual activities.

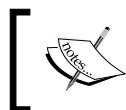
In the frontend, navigate to **Configuration | Actions** and click on **Disabled** next to the **Test Action** in the **Status** column, then click on **Test Action** in the **Name** column. Currently this action sends single e-mail to user **Admin** whenever a problem occurs. Let's complicate this situation.

- Our first user, **Admin**, would be notified every minute for the first five minutes, and every five minutes after that until the problem is resolved.
- **advanced_user** would be lower-level management, who would like to receive notification if a problem is not resolved in five minutes
- **monitoring_user** would be higher-level manager, who should be notified in 20 minutes, if the problem still is not resolved, and if it is not yet acknowledged.

While these times would be longer in real life, here we are interested in seeing escalation in action. Looking at the action details, there's a checkbox, conveniently named **Enable escalations**.

Enable escalations

Before we even start thinking about using escalations, there's an important change we must make. For the escalation to work properly, we must add action condition to fire only when trigger fires. Why is this important? Otherwise recovery events also would get escalated, and we would end up with lots and lots of useless notifications. So click on **New** in the **Action conditions** block, choose **Trigger value** in the first dropdown and **PROBLEM** in the last one, then click **Add**.



Always add trigger value condition when enabling escalations. Failing to do so can result in huge amount of useless messages, as OK state messages get escalated.



Well, that would limit escalations to problems occurring, but it would also limit message sending and we want to be notified also when the problem is resolved. That's why there's another checkbox in the main block, **Recovery message**. Enable the recovery message by marking this checkbox. Notice how this gives us two additional fields—we can customize recovery message. Instead of sending similar messages for problems and recoveries we can make recoveries stand out a bit more. Hey, that's a good idea—we will be sending out e-mails to management, let's add some "feel good" thing here. Modify **Recovery subject** field by adding "Resolved:" in front of the existing content.

Recovery message <input checked="" type="checkbox"/>
Recovery subject <input type="text"/> Resolved: [TRIGGER.NAME]: [STATUS]



Trigger value condition and custom recovery message can be used without escalations in cases when different recovery message is desired.



Always add trigger value condition when enabling recovery message. Failing to do so can result in recovery messages being escalated, and thus generate huge amount of useless messages.



Now we are ready to mark the **Enable escalations** checkbox, and observe **Action operations** block on the right-hand side change. In addition to the existing columns, **Details** and **Action**, three new ones appear – **Steps**, **Period**, and **Delay**. Additionally, below **Enable escalations** checkbox input field **Period** has appeared. All these input form changes allow us to view and modify action operations in more detail, adding capabilities to set escalation steps.

Looking at the operation list, we can see that it currently contains single operation only – sending of e-mail message to Admin user immediately and once only – that is indicated by the **Steps** column having only the first step listed.

Action operations					
	Steps	Details	Period (sec)	Delay	Action
	1	Send message to User "Admin"	Default	Immediately	<input type="button" value="Edit"/>

The first change we would like to perform – make sure that Admin receives notifications every minute for the first five minutes. Before we modify that though, we should change global action period, which by default is 3600. Looking at our requirements, both times when some change happens in notification requirements and time between two consequent, repeating notifications, user Admin will receive after the first five minute period is five minutes. Because of this, it makes sense for us to base our action on five minute steps, or 300 seconds, so enter 300 in the **Period** field and click **Save** for the action. Again click on **Test Action** in the **Name** column to open details. Now back to making sure Admin receives a notification every minute for the first five minutes – click on **Edit** in the **Action operations** block. Operation now has several new things to configure, namely **Step** and **Conditions**.

Notice how operation details also has **Period** field. This allows to override action-level period for each operation. We have action level period of 300 seconds, but these steps should be performed with one minute between them, so enter 60 in the **Period** field for **Step** area. For operations, first step means "immediately", thus **From** in the **Step** area satisfies us. On the other hand, it currently sends the message only once, but we want to pester our administrator for the first five minutes. Including the initial message, that's six messages in total, and the last one will be sent five minutes after the problem happened. So for **Step** area, **To** field, enter 6.

The final result should look like this:

From
Step To
Period
Operation type
Send message to Admin
Send only to
User medias Email 1-7,00:00-23:59;
Default message
Conditions

If it does, click **Save** in the **Edit operation** block. Now to the next task—Admin must receive notifications every five minutes after that, and until the problem is resolved. Click **New** in the **Action operations** block. Let's configure simple things first. Choose **Single user** in **Send message to** dropdown, then click **Select** next to this field and click on **Admin** in the resulting pop up.

Now we have to figure out what values to put in the **Step** fields. We want this operation to kick in after five minutes, but five minutes is already covered by the first operation, so actually that makes it ten minutes. For this operation, we can use default period, so we'll leave **Period** field in the **Step** area at zero. Steps one to six have already passed and it took five minutes. Default period being another five minutes, we can figure out that we will need this operation to kick in one step after the first one has finished. Confusing? Let's try to create a timeline. We have a single operation currently set, which overrides default period. After that, the default period starts working, and even though we currently have no operations assigned, we can calculate when further steps would be taken.

Step	Operation	Period (seconds)	Time passed
1	Send message to User "Admin"	Operation, 60	0
2	Send message to User "Admin"	Operation, 60	1 minute
3	Send message to User "Admin"	Operation, 60	2 minutes
4	Send message to User "Admin"	Operation, 60	3 minutes
5	Send message to User "Admin"	Operation, 60	4 minutes
6	Send message to User "Admin"	Operation, 60	5 minutes
7	-none-	Default, 300	10 minutes
8	-none-	Default, 300	15 minutes



The operation period only overrides periods between steps it spans. If an operation spans steps 5-7, it overrides periods 5-6 and 6-7. Single step operation does not override period.

This makes it easier to see that what we want is step 7, so enter 7 in the **From** field. We want this operation to continue until problem is resolved, thus 0 goes in the **To** field in the **Step** area. When done, click **Add**.



If two escalation operations overlap steps and both have custom period defined, smallest period is used for the overlapping steps.

We can see that Zabbix helpfully calculated the time when the operation would start, which allows us to quickly spot errors in our calculations. There are no errors here, it's at 10 minutes as desired.

	Steps	Details	Period (sec)	Delay	Action
<input type="checkbox"/>	1 - 6	Send message to User "Admin"	60	Immediately	<button>Edit</button>
<input type="checkbox"/>	7 - 0	Send message to User "Admin"	Default	00:10:00	<button>Edit</button>

With that covered, our lower-level manager, `advanced_user`, must be notified after five minutes, but only once. That means another operation – click **New** in the **Action operations** block. Select **Single user** in the **Send message to** dropdown, then click the **Select** button next to this field. In the pop up, click on `advanced_user` in the **Name** column. The single message should be simple – we know that 6th step happens after five minutes have passed, so let's enter 6 in both **From** and **To** fields, then click **Add**. Again, the **Delay** column shows that our calculation was correct and this step would be executed after five minutes, as expected.

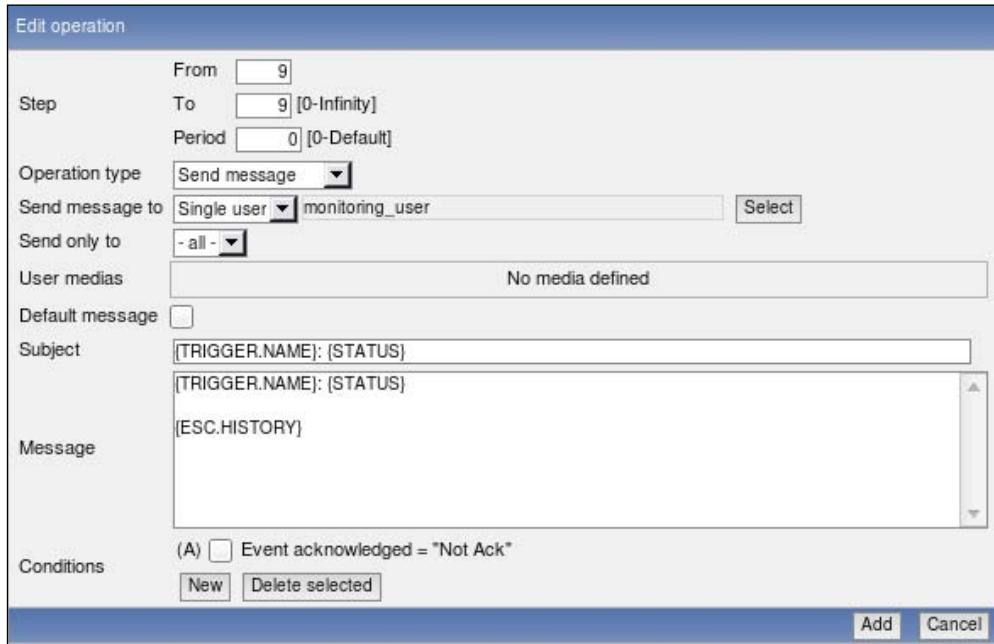
We are now left with the final task – notifying higher-level manager after 20 minutes, and only if the problem has not been acknowledged. As before, click **New** in the **Action operations** block. Select **Single user** in the **Send message to** dropdown, then click **Select** button next to this field. In the pop up, click on `monitoring_user` in the **Name** column. Let's try to continue our planned step table:

Step	Operation	Period (seconds)	Time passed
	...		
6	Send message to User "Admin"	Operation, 60	5 minutes
7	-none-	Default, 300	10 minutes
8	-none-	Default, 300	15 minutes
9	-none-	Default, 300	20 minutes

As steps just continue with the default period, this shows us that step 9 is the correct one. As we want only single notification here, enter 9 in both **From** and **To** fields.

[ It is not required to "fill" all steps with operations as we just did. Some steps in between can be skipped if planned schedule requires so. Even better, it is possible to skip first steps as well, thus sending out a delayed notification only if the problem has not been resolved.]

An additional requirement was to notify this user only if the problem is not acknowledged. To add such a restriction, click **New** in the **Conditions** area. The **New Operation condition** block is displayed, and the default setting already has **Not Ack** chosen, so click **Add** in the **New Operation condition** block. While we're almost done, there's one more bit we can do to make this notification less confusing for upper management. Currently, everybody receives the same message – trigger name, status, and last value of item that is being monitored. Last value might not be that interesting to the manager, thus we can try omitting it from those messages. Untick **Default message** checkbox and notice how we can customize subject and message for a specific operation. The custom operation message already omits technical detail (last value), but for the manager, it might also be useful to know who was notified and when. Luckily, there's another helpful macro, **{ESC.HISTORY}**. Let's modify message by adding an empty line, and then this macro. Here's what the final result for this operation should look like:



The screenshot shows the 'Edit operation' dialog box. The 'From' field is set to '9'. The 'Step' section shows 'To' as '9 [0-Infinity]' and 'Period' as '0 [0-Default]'. Under 'Operation type', 'Send message' is selected. In the 'Send message to' section, 'Single user' is chosen, and the recipient is 'monitoring_user'. There is a 'Select' button next to the recipient field. The 'Send only to' dropdown is set to '- all -'. The 'User medias' section indicates 'No media defined'. The 'Default message' checkbox is unchecked. The 'Subject' field contains '[TRIGGER.NAME]: [STATUS]' repeated twice. The 'Message' area contains '[ESC.HISTORY]'. In the 'Conditions' section, there is a condition '(A) Event acknowledged = "Not Ack"'. At the bottom, there are 'New' and 'Delete selected' buttons, and 'Add' and 'Cancel' buttons.

It's all fine, so click **Add**. We can now review action operations and verify that each operation starts when it should.

Action operations					
	Steps	Details	Period (sec)	Delay	Action
	1 - 6	Send message to User "Admin"	60	Immediately	<button>Edit</button>
	6	Send message to User "advanced_user"	Default	00:05:00	<button>Edit</button>
	7 - 0	Send message to User "Admin"	Default	00:10:00	<button>Edit</button>
	9	Send message to User "monitoring_user"	Default	00:20:00	<button>Edit</button>

New Delete selected

Everything seems to match the specification, so we can finally click **Save** in the **Action** block. With notification system in place, let's break something. On "Another Host", execute:

```
$ rm /tmp/testfile
```

It will take a short time for Zabbix to notice this problem and fire away the first e-mail to the Admin user. This e-mail won't be any different to the ones we received before. But now let's be patient and wait for some 20 minutes more. During this time, Admin user will receive more messages. But what we are really interested in is message contents to the monitoring_user. Once you receive this message, look at what it contains.

```
Testfile is missing: PROBLEM
Last value: 0

Problem started: 2009.09.25 15:11:40 Age: 41m
1. 2009.09.25 15:11:45 sent      Email admin@company.tld "Zabbix
Administrator (Admin)".
2. 2009.09.25 15:12:45 sent      Email admin@company.tld "Zabbix
Administrator (Admin)".
3. 2009.09.25 15:13:45 sent      Email admin@company.tld "Zabbix
Administrator (Admin)".
4. 2009.09.25 15:14:45 sent      Email admin@company.tld "Zabbix
Administrator (Admin)".
5. 2009.09.25 15:15:45 sent      Email admin@company.tld "Zabbix
Administrator (Admin)".
6. 2009.09.25 15:16:45 sent      Email admin@company.tld "Zabbix
Administrator (Admin)".
6. 2009.09.25 15:16:45 failed    "advanced user (advanced_user)"
No media defined for user "advanced user (advanced_user)"
7. 2009.09.25 15:21:45 sent      Email admin@company.tld "Zabbix
Administrator (Admin)".
8. 2009.09.25 15:26:45 sent      Email admin@company.tld "Zabbix
Administrator (Admin)".
```

It now contains a lot more information than just what happened – the manager has also received detailed list of who was notified of the problem. The user `Admin` has received many notifications, and then... hey, `advanced_user` has not received notification because their e-mail address is not configured. There's some work to do either for this user, or for the Zabbix administrators to fix this issue. And in this case, the issue is only escalated if nobody has acknowledged the problem before, which means nobody has even looked into it.

If we look carefully at the prefixed numbers, they are not numbers of entries in the history, it is actually escalation step. That gives us quick overview which notifications happened at the same time without comparing timestamps.

Let's fix the problem now, on "Another Host" execute:

```
$ touch /tmp/testfile
```

In a short while two e-mail messages should be sent – one to the `Admin` user, one to `monitoring_user`. As these are recovery messages, they will both have our custom subject:

```
Resolved: Testfile is missing: OK
```

Our test action had too short escalation thresholds for most real life situations. If reducing these meant creating an action from scratch, that would be very inconvenient. Though, let's see how easily we can adapt the existing one. In the frontend, navigate to **Configuration | Actions**, then click on **Test Action** in the **Name** column. While we might want to pester `Admin` user every minute for the first five minutes anyway, we might want to increase time before management is notified, as well as proportionally increase rate of messages to `Admin` user after that. In our action, only first step uses custom period **60**. This is controlled by the action level period. In the **Action** block, **Period** field, we currently have entered **300**, which in seconds is five minutes and that is the time between each two steps. Let's set it to **1200** (20 minutes) now, then click **Save** in the **Action** block. That brings us back to the action list – click on **Test Action** in the **Name** column. We are interested to see what changes this brought to the **Delay** column. Updating the action level period both pushed forward in time any escalation steps depending on it, as well as made further messages (steps from 7 onwards) sent to the `Admin` user less frequent. We could increase the interval between the first messages sent to `Admin` and make the notification sent later to `advanced_user` by increasing period for the very first operation.

Before:

	Steps	Details	Period (sec)	Delay	Action
	1 - 6	Send message to User "Admin"	60	Immediately	Edit
	6	Send message to User "advanced_user"	Default	00:05:00	Edit
	7 - 0	Send message to User "Admin"	Default	00:10:00	Edit
	9	Send message to User "monitoring_user"	Default	00:20:00	Edit

After:

Delay
Immediately
00:05:00
00:25:00
01:05:00

This allows us to easily scale notifications and escalations up from a testing configuration to something more appropriate to the actual situation, as well as adapting quickly to changing requirements. Let's create another problem. On "Another Host", execute:

```
$ rm /tmp/testfile
```

Wait for the trigger to fire and for a couple of e-mails arrive for the Admin user, then "solve" the problem:

```
$ touch /tmp/testfile
```

That should send a recovery e-mail to the Admin user soon. Hey, wait. Why for that user only? **Zabbix only sends recovery notifications to users who have received problem notifications.** As the problem did not get escalated for the management user to receive the notification, that user was not informed about resolving the problem either. A similar thing actually happened with advanced_user who did not have media assigned. As the notification was not sent even when the event was escalated (because no e-mail address was configured), Zabbix did not even try to send a recovery message to that user.

So in this case, if the Admin user resolved or acknowledged the issue before monitoring_user received e-mail about the problem, monitoring_user would receive neither the message about the problem, nor the one about resolving it.

As we can see, escalations are fairly flexible and allow you to combine many operations when responding to an event. We could imagine one fairly long and complex escalation sequence of a web server going down to proceed like:

- E-mail administrator
- Send SMS to admin
- Open report at helpdesk system
- E-mail management
- Send SMS to management
- Restart Apache

- Reboot the server
- Power cycle whole server room

Well, the last one might be a bit over the top, but we can indeed construct fine grained stepping up of reactions and notifications about problems.

Integration with issue management systems

Sending out messages to technicians or the helpdesk is nice, but there are times and conditions when it is desirable to automatically open an issue in some management system. This is most easily achieved with e-mail gateways that decent systems provide. To implement such an integration, following steps should be taken:

1. A user for ticketing system created
2. Media for this user assigned (e-mail address that system receives e-mail at)
3. Read-only access assigned for resources this user should receive alerts on (remember, no alerts are sent out if user does not have access to any of the hosts involved in the event generation)
4. Either a separate action is created, or this user is added as a recipient in an existing action operation with a custom message (by unmarking the "Default message" checkbox when editing the operation)

There's also step 5 – proper message contents should be formed so that receiving system knows what to do with the received message. This is specific to each system, but let's look at two examples of what the message contents should be set to. These examples provide only basic information – for added bonus points you can add other macros like last or average value display ones. Note that the specific syntax might change between versions, so check documentation for the version you are using.

Bugzilla

Everybody's favorite bug tracker, sometimes abused as a general issue management system. Still, Zabbix could monitor the status of software tests and open new tickets if, for example, compilation fails.

```
@{TRIGGER.NAME}
@product = <some existing product>
@component = <some existing component>
@version = 1.8
{DATE} - {TIME}
{TRIGGER.NAME}.
```

The *From* address is used to determine the user account that is creating the bug report.

CA Unicenter Service Desk

Unicenter Service Desk software from Computer Associates.

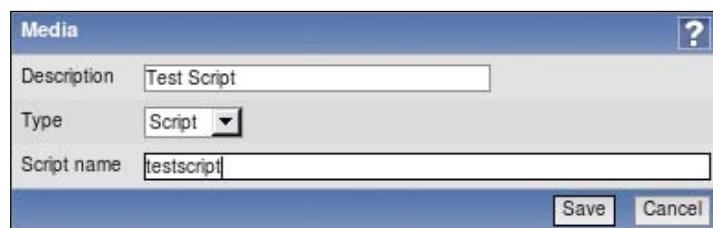
```
"start-request"
%CUSTOMER= <some existing user account>
%DESCRIPTION= {DATE} - {TIME}
{TRIGGER.NAME}.
{SUMMARY= {TRIGGER.NAME}.
%PRIORITY= {TRIGGER.NSEVERITY}
%CATEGORY= <some existing category>
"end-request"
```

[ Note the use of macro {TRIGGER.NSEVERITY} here – that's numeric trigger severity, with Not classified being 0 and Disaster being 5.]

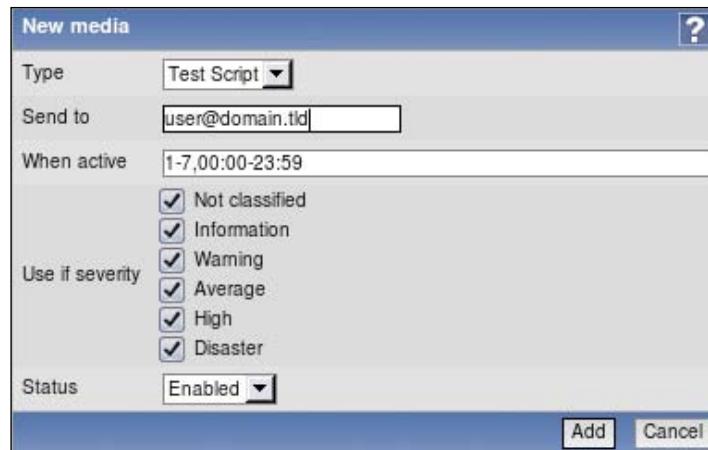
Using scripts as media

While Zabbix supports a decent range of notification mechanisms, there always comes a time when you need something very specific and default methods just don't cut it. Just for such situations Zabbix supports custom scripts to be used as media. Let's try to set one up. Open **Administration | Media types**, click **Create Media Type**. Enter these values:

- **Description:** Enter Test Script
- **Type:** Choose Script
- **Script name:** Enter testscript



When you are done, click **Save**. Now we should make sure this media is used at some point. Go to **Administration | Users**, find **monitoring_user** in the **Members** column and click on it, then click on **Add** in the **Media** section. In the **Type** dropdown select **Test Script** and in the **Send to** field enter **user@domain.tld**.



When you are done, click **Add** and confirm the changes by clicking on **Save** in the user editing form.

We entered the script name but where should the script be placed? Now is the time to return where we haven't been for some time – take a look at `/etc/zabbix/zabbix_server.conf` and check what value does `AlertScriptsPath` option have. By default it's set to `/home/zabbix/bin/`, so Zabbix will look in this directory. As root, create file `/home/zabbix/bin/testscript`.

```
# touch /home/zabbix/bin/testscript
# chmod 755 /home/zabbix/bin/testscript
```

Populate it with the following content:

```
#!/bin/bash

for i in "$@"; do
    echo $i >> /home/zabbix/script_received.log
done
```

As you can see, we are simply logging each passed parameter to a file for examination. Now generate another SNMP trap that would flip the `snmptraps` trigger state. Wait for the e-mail to arrive, then check `/home/zabbix/script_received.log` file. It should contain three lines:

```

user@domain.tld
Critical error from SNMP trap on snmptraps: OK
Critical error from SNMP trap: OK Last value: 127.0.0.1 "some other
string" NET-SNMP-MIB::netSnmpExperimental

```

We can see that three variables are passed to the script:

- The value entered in the **Send to** field when configuring user media
- The **Subject** field
- The **Message** field, with all lines concatenated

From here, basically anything can be done with the data. Passing to issue management systems that do not have e-mail gateway, sending through some media not supported directly by Zabbix, or displayed somewhere.

Let's revisit action configuration now – open **Configuration | Actions**, click on **Test Action** in the **Name** column. So we have a script executed now whenever `monitoring_user` would receive a notification. But what if we would like to skip script for notification, and only use it in some specific action? Thankfully, we don't have to create specific user just for such a scenario. In the **Action operations** block, click on **Edit** next to the last operation – sending of a message to `monitoring_user`. We can see user media list here, but right above it is the dropdown **Send only to**. In this dropdown choose **Email**, and only chosen media is listed now. Click on **Save** in the **Edit operation** block, then on **Save** in the **Action** block.



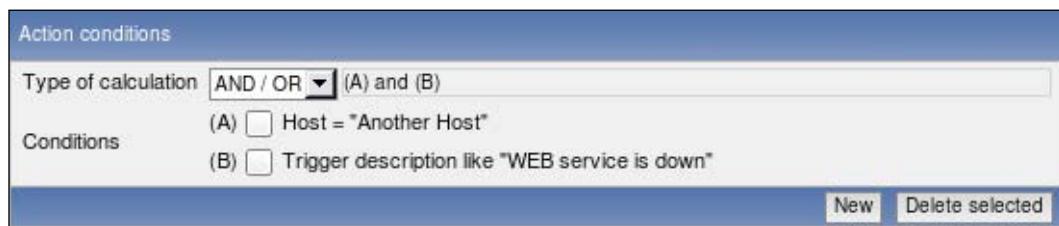
Using **Send only to** option it is possible to use different notification methods for different situations without creating multiple fake user accounts. For example, user might receive e-mail for the first few escalation steps, then an SMS would be sent.

Remote commands

Script media is quite powerful, and it could even be used to execute some command in response to the event. For the command to be executed on the monitored host, though, it would require some mechanism to connect, authorize and such, which might be somewhat too complicated. Zabbix provides another mechanism to respond to events – remote commands. Remote commands could be used in a variety of cases, some of which might be initiating a configuration backup when configuration change is detected or starting a service that has died. We will set up the latter scenario.

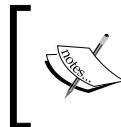
Navigate to **Configuration | Actions**, click **Create Action**. In the **Name** field, enter **Restart Apache**. Click the **New** button in the **Action conditions** block. In the **New condition** block that just appeared, choose **Host** in the first dropdown, then click on **Select** button. In the pop up, choose **Linux servers** in the **Group** dropdown, then click on **Another Host**. Click **Add** button.

Let's create another condition – again, click the **New** button in the **Action conditions** block. In the **New condition** block, in the first dropdown leave dropdowns to the defaults, **Trigger description** and **like** respectively. In the input field next to them enter **WEB service is down**, then click on **Add**. The end result should look as follows:



Now to the operations. In the **Action operations** block click on **New**. In the **Edit operation** block that just appeared, choose **Remote command** in the **Operation type** field. That presents us with a single textbox. Remote actions are to be entered here, with the syntax `hostname : command`. Let's change the contents of this field now to:

```
{HOSTNAME}:sudo /etc/init.d/apache2 restart
```



This step and the steps that come later assume existence and usability of `/etc/init.d/apache2` init script. If your distribution has different control script, use path to it instead.

Notice the use of `{HOSTNAME}` macro – you can use all the macros for remote commands that you can for notifications. We are restarting Apache just in case it has stopped responding instead of simply dying. You can also enter many remote actions to be performed, but we won't do that now, so just click **Add**. To save our new action, click the **Save** button.

Our remote command is almost ready to run, except on the agent side there's still some work to be done, so open `/etc/zabbix/zabbix_agentd.conf` as root, and look for `EnableRemoteCommands` parameter. Set it to `1` and uncomment it, save the config file, and restart `zabbix_agentd`.

That's still not all. As remote commands are passed to the Zabbix agent daemon which is running as a user `zabbix`, we also have to allow this user to actually restart Apache. As evidenced by the remote command, we will use `sudo` for that, so edit `/etc/sudoers` as root and add the following line:

```
zabbix  ALL=NOPASSWD: /etc/init.d/apache2
```

Again, change the script name if you need a different one. Now we are ready for the show. Stop web server on "Another Host". Wait for the trigger to update its state and check the web server status. It should be started again.

While the need to restart services like that indicates a problem that would be best fixed for the service itself, sometimes it can work as an emergency solution or in case of some unresponsive proprietary software vendor.

Summary

This chapter was packed with concepts of reacting to events that happen in your monitored environment. We learned to describe simple checks that define conditions that should be reacted to as trigger expressions. Triggers themselves have useful functionality with dependencies and we can make them depend on each other.

Another large concept was actions. Actions being the things controlling what is performed when a trigger fires, have a very wide range of things to configure at various levels, including conditions of various precision, message contents, and actual operations performed – starting with simple e-mail sending, using custom scripts, and ending with the powerful remote command execution. We also learned about other things affecting actions, like user media configuration and user permissions.

Let's refresh our memory on what alerting related concepts are there:

- **trigger** was a problem definition having a severity, with trigger expression containing information on calculations and thresholds
- **event** was something happening – that is, trigger changing state from PROBLEM to OK and so on
- **action** was a configuration entity, having specific sets of conditions that determine when it is invoked and operations
- **operation** was an action property that defined what to do if this action is invoked, and escalations were configured with the help of operations
- **alert or notification** was the actual thing sent out – e-mail, SMS, or any other message

In addition to simple one time messages we also figured out how built-in escalations work in Zabbix and escalated a few problems. While escalations allow us to draw fairly complex response scenarios, it is important to pay attention when configuring them, especially making sure that the action condition for trigger being in a problem state is created before escalations are enabled. Once enabled, they allow us to perform different operations based on how much time has passed since the problem and other factors.

By now we have learned of three ways to avoid unnecessary notifications:

- By using **trigger expression functions** like min, max, avg to fire a trigger only if a problem has been going on for a defined period of time;
- By using **hysteresis** and only returning to the OK state if the current value is of some comfort distance below (or above) the threshold;
- By creating **escalations** that skip the first few steps, thus only sending out messages if a problem has not been resolved for some time.

The first two methods are different from the last one. Using different trigger functions and hysteresis changes the way the trigger works, impacting how soon it fires and how soon it turns off again. With escalations we do not affect the trigger's behavior (thus they will still show up in **Monitoring | Triggers** and other locations), but we introduce delayed notification whenever the trigger fires.

This part of configuration has several little things that can both make life easier and introduce hard to spot problems. Hopefully the coverage of the basics here will help you to leverage the former and avoid the latter.

7

Simplifying Complex Configuration with Templates

Our current setup has two hosts with similar enough environments, so we copied items from one over to another. But what do we do when there are a lot of hosts with similar parameters to monitor? Copying items over manually over is quite tedious. It's even worse when something has to be changed for all the hosts, like an item interval or an executable's name. Luckily, Zabbix provides a means to configure these things in a unified fashion with the templating system.

Identifying template candidates

Templates allow a Zabbix administrator to reduce their workload and streamline the configuration. But to deploy them properly we have to first identify use cases that require or benefit from templates. Or, to put it short—what templates in Zabbix actually are.

When we created a second monitored Linux host, we manually copied items from the first host. If we wish to, we could also copy over triggers. Such copying around isn't the best job ever, so instead we can create items and triggers for a template, which is then linked to the host in question. As a result of the linkage, the host immediately gets all the items and triggers defined in the template. Later, when we want to change some item parameters for all the hosts, we only have to do it once. Changes made to the template propagate to the linked hosts. So templates make the most sense for items and triggers that you want to have on multiple hosts, like those Linux machines. Even if you have only a single device of a certain class, it might be worth creating a template for it in case new devices appear that could benefit from the same configuration.

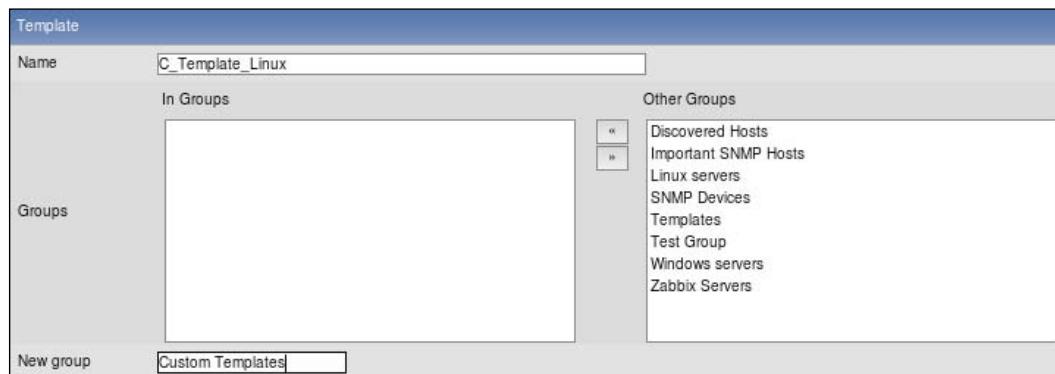
While the `snmptraps` host we created seems like a good candidate for directly created objects, we could have a situation where SNMP agents send in traps that are properly distributed between configured hosts in Zabbix, but every now and then a device would send in a trap that wouldn't have a host or corresponding SNMP item configured. If we still wanted traps like that to get sorted in corresponding items in our generic trap host, we would again use templates to create such items for corresponding hosts and our generic host.

So templates are a valuable tool in Zabbix configuration. That all sounds a bit dry though, so let's set up some actual templates.

Creating a template

Template management is done in the host section, so open **Configuration | Hosts**, choose **all** in the **Group** dropdown, then choose **Templates** from the dropdown in the upper-right corner. As we can see, there are already 42 (a coincidence, surely) predefined templates. We will create our own specialized one though; click on **Create Template**. This opens a simple form that we have to fill in:

- **Name:** Enter `C_Template_Linux`
- **New group:** Enter `Custom Templates`



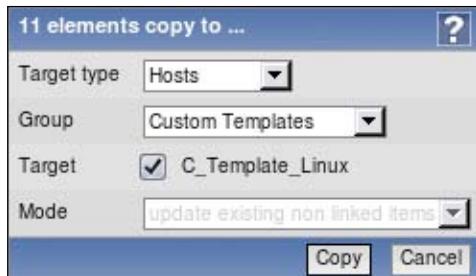
The `C_` at the front of the name might stand for "Custom". We are also creating a new group to hold our templates in, and instead of going through the group configuration we use the shortcut for group creation on this form. When you are done, click **Save**.

We now have the template, but it has no use—there are no items or triggers assigned. Go to **Configuration | Hosts**, where we will use a lazy and quick solution. Select **Linux servers** in the **Group** dropdown, then click on **Items** next to **Another Host**. Mark all items by clicking in the checkbox in the header, next to the **Description** column header, choose **Copy selected to...** in the dropdown below the list, then click **Go** and confirm the pop up.



Remember, to select a sequential subset of checkboxes you can use range selection—hold down *Ctrl*, click first box of the range, hold down *Ctrl* and click last box of the range.

In the next screen, choose **Custom Templates** in the **Group** dropdown. That leaves us with single entry, so mark checkbox next to **C_Template_Linux** in the **Target** section.



When you are done, click **Copy**. All items should be successfully copied.



In this case, the destination host (template) did not have any items configured. As it is not possible to have two items for a single host with the same key, attempting to copy over an already existing item would fail.

Now we have to do the same with triggers, so click on **Triggers** in the navigation bar above the item list (between item sub filter and item list), then click the checkbox in the header next to the **Name** column. This time uncheck the very first trigger in the list—**One SSH service is down**, because this trigger spans both hosts. If we copied this trigger to the template, that would create all kinds of weird effects.



The sequence here, copying items first, then triggers, was important. A trigger cannot be created if an item it references is missing, so attempting to copy triggers first would have failed.

Again, choose **Copy selected to...** in the dropdown below the list, then click **Go** and confirm the pop up. In the next screen, choose **Custom Templates** in the **Group** dropdown. Mark checkbox next to **C_Template_Linux** in the **Target** section, then click **Copy**. All triggers should be successfully copied. Of course, we don't have to create host first, create entities on it, then copy them to template – when creating a fresh template, you'll want to create entities on the template directly. If you have been less careful and haven't thought about templating beforehand, copying like this is a nice way to create template more quickly.

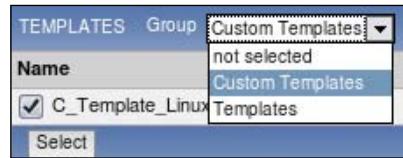
Linking templates to hosts

Now we'd like to link this template to our very first host, "A Test Host". First, let's compare item lists between the freshly created template and that host. Open **Configuration | Hosts** in two browser windows. In one, choose **Linux servers** in the **Group** dropdown, then click on **Items** next to **A Test Host**. In the other one, choose **Templates** in the first dropdown, select **Custom Templates** in the **Group** dropdown, then click on **Items** next to **C_Template_Linux**. Place the windows next to each other and compare the listing.

Description	Triggers	Key	Description	Triggers	Key
CPU Load	Triggers (1)	system.cpu.load	CPU Load	Triggers (0)	system.cpu.load
ICMP ping performance	Triggers (0)	icmppingsec	Experimental SNMP trap	Triggers (0)	netSnmpExperimental
Incoming traffic on interface lo	Triggers (0)	net.if.in[lo]	ICMP ping performance	Triggers (0)	icmppingsec
Incoming traffic on interface eth0	Triggers (0)	net.if.in[eth1]	Incoming traffic on interface lo	Triggers (0)	net.if.in[lo]
Outgoing traffic on interface eth0	Triggers (0)	ifOutOctets[eth0]	Incoming traffic on interface eth0	Triggers (0)	net.if.in[eth1]
SMTP server status	Triggers (0)	smtp	SMTP server status	Triggers (1)	smtp
SSH server status	Triggers (1)	net.tcp.service[ssh]	snmptraps	Triggers (0)	snmptraps
WEB server status	Triggers (0)	net.tcp.service[http,,80]	SSH server status	Triggers (0)	net.tcp.service[ssh]
Zabbix agent version	Triggers (0)	agent.version	Testfile exists	Triggers (1)	vfs.file.exists[/tmp/testfile]
			WEB server status	Triggers (1)	net.tcp.service[http,,80]
			Zabbix agent version	Triggers (0)	agent.version
selected	▼	Go (0)			

We can see here that the template has two more items than the host. Looking at the lists, we can see that the items available on the template but not the host, are both SNMP related items that we added later – **Experimental SNMP trap** and **snmptraps**, and also the check for a file, **Testfile exists**. If the template has three items the host is missing, but in total it only has 2 items more, that means the host should have one item that template doesn't – that's right, **Outgoing traffic on interface eth0** exists for the host but is missing in template. Keeping that in mind, return to **Configuration | Hosts**.

Select **Hosts** in the first dropdown, make sure **Group** dropdown says either **all** or **Linux servers** and click on **A Test Host** in the **Name** column. We finally get to use the **Linked templates** section—click on the **Add** button there. In the pop up, choose **Custom Templates** in the **Group** dropdown and mark the checkbox next to **C_Template_Linux**.

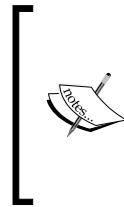


Click **Select**, then click **Save** in the main host editing form. Let's find out what this operation did—click the **Details** link in the upper-left corner. In the expanded details, we can see operations that took place. They start with items being updated.

When a template is linked to a host, identical entities that already exist on the host are linked against the template, but no historical data is lost. Scroll down a bit.

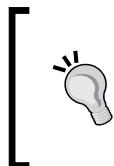
Details	Host updated
Item 'A Test Host:net.tcp.service[ssh]' updated	
Added new item A Test Host:netSnmpExperimental	
Item 'A Test Host:system.cpu.load' updated	
Item 'A Test Host:smtp' updated	
Added new item A Test Host:snmptraps	

Notice how after all the updated items there are two that are added—the ones we previously noticed were missing from the host. When a template is linked, entities that exist on the template only are added to the host and linked to the template. Scrolling down a bit further you'll be able to see that same thing happened with triggers.



Similar to items, mass update can also be used for hosts (by choosing it in the action dropdown below the host list). Care should be taken regarding template linkage using mass update—chosen templates in mass update replace already linked templates. That is, templates chosen in mass update are added to the selected hosts, but any other templates these hosts might have will be removed.

Now this all seems like quite a lot of work for nothing but if we had to add more hosts with same items and triggers, each host would be a tedious manual copying, which is quite error prone. With templates all we have to do is link the template to the freshly added host and all entities are there automatically.



Do not confuse templates for host groups. They are completely different things. Groups serve for a logical host grouping (and permission assigning), but templates define what is monitored on a host, what graphs it has, and so on. What's more, a single host group can contain both ordinary hosts and templates.

Now we could check out how linked items appear in configuration. Open **Configuration | Hosts**, click on **Items** next to **A Test Host**.

Description
C_Template_Linux:CPU Load
C_Template_Linux:Experimental SNMP trap
C_Template_Linux:ICMP ping performance
C_Template_Linux:Incoming traffic on interface lo
C_Template_Linux:Incoming traffic on interface eth0
Outgoing traffic on interface eth0
C_Template_Linux:SMTP server status
C_Template_Linux:snmptraps
C_Template_Linux:SSH server status
C_Template_Linux:Testfile exists
C_Template_Linux:WEB server status
C_Template_Linux:Zabbix agent version

There are two observations we can make right away. First, almost all items are prefixed with a template name (**C_Template_Linux** in this case) in grey text. Obviously, this indicates items that are linked from the template.

Second, a single item is not prefixed like that—**Outgoing traffic on interface eth0**. Remember, that was the only item existing for the host, but not for the template? If entities exist on the host only, linking does not do anything to them—they are left intact and attached to the host directly.

Let's see what a linked item looks like—click on **SMTP server status** in the **Description** column.

Item 'A Test Host:SMTP server status'

Host	A Test Host	Select
Description	SMTP server status	
Type	Simple check	
Key	smtp	
Type of information	Numeric (unsigned)	
Data type	Decimal	
Units		
Use multiplier	Do not use	
Update interval (in sec)	60	
Flexible intervals (sec)	No flexible intervals	
New flexible interval	Delay <input type="text" value="50"/>	Period <input type="text" value="1-7:00:00-23:59"/>
	Add	
Keep history (in days)	<input type="text" value="7"/>	Clear history
Keep trends (in days)	365	
Status	Active	
Store value	As is	
Show value throw map	Service state	
New application	<input type="text"/>	
Applications	<ul style="list-style-type: none"> -None- 	
<input type="button" value="Save"/> <input type="button" value="Clone"/> <input type="button" value="Cancel"/>		

Hey, what happened? Why are most fields shaded and can't be edited? Well, that's what a template is about. Most of the entity (in this case, item) parameters are configured in the template. As we can see, some fields are still editable. This means that we still can disable or enable items per individual host even when they are linked in from a template. The same goes for the update interval, history length, and few other parameters.



Clicking on the template name instead would open an item listing for that template. If you are looking at a linked in item and notice that it would be best changed upstream, this link provides quick access to that.

We now want to make this particular item for this host slightly differ from all other hosts that the template will be linked to, so let's change these things:

- **Update interval:** Change to 360
- **Keep history:** Change to 60

When you are done, click **Save**. Now this host will have two parameters for single item customized, while all other hosts, that will get linked against the template, will receive values from the template. Let's link one more host to our template now. Navigate to **Configuration | Hosts**. This time, instead of opening the host properties, choose **Templates** from the first dropdown, then choose **all** in the **Group** dropdown. Here we can see a full list of templates along with the hosts linked to them. The linkage area in this screen is quite empty right now, with default host **Zabbix Server** linked against **Template_Linux** and our **A Test Host** linked against **C_Template_Linux**.

Templates	Linked to
C_Template_Linux	A Test Host
Template_Linux	Zabbix Server



If the host name is printed in red it means the host is disabled.

Click on **C_Template_Linux** in the **Templates** column. That provides us with a form where multiple hosts can be easily linked against the current template or unlinked from it. In this case we want to link single host. In the **Hosts | Templates** section, choose **Linux servers** in the **Group** dropdown above the **Other** box, mark **Another Host** in that box and click on the **<<** button.

The screenshot shows the Zabbix configuration interface for linking hosts to templates. At the top, there is a dropdown menu labeled "Other | Group" with "Linux servers" selected. Below this, there is a checkbox labeled "Another Host". To the left, under the heading "In", there is a list box containing "A Test Host" and "Another Host". The "Another Host" entry is highlighted with a blue background, indicating it is selected for linking.

In multi-select in the **Hosts | Templates** section now has two hosts that will be linked against the current template. Click **Save**. You can expand **Details** section to see what exactly was done. As we already copied all elements from **Another Host** to the template beforehand, linking this host against the template did not create new items or triggers, it only updated them all. Looking at the template list, we can see two hosts linked against our template.

Move your mouse cursor over the host names—notice how they are actually links. Clicking them would open host properties to verify or change something, like quickly disabling a host or updating its IP address.

Templates ▲	Applications	Items	Triggers	Graphs	Linked templates	Linked to
C_Template_Linux	Applications (0)	Items (11)	Triggers (3)	Graphs (0)	-	Another Host, A Test Host

[ In the list, you can see many predefined templates. While they are great for reference, you are advised against using those without changes. Default templates contain many items and they are polled at short intervals, so for production systems you will definitely want to remove items not relevant for your environment and increase the intervals considerably.]

Changing configuration in template

Now we could try changing an item that is attached to the template. Open **Configuration | Hosts**, choose **Templates** from the first dropdown, select **Custom Templates** from the **Group** dropdown and click on **Items** next to **C_Template_Linux**, then click on **SMTP server status** in the **Description** column. As we can see, all fields are editable when we directly edit an attached instance of an item. Change the **Keep history** field to read "14", then click **Save**. Expand **Details** area at the top of the page to see what got changed.

Details	Item updated
⚠ Item 'Another Host:smtp' updated	
⚠ Item 'A Test Host:smtp' updated	
⚠ Item 'C_Template_Linux:smtp' updated	

This reinforces the principle one more time – **when item is updated in a template, change is propagated to all linked hosts**. This means that with a single action both linked hosts have history keeping period set to 14 days now. But we changed two item properties for one downstream host before, and we just changed one of those for the upstream template. What about downstream host's other item? Let's find out. Go to **Configuration | Hosts**, choose **Linux servers** in the **Group** dropdown and click on **Items** next to **A Test Host**. In the **Description** column, click on **SMTP server status**.

We can see that our downstream change for **Update interval** has been preserved, but the **Keep history** value has been overwritten with the one set for the template. That's because **only changed properties are set downstream** when editing template attached items. Now click **Cancel**.

Macro usage

We previously added triggers from "Another Host" to our template, but we didn't do that for "A Test Host". Let's find out whether it has some triggers we could use in the template. Click on **Triggers** in the navigation bar above the item list. We can see three triggers linked to the template already, one trigger that takes into account items from two different hosts (which we avoided copying before), and at the very top of the list, there's trigger that we are interested in. Mark checkbox next to **CPU Load too high on Test Host for last 3 minutes** trigger in the **Name** column and choose **Copy selected to...** in the dropdown below the trigger list, then click on the **Go** button and confirm the pop up. In the next window, choose **Custom Templates** in the **Group** dropdown, mark the checkbox next to the only remaining target (**C_Template_Linux**) and click **Copy**. This time our copying had three different effects, so let's expand **Details** box again.

Details	Trigger added
⚠ Added trigger "CPU Load too high on Test Host for last 3 minutes" to host "C_Template_Linux"	
⚠ Trigger "CPU Load too high on Test Host for last 3 minutes" updated for host "A Test Host"	
⚠ Added trigger "CPU Load too high on Test Host for last 3 minutes" to host "Another Host"	

The three different effects, as they appear in the details, are:

- The trigger is added to the template. This causes next two effects.
- As "A Test Host" is linked to the modified template and it already has such a trigger, the trigger for that host is updated to reflect the linkage.
- "Another Host" does not have such a trigger, so the trigger is added and linked to the template.

While we are still in the trigger list, select **Another Host** in the **Host** dropdown. Look carefully at the trigger that was added to this host in previous operation.

Name
C_Template_Linux:CPU Load too high on Test Host for last 3 minutes

Wait, that's definitely incorrect. The trigger refers to "Test Host", while this is "Another Host". The trigger name was correct when we first added it, but now the same trigger is applied to multiple hosts. In turn, the reference is incorrect for all hosts except one. Let's try to fix this. Select **Custom Templates** in the **Group** dropdown, then click on **CPU Load too high on Test Host for last 3 minutes** trigger in the **Name** column. Change the **Name** field to:

```
CPU Load too high on {HOSTNAME} for last 3 minutes
```

Yes, that's right, macros to the rescue again. Now click **Save**. In the trigger list for the template, the trigger name has now changed to **CPU Load too high on C_Template_Linux for last 3 minutes**. Let's see how it looks for the previously problematic host - select **Linux servers** in the **Group** dropdown and **Another Host** in the **Host** dropdown.

Name
C_Template_Linux:CPU Load too high on Another Host for last 3 minutes

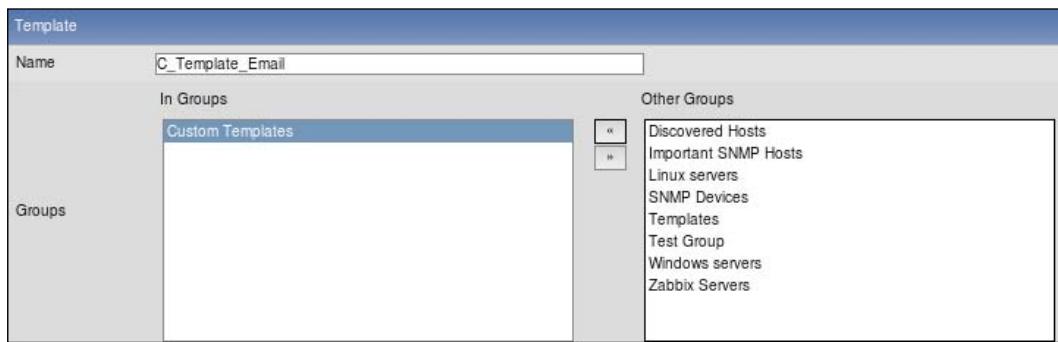
Excellent. The hostname is now dynamically changed on each linked host.

Entities are configured once in the template, but they can apply to many hosts. As a result, macros is the only way to provide "personalized" configuration for linked hosts.

Using multiple templates

There are two monitored hosts now, both having some services monitored and linked to the same template, when one of the hosts gets the responsibility of being an e-mail server removed. Our options from the Zabbix viewpoint include simply disabling e-mail related items for that host, creating a separate template for it and removing e-mail server related entities from main template, instead leaving them on the other server only. There's a better approach, though – splitting e-mail server related entities into a separate template.

Navigate to **Configuration | Hosts**, select **Templates** from the dropdown, then click **Create Template** button. Enter **C_Template_Email** in the **Name** field, mark **Custom Templates** in the **Other Groups** box and click **<<** button, then click **Save**.



Now let's populate this template – select **Custom Templates** in the **Group** dropdown and click on **Items** next to **C_Template_Linux**. Mark the checkbox next to the **SMTP server status** in the **Description** column, choose **Copy selected to...** in the action dropdown, click on the **Go** button and confirm the pop up. In the next screen, select **Custom Templates** in the **Group** dropdown, mark the checkbox next to **C_Template_Email**, and click **Copy**.

That deals with the item, but there's still the trigger left. Click **Triggers** in the navigation bar above the item list and mark checkbox next to **SMTP service is down** in the **Name** column. Choose **Copy selected to...** in the dropdown below the item list, click the **Go** button and confirm the pop up. In the next screen, again select **Custom Templates** in the **Group** dropdown, mark the checkbox next to **C_Template_Email** and click **Copy**.

We now have a simple dedicated e-mail server template that we can link to the hosts. Click on the **Templates list** in the navigational header, then click on **C_Template_Email** in the **Templates** column. In the **Hosts | Templates** block, choose **Linux servers** in the **Group** dropdown. Mark **A Test Host** and **Another Host** in the **Other Hosts** box, click the << button, then click **Save**.

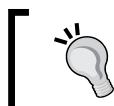


Take a look at the template linkage list after this operation. Each of the custom templates now has two hosts linked.

Templates	Applications	Items	Triggers	Graphs	Linked templates	Linked to
C_Template_Email	Applications (0)	Items (1)	Triggers (1)	Graphs (0)	-	Another Host, A Test Host
C_Template_Linux	Applications (0)	Items (11)	Triggers (4)	Graphs (0)	-	Another Host, A Test Host

There's still something to be done—**C_Template_Linux** has the SMTP service item and trigger, which we'll want to remove now. Click on **Triggers** next to **C_Template_Linux**. Mark checkbox next to **SMTP service is down** in the **Name** column, choose **Delete selected** in the dropdown at the bottom of the list and click on the **Go** button. Confirm the deletion pop up.

To do the same with an item, click on **Items** in the navigation bar, mark the checkbox next to **SMTP server status** in the **Description** column, choose **Delete selected** in the dropdown at the bottom of the list and click on the **Go** button. Confirm the deletion pop up.



The benefit of linking the new template before deleting items is that we didn't lose item history this way.

So a single host can be linked against multiple templates. This allows for a modular configuration where each template only provides a subset of entities, thus a server can be configured to have any combination of basic Linux, e-mail server, web server, file server, and any other templates.

Of course, with a single item and trigger this process seems too complex, but usually the e-mail server would have more parameters such as mail server process counts, SMTP, IMAP, POP3 service status, spam and virus filter status, queue length, and many others. At that point the ability to quickly make a collection of metrics monitored on a machine with a couple of clicks is more than welcome.

Unlinking templates from hosts

But we talked about one server losing the e-mail server duties, and we haven't fulfilled our goal yet. Let's deal with that now. Open **Configuration | Hosts** and choose **Linux servers** in the **Group** dropdown. Our first test host will not be serving SMTP any more, so click on **A Test Host** in the **Name** column and take a look at **Linked templates** section.



This section properly lists two linked in templates. We now want to unlink **C_Template_Email**, but there are two buttons – **Unlink** and **Unlink and clear**. What's the difference then? Let's try it out and start with the one that looks safer – mark checkbox next to **C_Template_Email**, click **Unlink**, then click **Save**. Expand the **Details** link to see what happened.

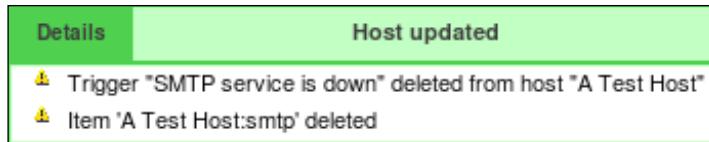
Details	Host updated
⚠ Trigger "SMTP service is down" unlinked	
⚠ Item 'smtp' unlinked	

Both item and trigger got unlinked, so it seems. Was that what we wanted? Let's see. Click on **Items** next to **A Test Host**.

Description ▲
C_Template_Linux:CPU Load
C_Template_Linux:Experimental SNMP trap
C_Template_Linux:ICMP ping performance
C_Template_Linux:Incoming traffic on interface lo
C_Template_Linux:Incoming traffic on interface eth0
Outgoing traffic on interface eth0
SMTP server status
C_Template_Linux:snmptrap
C_Template_Linux:SSH server status
C_Template_Linux:Testfile exists
C_Template_Linux:WEB server status
C_Template_Linux:Zabbix agent version

Well, not quite. So simple unlink does unlink only, and leaves a copy of the item on the previously linked host. That is handy if we want to create a different item or leave item on host to keep data for historical reasons, but not this time. To solve the current situation, we can manually delete both the trigger and the item, but that wouldn't be so easy if the host additionally had a bunch of directly attached entities. In that case one would have to manually hunt them down and remove, which allows for mistakes to be made. Instead, let's try a different route—relink this template, then remove it without a trace.

Click on **Hosts list** in the header, then click on **A Test Host** in the **Name** column. In the host properties, click **Add** in the **Linked templates** section. Mark checkbox next to **C_Template_Email**, click **Select**, then click **Save**. We are now back in our starting point with two templates linked—time to unlink again. Click on **A Test Host** in the **Name** column. Mark checkbox next to **C_Template_Email** in the **Linked templates** block, but this time, click **Unlink and clear**. Click **Save**, then expand **Details**.



And now it's done. Both item and trigger are actually deleted now. Look at the host list, notice how **Templates** column again offers a quick overview—that comes handy when you might want to quickly verify template linkage for all hosts in a particular group.

Name	Applications	Items	Triggers	Graphs	DNS	IP	Port	Templates
Another Host	Applications (0)	Items (11)	Triggers (5)	Graphs (0)	-	10.1.1.100	10050	C_Template_Email, C_Template_Linux
A Test Host	Applications (0)	Items (11)	Triggers (4)	Graphs (0)	-	127.0.0.1	10050	C_Template_Linux

Nested templates

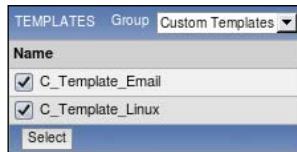
The one host still serving e-mail now has two templates assigned. But what if we separated out in individual templates all services, applications, and other data that can be logically grouped? That would result in a bunch of templates that we would need to link to single host. This is not tragic, but what if we had two servers like that? Or three? Or twenty? At some point even a configuration with templates can become hard to manage—each host can easily have a template count of a dozen in large environments.

Simplifying Complex Configuration with Templates

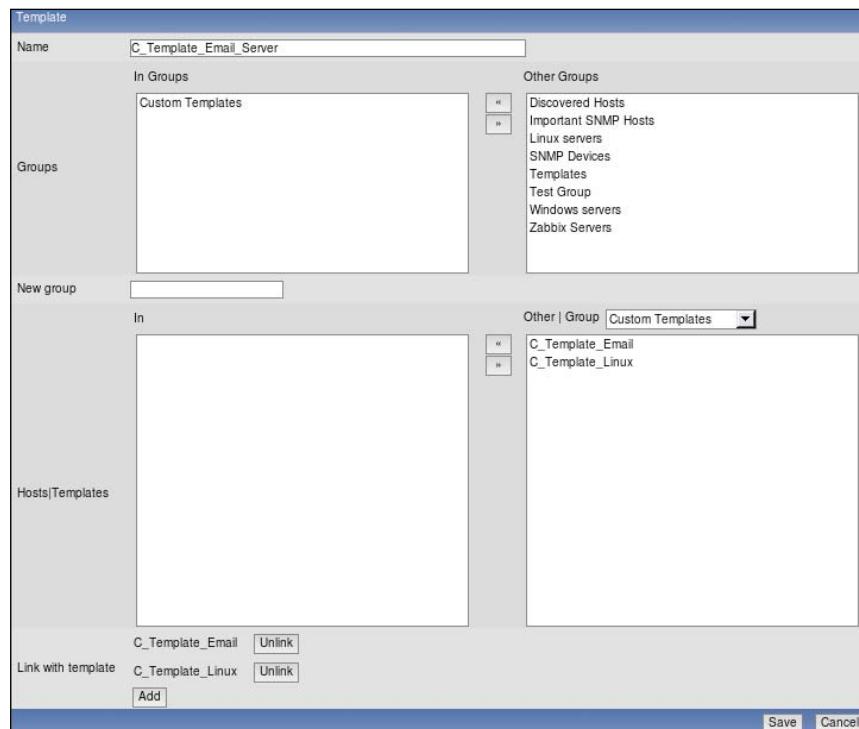
This is where the simplicity is coupled with powerful functionality. Behind the scenes, templates aren't that different from hosts. Actually, they are hosts, just somewhat special ones. This means that a template can be linked to another template, thus creating nested configuration.

How does that apply to our situation? Let's create a simple configuration that would allow easy addition of more hosts of the same setup. In **Configuration | Hosts**, select **Templates** from the first dropdown, then click **Create Template** button. In the **Name** field enter **C_Template_Email_Server**, mark **Custom Templates** in the **Other Groups** box and click the **<<** button.

See the **Link with Template** section below? Here we can link other templates to this one, so click **Add**. In the pop-up window mark the checkboxes next to **C_Template_Email** and **C_Template_Linux**.



Click **Select**. Both templates are added to the linkage section.



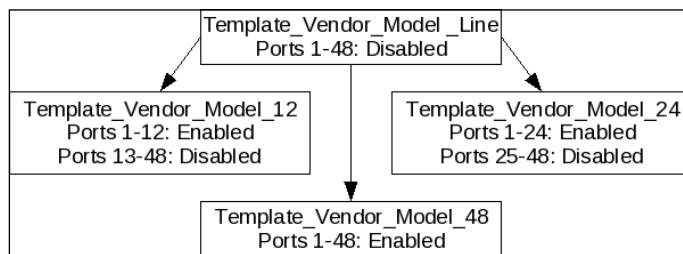
When you are done, click **Save**. We now have a template that encompasses basic Linux system configuration with e-mail server installed and running, so we still have to properly link it to a host that will serve this role.

While still having **Configuration | Hosts** open, select **Hosts** from the first dropdown, then click on **Another Host** in the **Name** column. In the **Linked templates** section, mark both checkboxes, click on the **Unlink** button. Click **Save**, then click on **Another Host** again. This time, click on the **Add** button in the **Linked templates** section. In the pop-up window, mark the checkbox next to **C_Template_Email_Server** and click **Select**, then click **Save** at the bottom of the form. The action successfully completes, so expand the **Details** link. As we can see here, all elements are updated. Essentially, the previous templates were unlinked, but the items and triggers left in place and then they got relinked to the new template. The biggest benefit from such a sequence again was keeping all item historical data.

But the biggest thing we did here, we created a **nested template**. Such a template is linked against other templates, thus it inherits all the items, triggers, and other characteristics, while usually making some modifications to the original template conditions. In this case, our nested template contains entities from two other templates.

While that seems to be only a little gain from the previous situation, two templates linked to a single host, it is a very valid approach when your monitored environment is slightly larger. If there's a single host requiring a specific combination of multiple templates, it is fine to link those templates directly to the host. As soon as the count increases, it is more convenient to set up template nesting, creating a single template to link for these hosts. When you have done that, adding a new host of the same class requires linking against a single template only, which greatly simplifies configuration and minimizes chance for mistakes.

Nested templating allows a great deal of flexibility, as you can also make minor changes to linked items. One example might be a bunch of similar switches that you have to monitor. They might have the same SNMP OIDs, but have simple differences like port count – there would be switches with 12, 24, and 48 ports. Instead of creating a template for each of the sub-models, you could instead use a nested setup like this:



In this example, all specific model templates are linked to the generic model line template, thus any change needed, like changing the interval for all the port items, can be performed only once. We could also have linked these templates more, like linking a 24 port model to a 12 port model, and a 48 port one to a 24 port one—though in this case such deep linking would not provide a meaningful benefit.

In a situation like this, adding a new switch would be as simple as linking it against the correct port count template, while we still benefit from the ease of updating the configuration.

Summary

In Zabbix, templates play a major role in simplifying the configuration and allowing large scale changes. If you are a proficient user of word processors, you probably use styles. The same concept is used in TeX, CSS styles for the web and elsewhere—separating content from the presentation helps to reduce amount of work required when changes have to be made.

While the comparison to styles might seem far-fetched at first, it actually is similar enough. Just like styles, you can separate a host from the services you provide, and you can define these services in a centralized fashion. Same as with a word processor document having heading style that allows to change font size for all headings of that level with one action, templates in Zabbix allow to change some parameter for all linked in hosts, direct or nested.

As we saw with all the rearrangement of items and triggers in templates, it is easier to plan a sane template policy before getting to the actual configuration. It is strongly suggested that you sit down and draw at least a very basic hierarchy of monitored things before rushing into the configuration—that will make things easier in the long run.

8

Visualizing the Data

So far we have only briefly looked at some basic available data visualization options, mostly simple graphs that show us how an item has changed over time. That's not all Zabbix provides — there are more options, which we will explore now. The visualization options of Zabbix that we will look at in this chapter include:

- Graphs, including simple and custom ones
- Maps that can show information based on geographical layout
- Screens that can include other elements
- Slideshows that change displayed information on a periodic basis automatically

Visualize what?

We have set up actions that send us information when we want to be informed, we have remote commands that can even restart services as needed and do many other things. So why visualize anything?

While for most this question will seem silly because we know quite well what data we are interested in and why we would want to see that data in a more polished form, some goals that can be achieved might not be that obvious.

Of course, it can be easier to assess the problem when looking at graphs, as this allows to easily spot time when problem started, correlate various parameters easily, and spot recurring anomalies. Things like graphs can also be used as a simple representation, to answer questions like "so what does that Zabbix system do?" That does come handy when trying to show results and benefits to non-technical management.

Another useful area is displaying data on a large screen. That usually is a high-level overview of the system state, and is placed in system operators' or helpdesk location. Imagine a large plasma TV, showing the helpdesk map of the country, listing various company locations and any problems in all of those.

There surely are many more scenarios you can come up with when having a nice graph or otherwise visually laid out information can be very helpful. We'll now look at exact options that are already built-in for Zabbix.

Single elements

We can combine visual elements in two categories – single and combined. With "single" we will refer to individual elements, showing certain information in one container, like a graph or a network map. While single elements can contain information from many items and hosts, in general they cannot include other Zabbix frontend elements.

Graphs are hard to beat for capacity planning when trying to convince management of a new database server purchase. If you can show an increase in visitors to your website and that with the current growth it will hit current limits in a couple of months, that is so much more convincing.

Graphs

While the previous definition might sound confusing, it's quite simple – an example of a single visual element is a graph. A graph can contain information on one or more items, but it cannot contain another Zabbix visual elements, like another graph. Thus a graph can be considered a single element.

Simple graphs

We already looked at the first visual element in this list; so called simple graphs. They are somewhat special, because there is no configuration required, you don't have to create them – simple graphs are available for every item. Right? Not quite. They are only available for numeric items, as it wouldn't make much sense to graph textual items. To refresh our memory, let's look at items in **Monitoring | Latest data**.

+ Description	Last check	Last value	Change	History
- - other - (11 Items)				
CPU Load	02 Mar 2010 18:51:01	1.18	+0.13	Graph
Experimental SNMP trap	12 Feb 2010 14:02:32	10.1.1.100 "test"	-	History
ICMP ping performance	02 Mar 2010 18:50:29	0.49 ms	-0.02 ms	Graph
Incoming traffic on interface lo	02 Mar 2010 18:51:00	28.28 B	-	Graph
Incoming traffic on interface eth0	02 Mar 2010 18:51:00	297.32 B	-187.61 B	Graph
SMTP server status	02 Mar 2010 18:50:28	Up (1)	-	Graph
snmptraps	12 Feb 2010 11:48:06	10.1.1.100 "test"	-	History
SSH server status	02 Mar 2010 18:50:25	Up (1)	-	Graph
Testfile exists	02 Mar 2010 18:51:15	1	-	Graph
WEB server status	02 Mar 2010 18:50:24	Up (1)	-	Graph
Zabbix agent version	23 Feb 2010 15:26:44	1.8.1	-	History

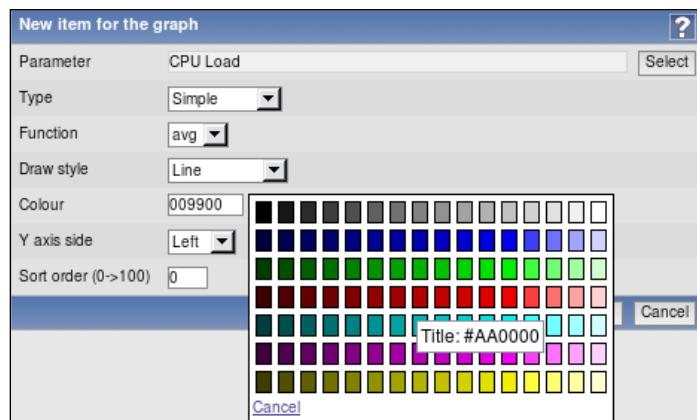
For anything other than numeric items the **History** column shows **History**.

For numeric items, we have **Graph** links. This depends only on how data is stored – things like units or value mapping do not influence availability of graphs. If you want to refresh information on basic graph controls like zooming, please refer to *Chapter 2*.

While no configuration is required for simple graphs, they also provide no configuration capabilities. They are easily available, but quite limited. Thus being useful for single items, there is no way to graph several items or change the visual style. Of course, it would be a huge limitation if there was no other way, but luckily there is – custom graphs.

Custom graphs

These have to be created manually, but they allow a great deal of customizability. To create a custom graph, open **Configuration | Hosts**, select **Graphs** in the first dropdown, then click the **Create Graph** button. Let's start with a recreation of a simple graph, so enter **CPU Load** in the **Name** field, then click the **Add** button in the **Items** section. in the pop up, click the **Select** button next to the **Parameter** field. In the next pop up we have to choose an item. Make sure **Custom Templates** is selected in the **Group** dropdown, select **C_Template_Linux** in the **Host** dropdown, and click on **CPU Load** in the **Description** column. While we can change some other parameters for this item, for now let's change color only. Color values can be entered manually, but that's not too convenient, so just click on the colored rectangle. That opens a nice color chooser. Pick one of the middle range red colors – notice that holding your mouse cursor on top of a cell for a few seconds will open a tooltip with color value.



When done, click **Add**. The following graph is immediately updated to reflect the current configured state. That doesn't help us much currently, as we chose an item from a template, which does not have any data itself.



The Zabbix color chooser provides a nice table, but it still is missing some colors; like orange, for example. You can enter an RGB color code directly in hex form (for example, orange would be similar to FFAA00). To find other useful values, you can either experiment, or use one of the many available online color calculators, like <http://www.rentadabob.nl/~english/rgbhexcalc.htm>.

Working time and trigger line

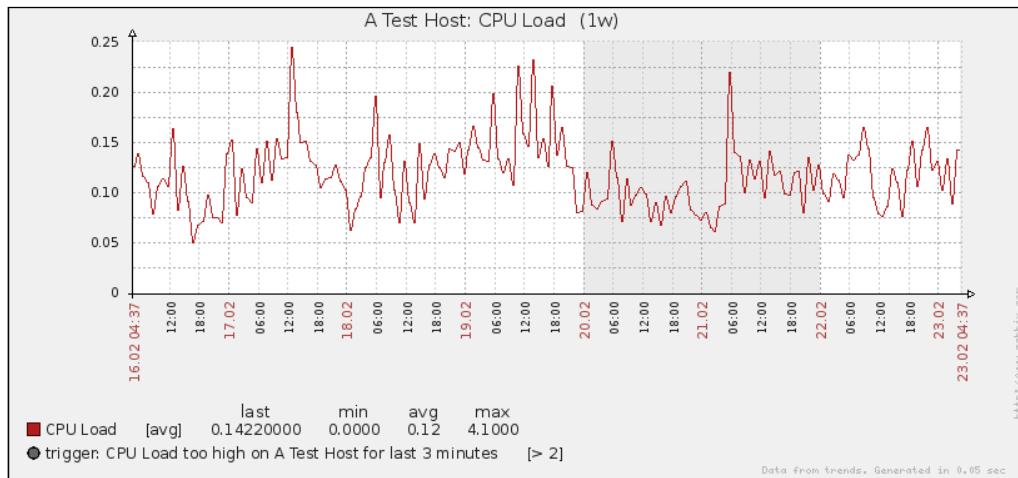
We already saw one simple customization option—changing the line color. Let's do some more changes here. Mark the checkboxes next to **Show working time** and **Show triggers**, then click **Save**.

Our custom graph is now saved, but where do we find it? While simple graphs are available from **Monitoring | Latest** data section, custom graphs have their own section. Go to **Monitoring | Graphs**, select **Linux servers** in the **Group** dropdown, **A Test Host** in the **Host** dropdown, and in the **Graph** dropdown, select **CPU Load**.



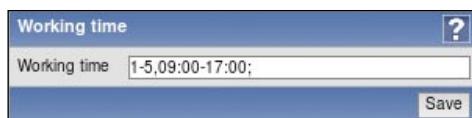
There's an interesting thing we did here, that you probably already noticed. While the item was added from a template, graph is available for host, with all the data correctly displayed for that host. That means the important concept, templating, works here as well. **Graphs can be attached to templates** in Zabbix, and afterwards are available for each host that is linked to such a template.

Well, those two options made no difference—the graph looks the same as it was in the configuration preview without trigger and working time options enabled. Or wasn't it? Click on 1w control in the lower-left corner, next to the Zoom caption.



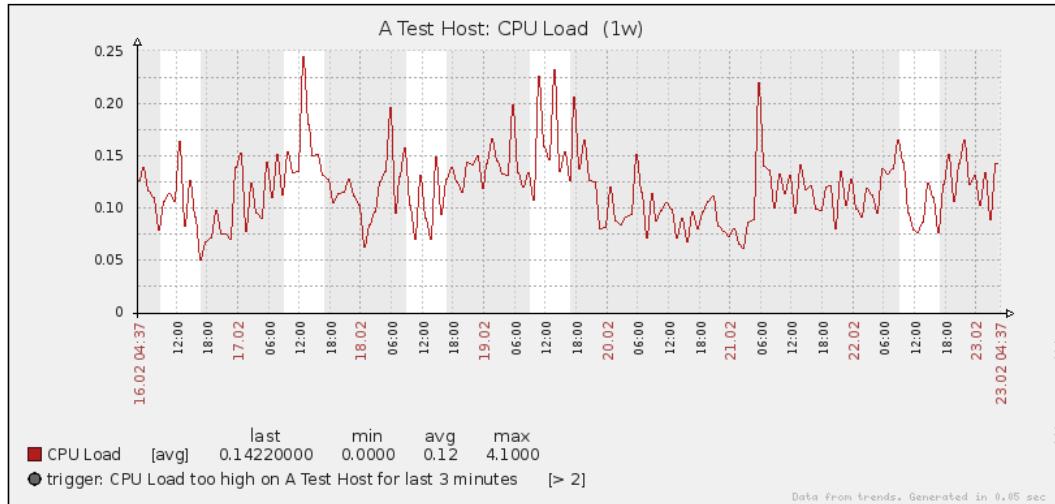
That's better, we can see that weekend is marked with a grey area. By the way, that's the same with simple graphs, except that you have no way to disable working time displaying for them. But weekends only might not be sufficient – it might be desirable to also easily see daily working hours on the graph. Let's try to influence this. Open **Administration | General**, choose **Working time** from the dropdown.

This option uses the same syntax as "When active" for user media, discussed in *Chapter 6* – Monday-Sunday represented by 1-7 and a 24-hour clock used to configure time. Currently this field reads **1-5,00:00-24:00**, which means Monday-Friday, midnight till midnight. Let's modify this somewhat to read **1-5,09:00-17:00**.



That means Monday-Friday, 09-17, which might be a reasonable working time. Click **Save** to accept the changes. Navigate back to **Monitoring | Graphs**, make sure **CPU Load** is selected in the **Graph** dropdown.

 It is suggested to use two browser tabs or windows, keeping **Monitoring | Graphs** open in one, and the graph details in the **Configuration** section in the other. This way, you will be able to refresh the monitoring section after making configuration changes, saving a few clicks back and forth.



There, grey areas now mark our real night time and weekends, thus allowing us to easily identify at what time issues started and ended.

Anyway, what about that trigger option we enabled? Taking a second look at our graph, we can see both a dotted line and a legend entry, which explains that it depicts the trigger threshold. The trigger line is displayed for simple expressions only.



If the load on your machine has been low during the displayed period, you won't see the trigger line displayed on a graph. The y-axis auto-scaling will exclude the range where trigger would be displayed.

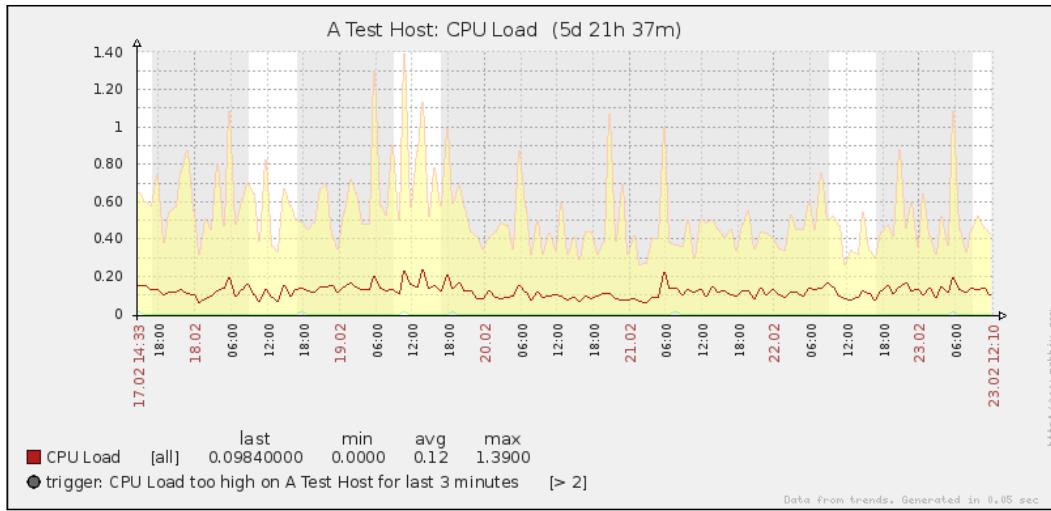


Again, the trigger line is displayed in simple graphs with no way to disable it.

What we have now is quite similar to the simple graphs, though there's one notable difference when the displayed period is longer – simple graphs show three different lines, with the area between them filled, while our graph has single line only (the difference is easier to spot when displayed period approaches three days). Can we duplicate that behavior? Go to **Configuration | Hosts**, choose **Graphs** in the first dropdown. In the list, we can see all graphs that reference items from the selected host, of which there is only one currently. Click on **CPU Load** in the **Name** column to open the editing form, then click **C_Template_Linux: CPU Load** in the **Items** section. Take a closer look at the **Function** dropdown.



Currently we have **avg** selected, which simply draws average values. The other choices are quite obvious, **min** and **max**, except the one that looks suspicious, **all**. Select **all**, then click **Save** and again **Save** in the graph form. Again, open **Monitoring | Graphs**, make sure **CPU Load** is selected in the **Graph** dropdown.



Graph now has three lines, representing minimum, maximum, and average values for each point in time, although in this example lower line is almost always at zero.

Two y-axis

We have now faithfully replicated a simple graph (well, the simple graph uses dark green for average values, while we use red, which is a minor difference). While such an experience should make you appreciate the availability of simple graphs, custom graphs would be quite useless if that was all we could achieve with them. Customizations like color, function and working time displaying can be useful, but they are quite minor ones. Let's see what else can we throw at the graph. First, some quick preparation—as we are creating graph for the template, we'd benefit from having one item that we left directly linked to a host to be linked against the template now.

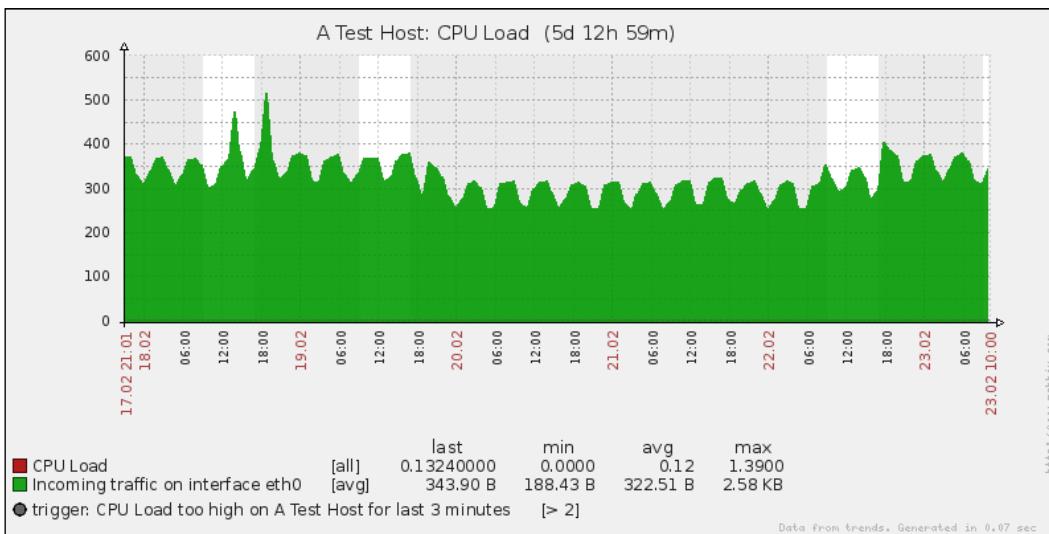
Open **Configuration | Hosts**, select **Linux servers** in the **Group** dropdown and click on **Items** next to **A Test Host**, then mark the checkbox next to **Outgoing traffic on interface eth0**, choose **Copy selected to...** in the dropdown below the items and then click the **Go** button. Confirm the pop up, and in the copying form, select **Custom Templates** in the **Group** dropdown, mark the checkbox next to **C_Template_Linux**, then click **Copy**.

Now we are ready to improve our graph—open **Configuration | Hosts**, choose **Graphs** in the first dropdown, select **Custom Templates** in the **Group** dropdown, and click on **CPU Load** in the **Name** column next to **C_Template_Linux**.



Notice that you cannot choose any other host or template now. The reason is that a graph can contain either items from a single template, or from one or more hosts. If a graph has item from a host added, then no templated item can be added to it anymore.

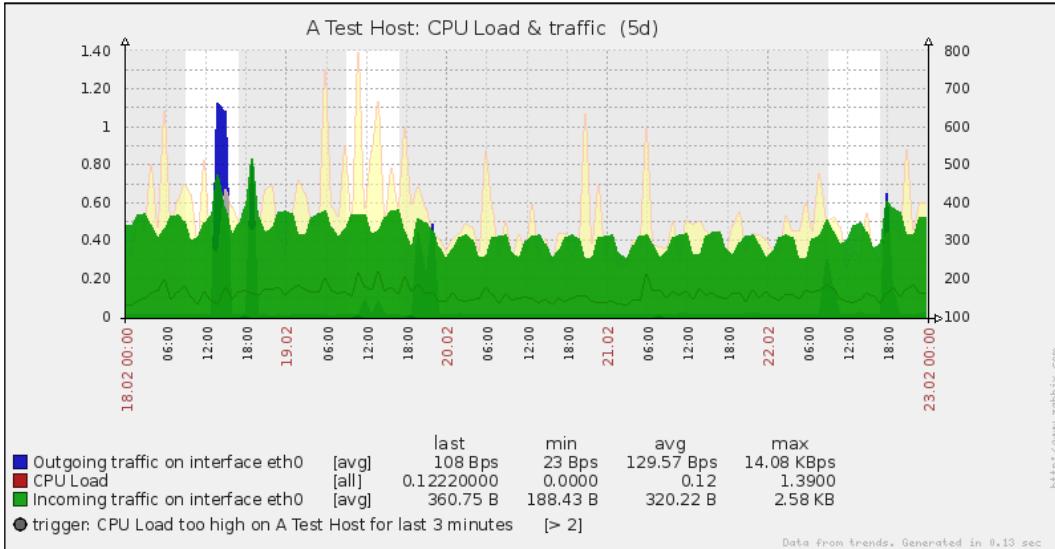
Click on **Add** in the **Items** section, then click **Select** next to the **Parameter** field. In the next pop up, click on **Incoming traffic on interface eth0** in the **Description** column. While we're still in the item properties editing form, select **Filled region** in the **Draw style** dropdown, then click **Save** and again **Save** in the graph editing form. Check the graph in **Monitoring | Graphs** section.



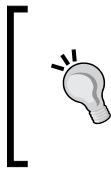
Hey, that's quite ugly. Network traffic values are much larger than system load ones, thus system load can be barely seen as a small line at the bottom of the graph. Let's try to fix this back in the graph configuration list by clicking on **CPU Load graph**, then clicking on **C_Template_Linux: Incoming traffic on interface eth0**. Change **Y axis side** dropdown to **Right**, then click **Save**.

That covers incoming traffic, now let's get to the outgoing traffic. Click **Add** in the **Items** section, then **Select** next to the **Parameter** field. Click on **Outgoing traffic on interface eth0**. Choose **Filled region** in the **Draw style** dropdown and choose some other color – one of the blue shades should do. Don't forget to set the **Y axis side** to **Right** again, otherwise the graph will become unreadable again.

As our graph now has more than just CPU load, change the **Name** field to something like "CPU Load & traffic", then click **Save** for the graph itself. Take a look at **Monitoring | Graphs** to see what this change did.



That's way better, now each of the different scale values is mapped against an appropriate y-axis. As we can see, for this machine the network traffic increase more or less corresponds to the overall system load increase.



Notice how the filled area is transparent where it overlaps with another area. This allows us to see values even if they are behind a larger area, but it's suggested to avoid placing many elements with the filled region draw style on the same graph, as the graph can become quite unreadable in that case.

Sort order

Still, outgoing traffic is mostly lower than incoming traffic, thus the green area is covering the blue one, making the graph look less pretty. If your graphs look similar, you'll probably want to push green color in the background – let's try to do that.

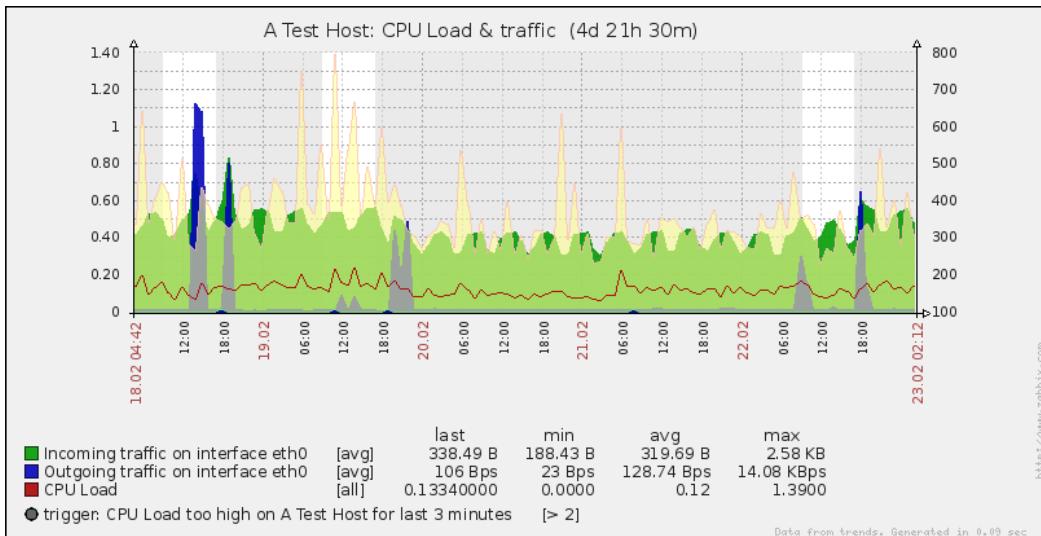
Back in **Configuration | Hosts**, select **Graphs** in the first dropdown and click on **CPU Load & traffic**, then look at the item list. We want outgoing traffic to appear in front of incoming traffic, so click **Down** next to the **C_Template_Linux: Outgoing traffic on interface eth0** entry. Most likely, item ordering has changed in some weird way. That's because the actual ordering in the table does not matter – that control only changed the sort order of the items, and if you click on **C_Template_Linux: Outgoing traffic on interface eth0**, you'll notice that this item now has sort order of 2. In Zabbix graphs, sort order means "order in which items are laid out on the graph", thus the more in the foreground we want an item, the larger value we should use for it. **Up** and **Down** links only change this value by one unit each time. You can edit the values directly by clicking on each item, then saving it, so edit the values to be like this:

- **C_Template_Linux: CPU Load** – 30
- **C_Template_Linux: Outgoing traffic on interface eth0** – 20
- **C_Template_Linux: Incoming traffic on interface eth0** – 10

Why are the used values so large? Well, just in case you decide to add another items in between, you'll have the room to do so easily without reordering existing ones. The final item table should look similar to this:

C_Template_Linux: Incoming traffic on interface eth0	avg	Simple	Right	Filled region	█	Down
C_Template_Linux: Outgoing traffic on interface eth0	avg	Simple	Right	Filled region	█	Up Down
C_Template_Linux: CPU Load	all	Simple	Left	Line	█	Up

Don't forget to click **Save** now, no changes made will be saved until then. It's time to find out how the graph looks now at **Monitoring | Graphs**.



Now that's even better. The smaller filled area is displayed in front of the larger one, while load line is drawn on top of both areas.



You can easily save graphs right from the frontend by right clicking and saving them. This can be useful to send one over e-mail or place on a webpage. There's one little trick, though—you have to click outside of the area that accepts dragging actions for zooming. Try legend area, for example. This works both for custom and simple graphs.

Placing things like the system load and web server connection count on a single graph would be quite useless without using two y-axis. You can probably think of many other item combinations that would benefit from such a feature.

Custom y-axis scale

As you have probably noticed, the y-axis scale is automatically adjusted to make all values fit nicely in the chosen range. Sometimes you might want to customize that though. Let's prepare a quick and simple dataset for that.

Go to **Configuration | Hosts**, select **Templates** in the first dropdown, and click on **Items** next to **C_Template_Linux**, then click the **Create Item** button.

Fill in these values:

- **Description:** Enter **Diskspace on \$1 (\$2)**
- **Key:** Enter **vfs.fs.size[/,total]**
- **Units:** Enter **B**
- **Update interval:** Enter **3600**

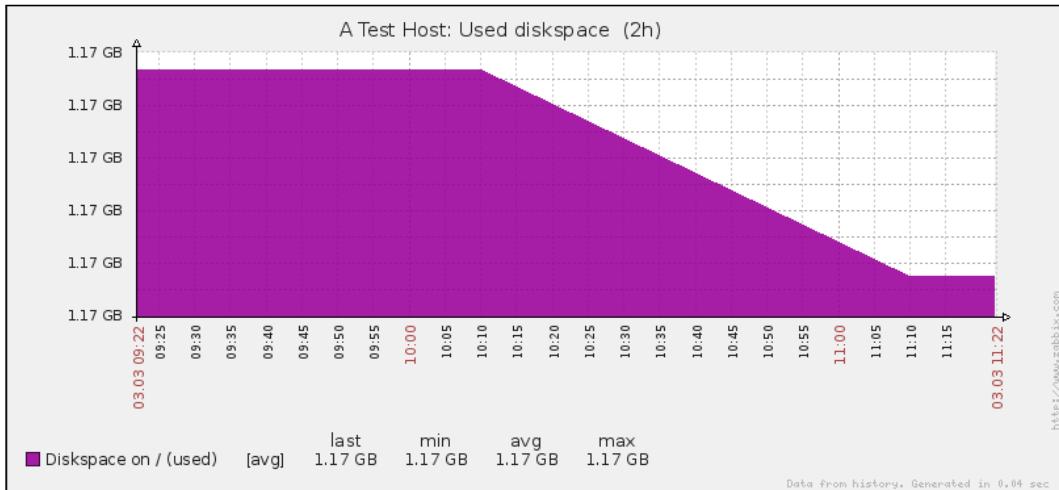
When you are done, click **Save**.

Now, click on **Diskspace on / (total)** in the **Description** column and click the **Clone** button at the bottom. Make only a single change, replace **total** in the **Key** field with **used**, so that key now reads **vfs.fs.size[/,used]**, then click **Save**.

Choose **Graphs** from the first dropdown and click **Create Graph**. In the **Name** field, enter **Used diskspace**. Click the **Add** button in the **Items** section, then click **Select** next to the **Parameter** field. In the new pop up, choose **Custom Templates** in the **Group** dropdown and **C_Template_Linux** in the **Host** dropdown, then click on **Diskspace on / (used)** in the **Description** column. In the **Draw style** dropdown, choose **Filled region**. Feel free to change the color, then click **Add**, followed by **Save**.

Visualizing the Data

Take a look at what the graph looks like in the **Monitoring | Graphs** section for **A Test Host**.



So this particular host has slightly more than one gigabyte used on the root file system (and that has slightly changed during the last hour). But the graph is quite hard to read—it does not show how full the partition relatively is, as the y-axis starts at about the same value data has. Regarding the upper values on the y-axis, we can figure out the total disk space on root file system in **Monitoring | Latest data**.

Diskspace on / (total)	03 Mar 2010 11:10:16	3.74 GB
------------------------	----------------------	---------

So there is 3.74 GB of space, which also is not reflected on the graph. Let's try to make the graph slightly more readable. In **Configuration | Hosts**, choose **Graphs** in the first dropdown, click on **Used diskspace** in the **Name** column and take a look at two options—**Y axis MIN value** and **Y axis MAX value**. They are both set to **Calculated** currently, but that doesn't seem to work too well for our current scenario. First, we want to make sure graph starts at zero, so change the **Y axis MIN value** to **Fixed**. This allows us to enter any arbitrary value, but a default of zero is what we want.

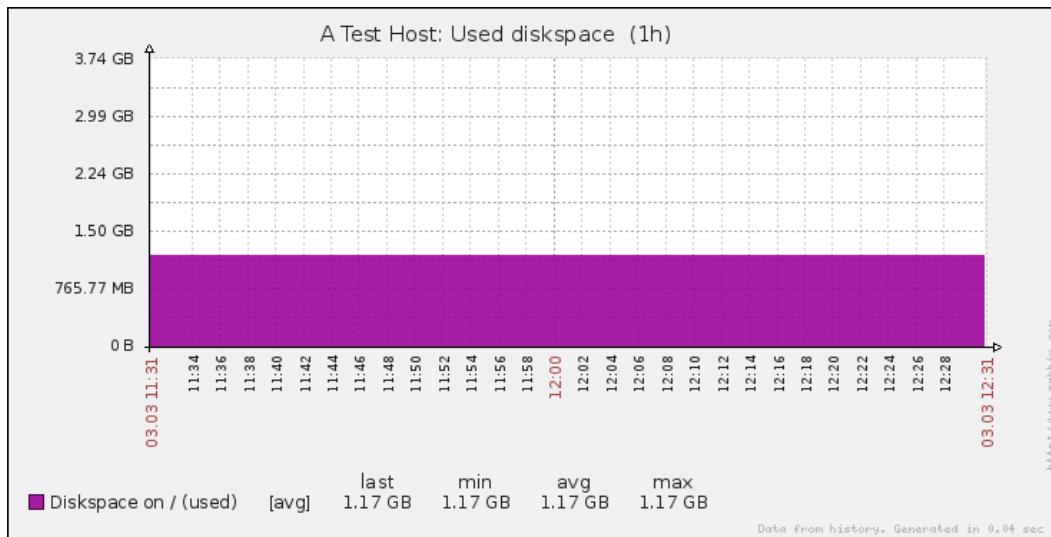
For the upper limit, we could calculate what 3.74 GB is in bytes and insert that value, but what if available disk space changes? Let's say this system gets disk upgrade and its root filesystem has more space.

Diskspace on / (total)	03 Mar 2010 11:10:18	14.45 GB
------------------------	----------------------	----------

It increased for almost five times its original value. So does this mean we will have to update Zabbix configuration each time this could happen? Luckily, no. There's a nice solution for situations just like this. In **Y axis MAX value** dropdown, select **Item**. That adds another field and a button, so click on **Select**. Select **Linux servers** in the **Group** dropdown, **A Test Host** in the **Host** dropdown, then click on **Diskspace on / (total)** in the **Description** column. The final y-axis configuration should look like this:

Y axis MIN value	Fixed	0.0000
Y axis MAX value	Item	Diskspace on / (total)

If it does, click **Save**. Now is the time to check out the effect on the graph – see the **Used diskspace** graph in **Monitoring | Graphs**.



Now the graph allows to easily identify how full the disk is. This approach can also be used for used memory or any other item where you want to see full scale of possible values.

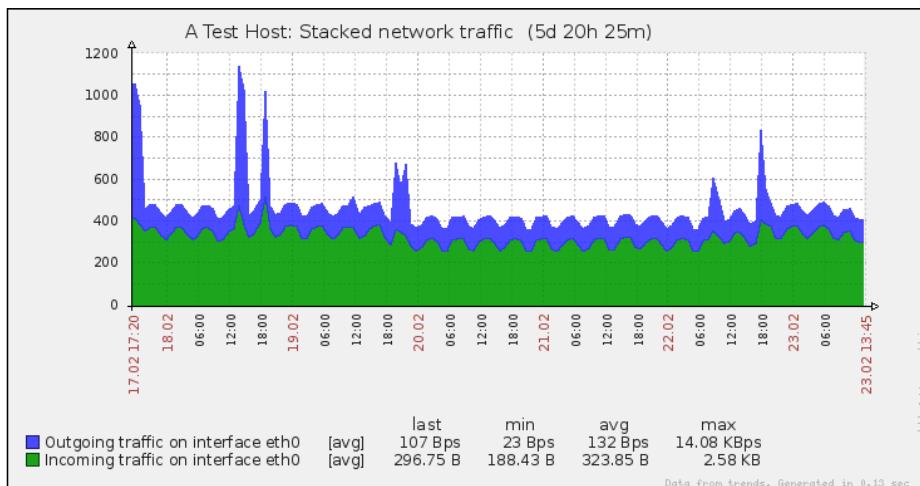
Stacked graphs

Our previous graph that contained multiple items, network traffic and CPU load, placed the items one on top of another. But sometimes we might want to place them one on top of one another on a different axis—stack them. Possible uses could be memory usage, where we could stack buffers, cached and actual memory (and link the y-axis maximal value to the total amount of memory), stacked network traffic over several interfaces to see total network load, or any other situation where we would want to see both total and value distribution. Let's try to create a stacked graph. Open **Configuration | Hosts**, choose **Graphs** in the first dropdown, and click **Create Graph** button.

In the **Name** field enter **Stacked network traffic** and change **Graph type** dropdown to **Stacked**. Click on **Add** in the **Items** section and click on **Select** button next to the **Parameter** field. In the upcoming pop up, make sure **C_Template_Linux** is selected in the **Host** dropdown and click on **Incoming traffic on interface eth0** in the **Description** column. Click **Add** in the item details form.

If we had several active interfaces on the test machine, it might be more interesting to stack incoming traffic over all the interfaces, but let's keep it simple this time and simply add outgoing traffic on top of the incoming one—click **Add** in the **Items** section again, then click on the **Select** button next to the **Parameter** field. This time choose **Outgoing traffic on interface eth0** in the **Description** column and also change the color—some nice shade of blue will do. When done, click **Add** in the item details form and **Save** in the graph editing form. Notice how we did not have a choice of draw style when using a stacked graph—all items have the **Filled region** type.

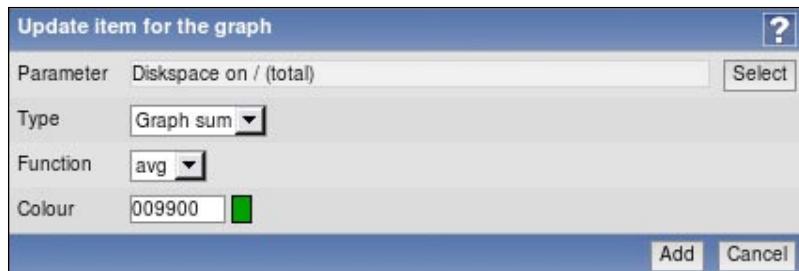
Check out **Monitoring | Graphs** to see the new graph, make sure to select **Stacked network traffic** from the dropdown.



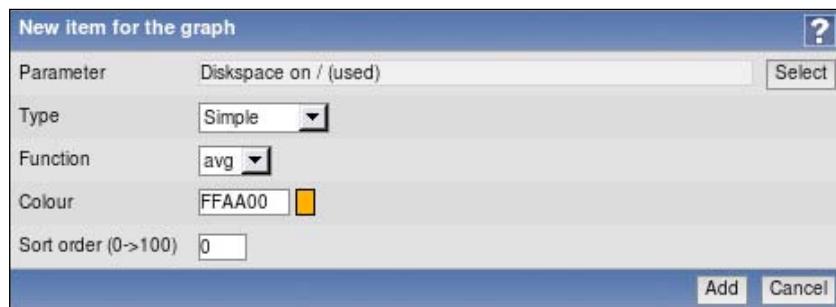
With stacked graphs we can see both the total amount (indicated by the top of the data area) and the individual amounts that items contribute to the total.

Pie graphs

The graphs we already created offer a wide range of customizations, but sometimes you just want it to look different. For those situations, it is possible to create pie graphs. Go to **Configuration | Hosts**, select **Graphs** in the first dropdown, and click on **Create Graph** button. In the **Name** field, enter **Used diskspace (pie)**. In the **Graph type** dropdown, choose **Pie**. Click **Add** in the **Items** section, then click **Select** next to the **Parameter** field. Make sure **C_Template_Linux** is selected in the **Host** dropdown, then click on **Diskspace on / (total)** in the **Description** column. Now, total disk space is a slightly special item, so select **Graph sum** in the **Type** dropdown.

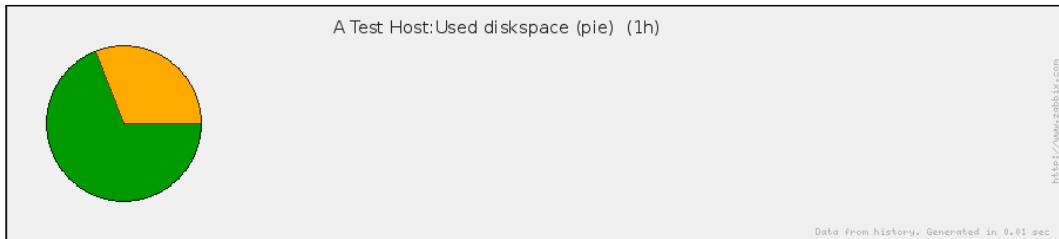


Click the **Add** button to add our graph sum item. Click **Add** button in the **Items** section again and click on **Select** button next to the **Parameter** field. This time click on **Diskspace on / (used)** item in the **Description** column. For this item we should change the color, so choose something that could represent used disk space.

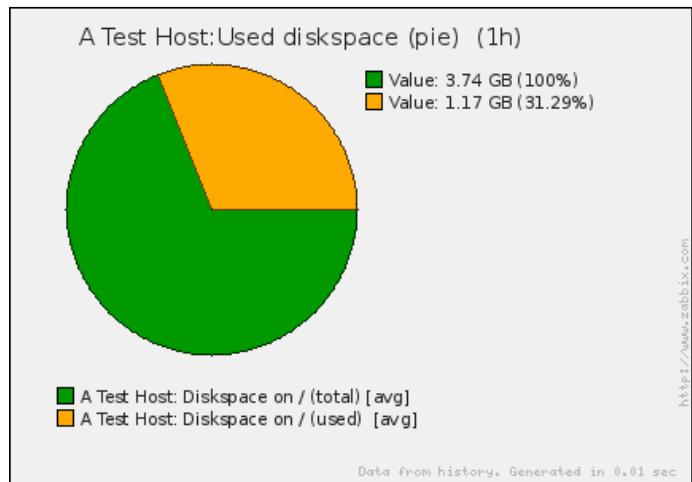


Visualizing the Data

For this item we are leaving **Type** field set to **Simple**, which should be done for all other items that are not graph total items. When you are done, click **Add**. We're done with adding items, so click **Save**. Back in **Monitoring | Graphs**, select **Used diskspace (pie)**.

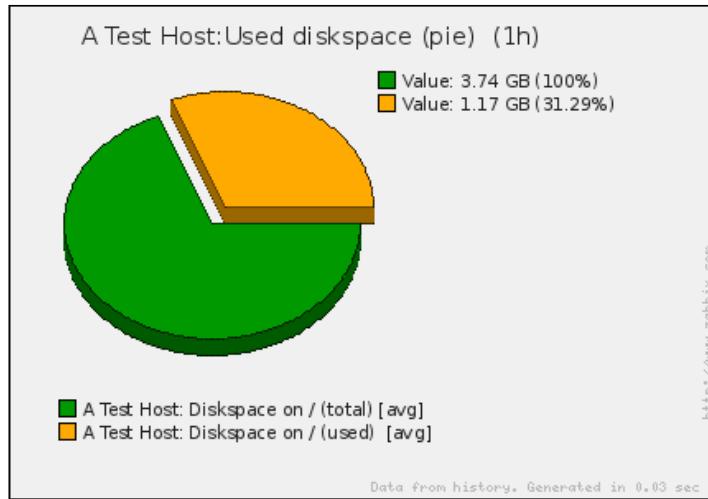


Great, it looks about right, except for the large, empty area on the right side. How can we get rid of that? Go to **Configuration | Hosts**, choose **Graphs** in the first dropdown and click on **Used diskspace (pie)** in the **Name** column. This time width and height controls will be useful. Change **Width** field to **430** and **Height** field to **300**. While we're here, also mark the checkbox next to the **Legend** and click **Save**. Let's check out whether it's any better in **Monitoring | Graphs** again.



It really is better, we got rid of the huge white area and the legend also helps. Pie graphs could also be useful for displaying of memory information—having the whole pie split into total, buffers, cached, and really used memory.

Let's try another change. Go to **Configuration | Hosts**, choose **Graphs** in the first dropdown and click on **Used diskspace (pie)**. Select **Exploded** in the **Graph type** dropdown and mark checkbox next to **3D view**. Save these changes and refresh the graph view in **Monitoring | Graphs**.



So for pie graphs you can combine options like 3D view and an exploded pie graph to create visual output to your liking.

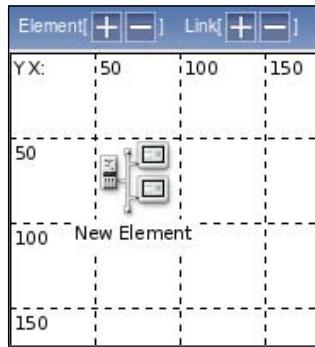
Maps

We have covered several ways of data visualization, which allow for quite a wide range of views. While the ability to place different things on a single graph allows to look at data and events in more context, sometimes you might want to have a broader view of things and how they are connected. Or maybe you need something shiny to show off.

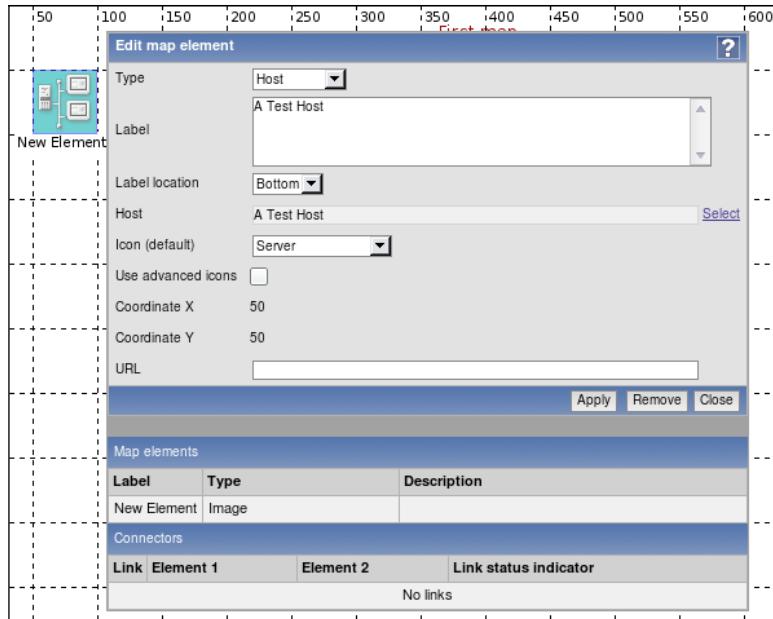
There's a functionality in Zabbix that allows one to create maps. While sometimes referred to as *network maps*, nothing prevents you from using these to map out anything you like. Before we start, make sure there are no active triggers for both servers—check that under **Monitoring | Triggers** and fix any problems you see.

Creating a map

Let's try to create a simple map now – navigate to **Configuration | Maps**, click **Create Map**. Enter **First map** in the **Name** field and click **Save**. Hey, was that all? Where can we actually configure the map? In the **Name** column, click on **First map**. Yes, now that's more of an editing interface. First we have to add something, so click on the **+** sign next to the **Element** caption at the top of the map. This adds some element at the upper-left corner of the map. The caption is slightly cropped by the map border though, and location itself isn't exactly great. To solve this, click and drag the icon elsewhere, somewhere around the cell at 50x50.



It still doesn't look too useful, so what to do with it now? Just click on it once. This opens up the element properties form, which allows you to set various details regarding the selected element. Notice how element itself is highlighted as well now. For a simple start we'll use hosts, so choose **Host** in the **Type** dropdown – notice how this changes the form slightly. Enter **A Test Host** in the **Label** textarea, then click **Select** next to the **Host** field. In the resulting pop up, select **Linux servers** in the **Group** dropdown, then click on **A Test Host**. Select **Server** in the **Icon (default)** dropdown. The result should look like this:



For a simple host that should be enough, so click **Apply**, then **Close** to remove the property editor. The map is regenerated to display the changes we made.



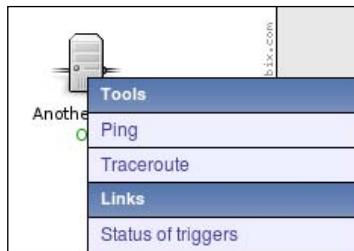
Our test host has no problems registered. A map with a single element isn't that exciting, so click + sign next to the **Element** caption again, then drag the new element around the cell at 450x50. Click it once and change its properties. Start by choosing **Host** in the **Type** dropdown, then enter **Another Host** for the **Label**, click on **Select** next to the **Host** field and choose **Another Host**. Change the **Icon (default)** dropdown to **Server**, then click **Apply**.

A map is not saved automatically – to do that click the **Save** button in the upper-right corner and dismiss the pop up (it does not matter in this case which option you choose as we are navigating to another section immediately afterwards).

Now is a good time to check what the map looks like, so go to **Monitoring | Maps** and make sure **First map** is selected in the dropdown. It should look quite nice, with the guidance lines removed, except for the large white area, like we had with the pie graph (and which is a reason why it won't be reproduced here). That calls for a fix, so get back to **Configuration | Maps** and click on **Edit** next to **First map**. Enter **600** in the **Width** field and **200** in the **Height** field, then click **Save**. Check **Monitoring | Maps** again.



That looks much better, and we verified that we can easily change map dimensions in case we need to add more elements to the map. So, what else does this display provide besides a nice view? Click on **Another Host** icon.



Here you have access to some scripts for **Ping** and **Traceroute**, and also a link to status of triggers page, which will be filtered to provide list of currently active triggers for this host.

 Middle clicking the host will take you directly to the status of triggers page.

We talked about using maps to see how things are connected. Before we explore that further, let's create basic testing infrastructure. On both "A Test Host" and "Another Host" execute:

```
$ touch /tmp/severity{1,2,3}
```

In the frontend, navigate to **Configuration | Hosts**, choose **Templates** in the first dropdown, then click on **Items**, next to **C_Template_Linux** and click on **Create Item** button. Enter **Link \$1** in the **Description** field and `vfs.file.exists[/tmp/severity1]` in the **Key** field, then click **Save**. Now clone this item (by clicking on it, then clicking on the **Clone** button) and create two more, changing the trailing number to "2" and "3" accordingly.



Do not forget to click **Clone** after opening item details, otherwise you will simply edit the existing item.

Verify that you have those three items set up correctly.

Link /tmp/severity1	Triggers (0)	<code>vfs.file.exists[/tmp/severity1]</code>
Link /tmp/severity2	Triggers (0)	<code>vfs.file.exists[/tmp/severity2]</code>
Link /tmp/severity3	Triggers (0)	<code>vfs.file.exists[/tmp/severity3]</code>

And now for the triggers – in the navigation bar click **Triggers** and click **Create Trigger** button. Enter **Severity 1 missing on {HOSTNAME}** in the **Name** field and `{C_Template_Linux:vfs.file.exists[/tmp/severity1].last(0)}=0` in the **Expression** field. Select **Warning** in the **Severity** dropdown, then click **Save**. Same as with items, clone this trigger twice and change the severity number in both fields (**Name** and **Expression**). Select **Average** for the second trigger and **High** for the third trigger in the **Severity** dropdown. The final three triggers should look like this:

Warning	Enabled	Severity 1 missing on C_Template_Linux	<code>(C_Template_Linux:vfs.file.exists[/tmp/severity1].last(0))=0</code>
Average	Enabled	Severity 2 missing on C_Template_Linux	<code>(C_Template_Linux:vfs.file.exists[/tmp/severity2].last(0))=0</code>
High	Enabled	Severity 3 missing on C_Template_Linux	<code>(C_Template_Linux:vfs.file.exists[/tmp/severity3].last(0))=0</code>

Linking map elements

We now have our testing environment in place, so let's see what functionality can we get from the map links. Go to **Configuration | Maps** and click on **First map**.

The triplet of items and triggers we created before can be used as network link problem indicators now. You can add connectors or links in maps that connect two elements. Additionally, it is possible to change connector properties depending on trigger state. Let's say you have a network link between two server rooms. You want the link to change appearance depending on connection state like this:

- **No problems** – green line;
- **High latency** – yellow line;

- **Connection problems for 3 minutes** – red line;
- **Connection problems for 10 minutes** – red, bold line.

The good news is, Zabbix supports such a configuration. We will use our three items and triggers to simulate each of these states. Let's try to add a link – click on the + button next to the **Link** caption at the top of the map. Now that didn't work. A pop up informs us that **Two elements should be selected**. How can we do that?

Click once on **A Test Host**, then hold down *Shift* (*Ctrl* works as well) and click on **Another Host**. This selects both hosts now – great. The property pop up changed as well to show properties that can be mass-changed for both elements in one go.

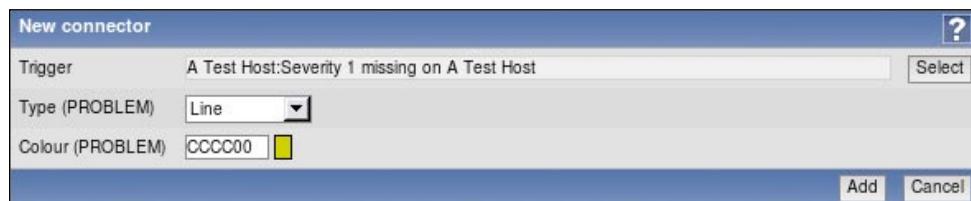


More than two elements on a map can be selected this way to perform mass changes on them.



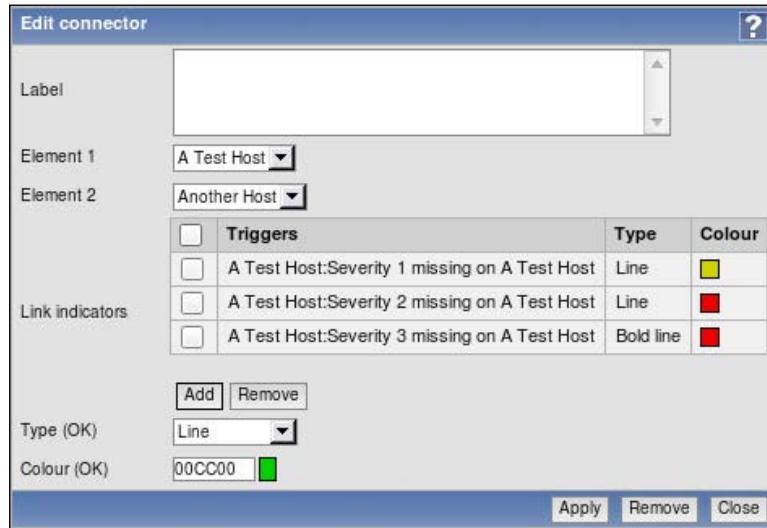
Click on the + button next to the **Link** caption again. Notice how in the property editor's **Connectors** section a new link has appeared – click on it. The map is also regenerated to show the new connector, which by default is blue.

First, let's define what the connector should look like when there are no problems. For **Colour (OK)** field, click on the color picker and choose one of the green ones. We'll continue with defining conditions and their effect on the link, so click **Add** in the **Link indicators** section. In the resulting pop up, click **Select** next to the **Trigger** field. Select **Linux servers** in the **Group** field and **A Test Host** in the **Host** dropdown, then click on **Severity 1 missing on A Test Host**. This trigger will simulate high latency, and we want the link to turn yellow in that case. Change the color to yellow in the color picker, then click **Add**.



Let's continue with condition configuration – again, click on **Add**, then **Select**. This time choose **Severity 2 missing on A Test Host**. Severity 2 will simulate connection loss for three minutes, and that one deserves a red line. Choose some shade of red in the color picker, then click **Add**.

Now follow the same procedure again – click **Add**, **Select**, choose **Severity 3 missing on A Test Host**. This was more serious, connection has been lost for 10 minutes, and it warrants a bold, red line. Choose **Bold line** in the **Type (PROBLEM)** dropdown and again, some shade of red in the color picker, then click **Add**. The final connector configuration should look similar to this:



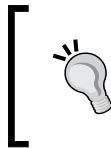
When you are done, click **Apply** in the connector area, then **Save**, and dismiss the pop up. The map preview already shows the line in green, but let's look at the map as it is intended, in the **Monitoring | Maps** section. Everything looks fine, both hosts show **OK**, and the link is green. Execute on "A Test Host":

```
$ rm /tmp/severity2
```

We just broke our connection to the remote datacenter for three minutes. Check the map again. You might have to wait for up to 30 seconds for the changes to show, then refresh it.



That's great, in a way. The link is shown as being down and one host has the active trigger listed. Notice how label text is off the map area, though. When creating your own maps you will want to think about possible values in advance so that they do not overflow map borders or overlap with each other.



You can also customize icons used in various states by enabling **Use advanced icons** option in element properties. In addition to the normal state, map elements in Zabbix can use different icons for problem – unknown, maintenance, and disabled states.

Note that there is no requirement for the associated trigger to be on a host that's connected to the link – it could even not be on the map at all. If you decided to draw a link between two hosts, the trigger could come from a completely different host. In that case both elements would show status as "OK", but the link would change the properties.

Some time has passed, let's see what happens when our link has been down for 10 minutes. Again, on "A Test Host" execute:

```
$ rm /tmp/severity3
```

Wait for some 30 seconds, refresh the map again.



To attract more attention from an operator, line is now bold, and instead of the trigger name a problem count is listed. Now, let's say our latency check is less frequent, and the trigger fires only now. On "A Test Host" execute:

```
$ rm /tmp/severity1
```

Wait for 30 seconds, then refresh the map. We should now see a yellow line... not quite. Actually, the bold red line is still there. Why so? The thing is, the order in which triggers fire does not matter – **trigger severity determines which style takes precedence**. We carefully set three different severities for our triggers, so there's no ambiguity when triggers fire. What happens if you add multiple triggers as status

indicators that have the same severity but different styles and they all fire? Well, don't. While you can technically create such a situation, that would make no sense. If you have multiple triggers of the same severity, just use identical styles for them. Feel free to experiment with removing and adding test files; the link should always be styled like specified for the attached active trigger with the highest severity.

There's no practical limit on the amount of status indicators, so you can easily add more levels of visual difference.

Further map customization

Let's find out some other things that can add nice touches to the map configuration.

Macros in labels

Open **Configuration | Maps**, click on the **First map**, and then click on the **A Test Host** icon. In the **Label** field, enter `{HOSTNAME} - {IPADDRESS}` and select **Top** in the **Label location** dropdown. Click on **Apply**, then save the map and take a look at it in the **Monitoring** section.



So some macros work even in element labels, and we can choose the label's position. Macros used here reduce configuration burden, another use could be the display of live data. For example, showing the number of currently connected users for each wireless hotspot.

We can also see that this host still has three problems in total, so execute on "A Test Host":

```
$ touch /tmp/severity{1,2,3}
```

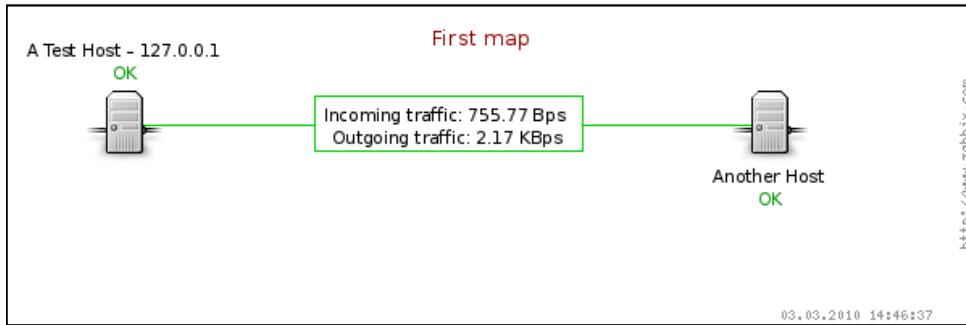
Link labels

What about links; maybe we can put labels on them as well? Actually, yes, since Zabbix 1.8 that is possible as well. In **Configuration | Maps**, click on the **First map** and then click on **A Test Host** icon. Click on **Link 1** in **Connectors** section to open its properties, then enter in the **Label** area:

```
Incoming traffic: {A Test Host:net.if.in[eth0].last(0)}
Outgoing traffic: {A Test Host:ifOutOctets[eth0].last(0)}
```

Visualizing the Data

We are mixing here both freeform text (you could label some link "Slow link", for example), and macros (in this case, referring to specific traffic items). Save the map, then take a look at it in the monitoring section.



As can be seen, map labels are rendered in a rectangle over the link, having the same color as the link itself. Both macros we used and multiline layout work nicely.

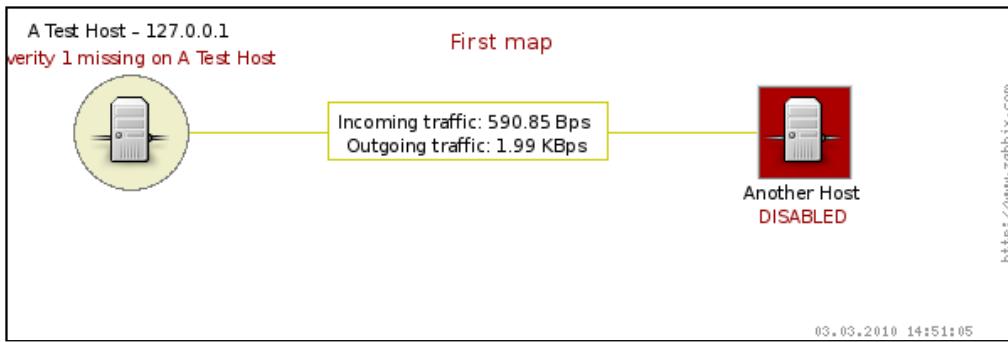
For a full list of supported macros in element labels see Zabbix manual.

Map element highlighting

Having problems listed below map elements is helpful, but if the map will be displayed on a large display that people will look at from a distance then it would be helpful to indicate problems in a more visible way. Zabbix maps have map level support for that, named icon highlighting. Open **Configuration | Maps**, click on **Edit** next to **First map** and mark checkbox **Icon highlighting**. This setting determines whether map elements receive additional visualization depending on their status. Click **Save**, then open **Configuration | Hosts**. Click on **Monitored** next to **Another Host** to disable it. Additionally, run on "A Test Host":

```
$ rm /tmp/severity1
```

After 30 seconds or so, verify what the map looks like in the monitoring section.



While the link has changed the color as expected (and label border with it), both hosts now have some sort of background. What does this mean?

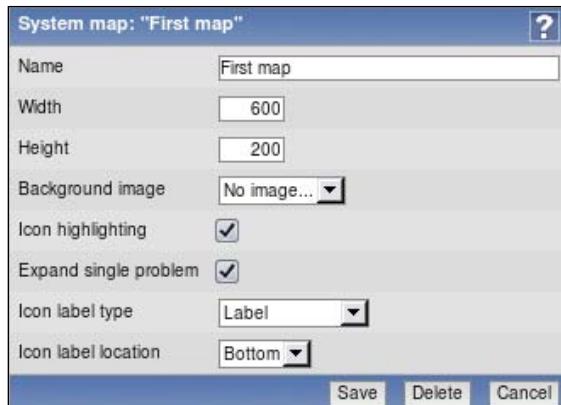
- The **round** background denotes trigger status. If any trigger is not in OK state, trigger with highest priority determines coloring of the circle.
- The **square** background denotes host status. Disabled hosts or hosts whose status cannot be determined receive this highlighting.

To return things to normal state, open **Configuration | Hosts**, click on **Not monitored** next to **Another Host**, then execute on "A Test Host":

```
$ touch /tmp/severity1
```

Global map options

When we created this map, we skipped them, but there are more global map options – global in the sense that they affect whole map (but not all maps). Go to **Configuration | Maps**, then click on **Edit** next to **First map** in the **Name** column.



We already covered the **Name**, **Width**, **Height**, and **Icon highlighting** fields, now we'll explore the rest.

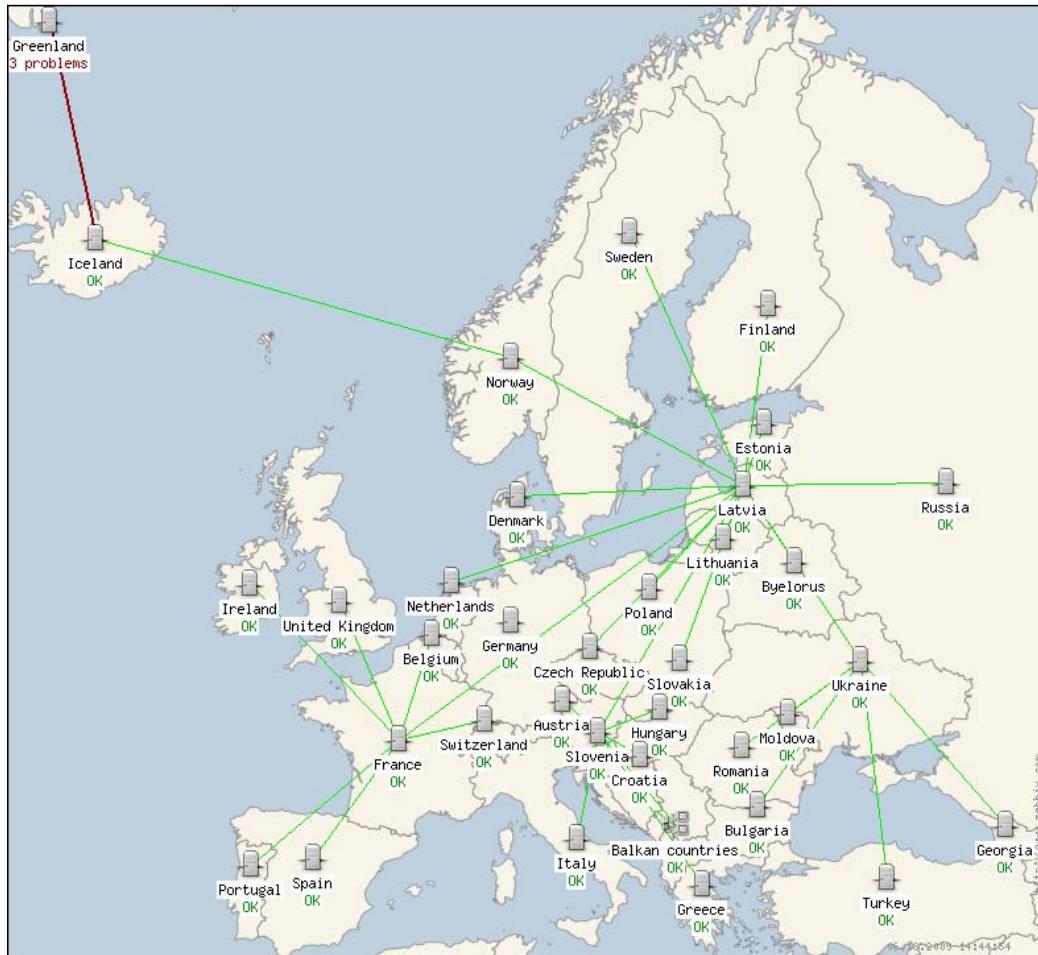
- **Background image:** This is getting interesting. It turns out we can place an image in the background.
- **Expand single problem:** It allows to control display of a single problem for a map element. In our case, when we had single problem trigger, it would not have displayed the trigger name, but text **1 Problem** instead.
- **Icon label type** sets whatever is used for labels. By default it's set to **label**, like we used. Other options are **IP address**, **Element name**, **Status only**, and **Nothing**, all of which are self-explanatory.

Visualizing the Data

- **Icon label location:** Allows us to specify the default label location, thus if you want it on the left for all or most elements, you can make configuration of the map easier.

Something like a country map might look nice as a background. Unfortunately, the dropdown offers no choices – what can we do? Open **Administration | General** and choose **Images** from the dropdown. Here we can edit all the small images used for map elements. But take a closer look at the upper-right corner – in the **Type** dropdown, you can select **Background**. That reveals the text **No images defined**, but there's also a **Create Image** button. Clicking this button will allow you to upload any image, so your imagination and artistic skills are the only limit.

Here's an example of what a geographical map might look like. We've used cc-by-sa open licensed map (CloudMade and openstreetmap.org):



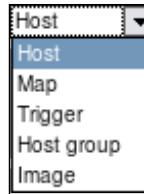
A geographical map is used as a background here, and different elements are interconnected. Notice the usage of the Balkan countries element. At this zoom level, there was no way to place individual countries in this region, thus this element would now work as a link to another, more detailed map.

The current icon set that is shipped with Zabbix is quite limited, but in this section you can also upload different icons to use with individual elements.

Available map elements

Hosts are not the only element type we could add to the map. In **Configuration | Maps**, click on **First map** in the **Name** column.

Click the **Another host** and expand **Type** dropdown in the element properties. We won't configure additional types right now, but let's look at what's available.



- **Host:** We already covered what a host is. A host gathers information on all associated triggers.
- **Map:** You can actually insert a link to another map. It will have an icon like all elements, and clicking it would open that map. This allows us to create interesting drilldown configurations. We could have a world map, then linked in continental maps, followed by country-level maps, city-level maps, server room level maps, rack-level maps, system-level maps, and on the other end we could actually expand to have a map with different planets and galaxies! Well, I got carried away. Of course, each level would have an appropriate map or schematic set as a background image.
- **Trigger:** This works very similar to a host, except only single trigger is linked. This way you can place a single trigger on the map that is not affected by other triggers on the system. In our nested maps scenario we could use triggers in the last display, placing server schematic in the background and adding individual triggers for fans, CPUs, and so on.

- **Host group:** Works like a host, except information about all hosts in a group is gathered. This can be handy for a higher-level overview, but it's especially nice in our above nested scenario where we could group all hosts by continents, countries, and cities, thus placing some icons on an upper-level map. For example, we could have per country host group elements placed on the global map. If we have enough room, that is.
- **Image:** This allows us to place an image on the map. The image could be something visual only, such as conditioner location in a server room, but it can also have an URL assigned, thus you can link to arbitrary objects.

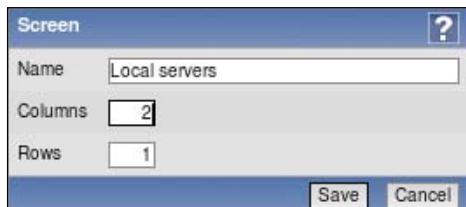
You should be able to create good looking and functional maps now. Before you start working on a larger map, I recommend you plan it out—doing large scale changes later in the process can be time consuming.

Compound elements

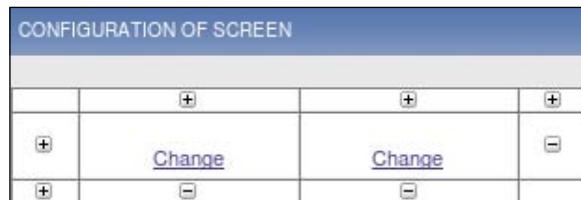
We looked at all single visualization elements before, now is the time to move forward. Compound elements (that have nothing to do with map elements) allow us to combine single elements and other sources to provide a more informative or good looking overview. We might want to see a map of our network at the same time as a graph of main outbound link, and maybe also a list of current problems

Screens

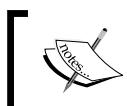
While we can place different information on a map, we can't, for example, place a graph below it, showing some statistic. For a more wide range of supported elements and a different combining approach in Zabbix we can use screens. Let's proceed with hacking one together—navigate to **Configuration | Screens** and click on the **Create Screen** button. Enter **Local servers** in the **Name** field and **2** in the **Columns** field. Hopefully we will be able to add more later if needed.



Click **Save**, then click **Local servers** in the **Name** column. We are presented with a fairly unimpressive view.

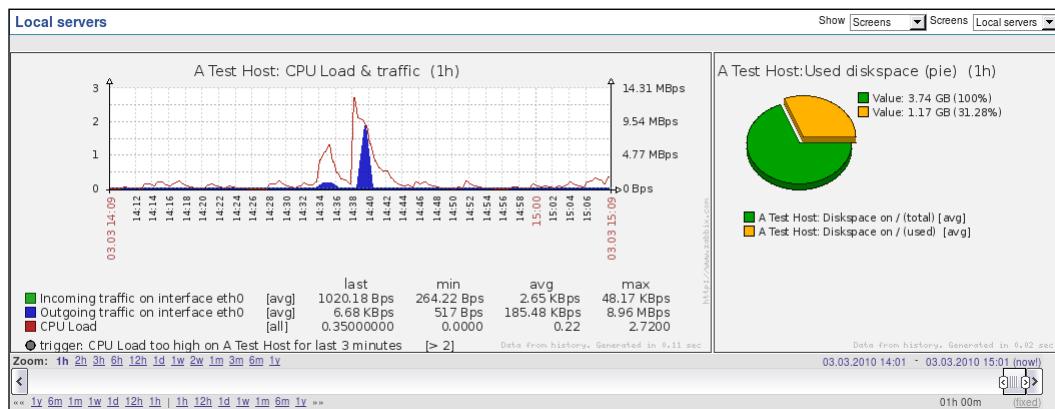


So it's up to us to spice it up. Click on the left-hand **Change** link, and we have an editing form replacing previous cell contents. Default resource type is graph, and we have some defined – click **Select** next to the **Graph name** field. In the upcoming window, make sure **A Test Host** is selected in the **Host** dropdown, then click on **CPU Load & traffic**. That's all we want to configure here, so click **Save**.



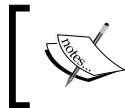
It is not required to save the screen explicitly, as it is with most other configuration sections. All changes done are immediately saved.

Now, click on the other **Change** link, then on **Select** next to the **Graph name** field. In the upcoming window click on **Used diskspace (pie)**. Remember, we tuned pie chart dimensions before? When inserting elements for screen, we override their configured dimensions. This time our pie chart has to share space with the other graph, so enter **350** in the **Width** field and **300** in the **Height** field, then click **Save**. While we can immediately see the result of our work here, let's look at it in all the glory – open **Monitoring | Screens**.



We now have both graphs displayed on a single page. But hey, look at the lower part of the screen—controls there look very much like the ones we used for graphs. And they are, using these controls it is possible to do the same things as with graphs, only for all of the screen elements. So **we can make all screen elements display data for a longer period of time or see what the situation was at some point in the past.**

Two graphs are nice, but before we talked about having a map and a graph on the same page—let's see how we can make that happen. Navigate to **Configuration | Screens** and click on **Local servers**. We want to add our map at the top of this screen, but we can see here that we created our map with two columns and single row, so we have to add more. Couldn't we do that in the general screen properties, using the fields when we created the screen? Of course we could, with one limitation. Increasing the column and row count there will only add new columns and rows to the right or at the bottom respectively. There is no way to insert rows and columns at arbitrary positions using that form. That's why we will use a different approach.



Reducing the column and row count is only possible from the right hand and bottom when using generic screen properties form. Any elements that are configured in the removed fields will also be removed.



Look at those + and - buttons at the top and at the left-hand side. They allow you to insert or remove columns and rows at arbitrary positions. While the layout might seem confusing at first, understanding few basic principles should allow you to use them effectively.

- Buttons at the top and bottom operate on columns
- Buttons to the left and to the right operate on rows
- + buttons add a column or a row *before* the column or row they are positioned at
- - buttons remove a column or a row they are at

In this case, we want to add another row at the top, so click the upper-left + icon in the first column. This adds a row with two columns, both having the **Change** link just like before. Click on the first **Change** link. It's not a graph we want to add, so choose **Map** in the **Resource** dropdown. Click on **Select** next to the **Parameter** field, then click **First map**. If we leave other parameters as they are, the map would appear on top of the left column. Having it centered above both columns would look better. That's what **Column span** option is—enter **2** in that field, then click **Save**. As can be immediately seen, this screen element now spans two column. This capability is not limited to maps, any element can span multiple columns or rows.

Dynamic screens

We now have a screen containing a network map and two graphs, showing data about "A Test Host". Now we should create a screen, showing data for "Another Host". We'll probably have to repeat all the steps we did for this one. That would be quite bad, especially for many hosts, wouldn't it? That's why there is a different, much easier approach.

Click the **CPU Load & traffic** graph in the screen configuration, look at the last parameter in there.



Let's find out what a dynamic item means – mark this option and click **Save**. While that seemingly did nothing, edit the other graph, mark **Dynamic item** checkbox and click **Save**. It's now time to check out the result in **Monitoring | Screens**. Look at the available dropdowns at the top of the screen.



As soon as we marked some elements as dynamic, we got the choice of other hosts. Let's check out how well this works. Select **Linux servers** in the **Group** dropdown and **Another Host** in the **Host** dropdown.



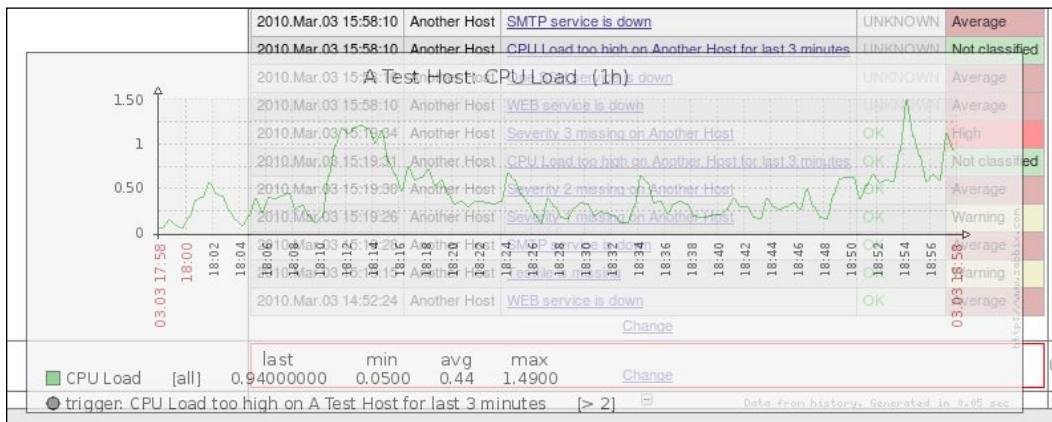
Visualizing the Data

Wonderful, elements marked as dynamic now show data from the selected host, while non-dynamic elements show the same data, no matter which host is selected.

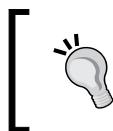
This is a nice, simple screen but there were many more available screen elements to choose from, so let's create another screen. Open **Configuration | Screens**, and click the **Create Screen** button. In the upcoming form, enter **Experimental screen** in the **Name** column and enter **2** for both **Columns** and **Rows** fields, then click **Save**. In the screen list, click on **Experimental screen** entry. As before, click on the **Change** link in the upper-left cell. In the **Resource** dropdown, choose **Simple graph**, then click on **Select** next to the **Parameter** field.

As we can see, all the graphs that are available without any manual configuration can also be added to a screen. Here, click on the **CPU Load** entry. In the **Width** field, enter **600**, then click **Save**. Click on the **Change** link in the upper-right cell. Choose **History of events** from the **Resource** dropdown, then click **Save**.

Well, suddenly our graph doesn't look that great any more, it should be taller to fit this layout. We could place it below the events list, but that would require deleting it and reconfiguring the lower-right cell. Well, not quite. Drag the graph to the lower-right cell. Notice how red border indicates the cell that dragged element will be dropped into. When correct cell is highlighted, release the mouse button.



The element (in this case, a graph) is moved from one cell to another, requiring no reconfiguration.

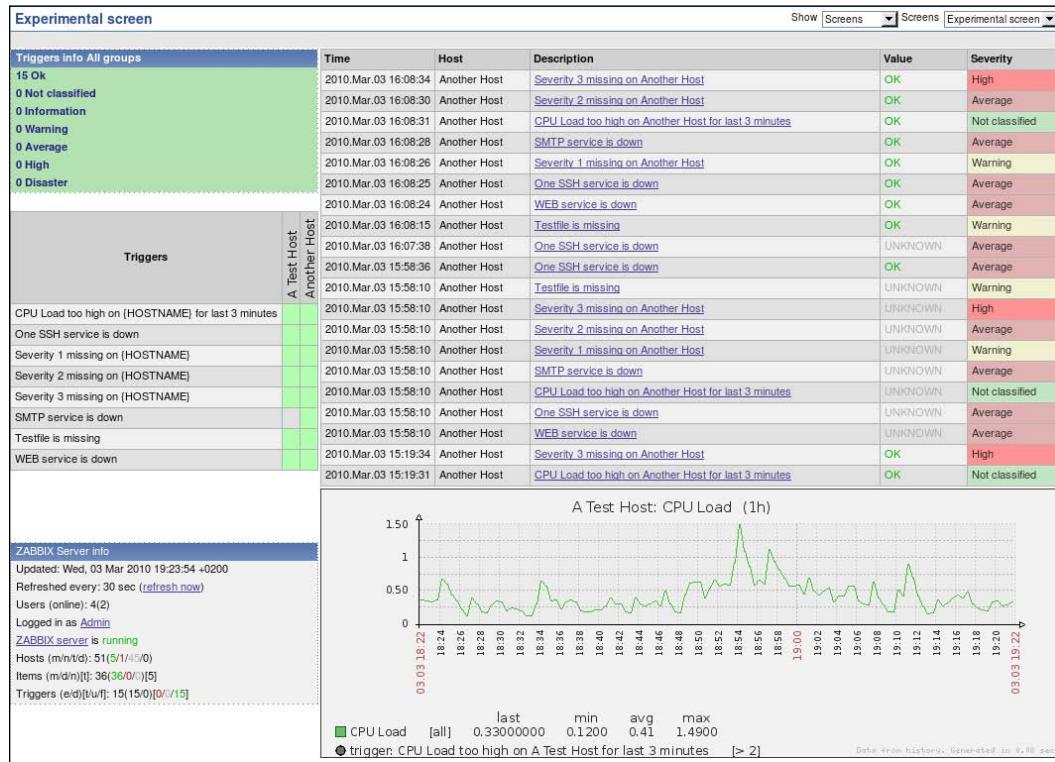


If you notice erratic behavior when dragging screen elements, this might be because you are using an older browser. Before deciding it is a bug, it's suggested that you try latest version of your browser and/or other browsers.

The upper-left cell is now empty, so click on **Change** there. Select **Triggers info** in the **Resource** dropdown, select **Vertical** in the **Style** dropdown, then click **Save**. This screen element provides us with a high-level information on trigger distribution by severity. Let's populate this screen even more now. Click the **Change** link in the lower-left corner. In the screen element configuration, select **Triggers overview** in the **Resource** dropdown, then click on **Select** next to the **Group** field. Choose **Linux servers** in the upcoming pop up. We have more triggers than hosts in this group – select **Top** in the **Hosts location** dropdown, then click **Save**. Well, the elements are misaligned again, right?

We'll try out some alignment work now. Click on the second + button from the top in the first column. This inserts a row before the second row. Drag the "Triggers overview" element (the one we added last) up one row, to the first cell in the row we just added. Click the **Change** link for the "History of events" element (upper-right cell), enter **20** in the **Show lines** field and **2** in the **Row span** field, click **Save**.

Our screen now looks quite nice, except that lower-left corner is empty. Click on **Change** in that cell, select **Server info** in the **Resource** dropdown, then click **Save**. The screen looks fairly well laid out now. Let's look at it in the **Monitoring | Screens** section – select **Experimental screen** in the **Screens** dropdown.



As we discovered, screens in Zabbix allow for very flexible visual layouts. You can choose to have a map, followed by more detailed graphs. Or you can have graphs of most important statistics for a group of servers, and a trigger summary at the top. Or any other combination – there are many more possible screen elements to be added.



As screens can contain lots of information, including many graphs, they can be performance intensive, especially if many users look at them at the same time.



Slide shows

We now have a couple of screens, but to switch between them manual interaction is required. While that's mostly acceptable for casual use, it would be hard to do if you wanted to display them on a large display for a helpdesk. Manual switching would soon get annoying even if you simply had Zabbix open on a secondary monitor all the time.

Another functionality comes to help – slide shows. Slide shows in Zabbix are simple to set up, so go to **Configuration | Screens**. Why screens? Take a look at the dropdown in the upper-right corner and choose **Slide shows** in there, then click on the **Slide show** button. Enter **First slide show** in the **Name** field, then click **Add** in the **Slides** section. Click **Select** in the **New slide** section and choose **Local servers**, then click **Add**. Notice how we didn't change the default value in the **Delay** field, but the **Delay** column shows **10**. That's because leaving the default value of "0" in that field uses the value from **Update interval** field above – that is the default value all new slides will receive.

Again, click **Add** in the **Slides** section, then **Select** in the **New slide** section, and choose **Experimental screen**. This time, enter "5" in the **Delay** field, then click **Add**.

The screenshot shows the 'Slide show' configuration dialog. At the top, the 'Name' field contains 'First slide show'. Below it, the 'Update interval (in sec)' dropdown is set to '10'. The 'Slides' section contains two rows:

	Screen	Delay	Sort
Slides	<input type="checkbox"/> Local servers	10	Down
	<input type="checkbox"/> Experimental screen	5	Up

At the bottom of the dialog are 'Add' and 'Delete selected' buttons, and 'Save' and 'Cancel' buttons.

Notice how non-default values in the **Delay** column are displayed in bold. Links in the last column allow you to reorder entries, which we won't do now, so just click **Save**.



If you want to add a single element to a slide show, like a map or graph, you will have to create a screen, containing this element only.



Now open **Monitoring | Screens** and select **Slide shows** in the **Show** dropdown. It starts plain, showing single screen and then it switches to the other screen after 10 seconds. Then back, after 5 seconds, and so the cycle continues.

We could show more screens, for example, a large high-level overview for 30 seconds. In that case cycle through the main server group screens, showing each one for five seconds.

Showing data on a big display

While visualization on an individual level is important, the real challenge comes when there's a need to create views for a large display, usually placed for helpdesk or technical operators to quickly identify problems. This poses several challenges.

Challenges

Displaying Zabbix on a large screen for many people to view poses several challenges that are rarely met in casual usage. Taking into account the display location, the experience level of the people who will be expected to look at it, and other factors can visibly shape your decisions on how to configure this aspect of information displaying.

Non-interactive display

In the majority of cases, data displayed on such a screen will be non-interactive. Such a requirement is posed because drilldown usually happens on individual workstations, leaving the main display accessible for others. Additionally, somebody could easily leave the main display in an unusable state, so usually no direct access is provided. This means that data placed on the display must not rely on the ability to view problem details. It should be enough for the level of technical support to gather the required knowledge.

Information overload

Having to place all information regarding the well-being of the infrastructure of an organization can result in a cluttered display, where too many details in a way too small font are crammed on the screen. This is the opposite of the previous challenge – you would have to decide what services are important and how to define each service. This will require you to be working closely with the people responding for those services so that correct dependency chains can be built (which will be the most often used method to simplify and reduce displayed data while still keeping it useful).

Both of these challenges can be solved with careful usage of screens and slide shows that display properly dependent statuses. Do not rely on slide shows too much – it can become annoying to wait for that slide to come by again because it was up for a few seconds only and now there are 10 more slides to cycle through.

Displaying a specific section automatically

For a central display there are some more requirements. It should open automatically upon boot and display the desired information, for example, some nice geographical map. While this might be achieved with some client side scripting, there's a much easier solution, which we have already explored.

For a reminder, open **Administration | Users** and select **Users** in the dropdown. Click on **monitoring_user** in the **Alias** column and look at two options, **Auto-login** and **URL (after login)**.

Auto-login	<input type="checkbox"/>
Auto-logout (min 90 seconds)	<input type="checkbox"/> 90
Refresh (in seconds)	30
Rows per page	50
URL (after login)	dashboard.php

If we marked the **Auto-login** option for some user that is used by such a display station, it would be enough to log in once, and that user would be logged in automatically upon each page access. This feature relies on browser cookies, so the browser used should support and store cookies. The **URL (after login)** option allows the user to immediately navigate to the specified page. So all that's left is that the display box should launch a browser upon bootup and point to the Zabbix installation, which should be trivial. When the box starts up, it will, without any manual intervention, open specified page (which will usually be a screen or slide show).

When displaying data on such large screens, explore available options and functionality carefully – maybe the latest data display is the most appropriate in some cases. When using trigger overview, evaluate the host/trigger relationship and choose which should be displayed on which axis.

Recent change flashing

As we observed before, if a trigger fires, it appears in the list with flashing PROBLEM. It flashes for 30 minutes, then it becomes static. If it now goes in OK state, it flashes OK for 30 minutes, then disappears from the list. While this behavior usually does not create any problems with casual use, on non-interactive displays problems can arise in some situations when a relatively large amount of triggers start flashing in the OK state. This pushes important triggers in the PROBLEM state off the screen. This behavior is easy to customize – take a look at the frontend directory, file `include/defines.inc.php`. This file contains two variables that allow to tweak the time period for trigger state blinking.

- `TRIGGER_BLINK_PERIOD`: This parameter controls trigger flashing in both OK and PROBLEM states and time in seconds is specified here.
- `TRIGGER_FALSE_PERIOD`: This parameter controls trigger flashing in OK state only and time in seconds is specified here.

It is possible to make all triggers blink for 15 minutes and OK state triggers for 5 minutes. Specifying "0" disables the corresponding flashing state for any of the periods. These parameters are global, flashing cannot be customized per user.

Summary

We have learned to create graphs of different types and how to customize them. This allows us to place multiple items in a single graph, change their visual characteristics, choose different graph types, modify graph y-axis scaling, and several other parameters.

Creation of additional visualization elements like network maps, screens, and slide shows also should not be a problem any more. We will be able to create nice looking network maps; whether they show single data center, or lots of locations spread out all over the world. With screens we will be able to choose what information to display at the same time either for large wall display, or a view to include all graphs showing how well a particular server is doing, which system administrators will occasionally refer to.

Still, sometimes you might be tasked with more specific report creation. Before looking for external solutions, it is worth exploring what other reporting capabilities Zabbix has built-in, and that's what we will do in the next chapter which is about bar graphs.

9

Creating Reports

We are now familiar with most of the functionality that is available in the **Monitoring** section and we know how to configure it to get the results we want. A slightly different angle of information is provided in the **Reports** section, which we will explore now. We will cover the following topics:

- Simple reports, including one on Zabbix itself, an availability report which displays availability percentages and provides high-level graphs and a report which shows which triggers have fired most often.
- Bar reports, which are the most sophisticated built-in data visualization tool in the Zabbix frontend. As such, they are also quite intimidating, thus we will create a couple of reports with all three different bar report types.

Simple reports

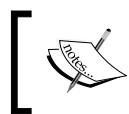
Simple reports, if we may call them this, are reports which require little or no configuration. They provide information on both monitored devices and the Zabbix server itself.

Status of Zabbix

Found at **Reports | Status of Zabbix**, this is the simplest report in Zabbix, and we already looked at it a bit. Let's refresh our memory about things that we can see there.

Parameter	Value	Details
Zabbix server is running	Yes	-
Number of hosts (monitored/not monitored/templates)	51	5 / 1 / 45
Number of items (monitored/disabled/not supported)	36	36 / 0 / 0
Number of triggers (monitored/not supported)	15	15 / 0 [0 / 0 / 15]
Title: Only items assigned to enabled hosts are counted		
Number of users (online)	2	
Required server performance, new values per second	0.7345	-

As the title promises, some general statistics on the current Zabbix installation are provided, like whether the Zabbix server is running and how large the installation is. We can see the amount of configured hosts; with breakdown of those that are monitored, not monitored, and templates, as well as similar statistics for items and triggers. Moving the mouse cursor over item and trigger rows will explain what is counted there. We can also see a count of the number of users configured and currently logged in.



This report is also displayed in **Monitoring | Dashboard**, but it is **only available for users of the type Zabbix Super Admin** as it contains detailed information on Zabbix's setup.



The last row, **Required server performance, new values per second**, is usually the most interesting one. This value is calculated by Zabbix and is not reflecting actual new values coming in, but new values that will come in according to the configuration. This is a good indicator of required server performance, especially for the disk subsystem. Each new value will require multiple queries to the database, and several writes, so as new values per second goes up, you might notice some load increase.

This report additionally shows any problems the web frontend might have discovered with your PHP configuration. These are displayed as additional rows for the table with a red font, for example, on the dashboard:

Status of Zabbix		
Parameter	Value	Details
Zabbix server is running	Yes	-
Number of hosts (monitored/not monitored/templates)	51	5 / 1 / 45
Number of items (monitored/disabled/not supported)	36	36 / 0 / 0
Number of triggers (enabled/disabled)[true/unknown/false]	15	15 / 0 [0 / 0 / 15]
Number of users (online)	4	2
Required server performance, new values per second	0.7345	-
PHP memory limit	64M	128M is a minimal PHP memory limitation
PHP max execution time	200	300 sec is a minimal limitation on execution time of PHP scripts

Updated: 13:00:02



This feature is available in Zabbix version 1.8.1 and newer.

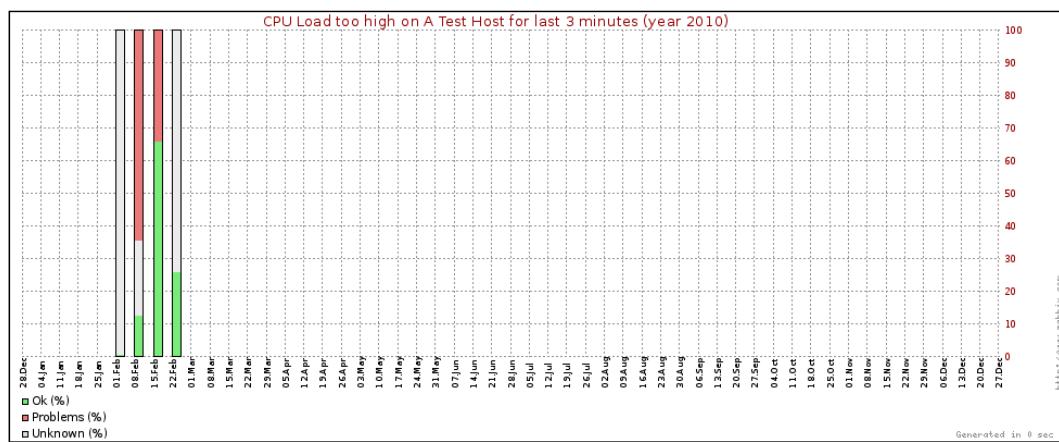


Availability report

It's time to move to the next report in the list, go to **Reports | Availability report**. This report deals with monitored data and allows to set filtering and some options. The default mode is to filter by host, and we can see data for all hosts. Expand the filter at the top of the list and choose **A Test Host** in the **Host** dropdown.

Name	Problems	Ok	Unknown	Graph
CPU Load too high on A Test Host for last 3 minutes	29.6408%	31.0467%	39.3125%	Show
One SSH service is down	0.0000%	83.4358%	16.5642%	Show
Severity 1 missing on A Test Host	0.0000%	0.0000%	0.0000%	Show
Severity 2 missing on A Test Host	0.0000%	0.0000%	0.0000%	Show
Severity 3 missing on A Test Host	0.0000%	0.0000%	0.0000%	Show
Testfile is missing	39.7588%	6.2118%	54.0293%	Show
WEB service is down	0.0000%	45.9726%	54.0274%	Show

Here we can see a report on what amount of time a trigger has spent in each one of the three states. This allows for an easy evaluation of availability for a particular service. Here we can see that this machine spent a fairly large amount of time with a high CPU load. The availability report also provides graphs – click the **Show** link in the **Graph** column next to the **CPU Load too high on A Test Host for last 3 minutes** trigger.



As we can see, this graph differs from all the graphs we have seen before. It displays data for the current year and each bar represents one week (again, starting with Monday). On each bar, the state of the trigger is displayed, with green being an "OK" state, red being a problem state and grey an "UNKNOWN" state. A high-level graph like this allows you to quickly evaluate how well a particular service has performed during a fixed period of time.

While the view provided with the host and host group filter is nice, it shows all triggers for all the matching hosts, which isn't that great if we have lots and lots of hosts with multiple triggers configured for each—like a server farm or network devices. There's also a nice support for such a scenario in Zabbix—go back to **Reports | Availability report** and in the **Mode** dropdown, select **By trigger template**. You should still have the filter open, which now looks notably more complex. Let's try to produce report about both of our servers. Select **Custom Templates** in the **Group** dropdown and **C_Template_Linux** in the **Host** dropdown. Now, in the **Trigger** dropdown, select **CPU Load too high on C_Template_Linux for last 3 minutes**.

The screenshot shows the Zabbix 'Availability report' filter interface. The 'Group' dropdown is set to 'Custom Templates'. The 'Host' dropdown is set to 'C_Template_Linux'. The 'Trigger' dropdown is set to 'CPU Load too high on C_Template_Linux for last 3 minutes'. The 'Filter Host group' dropdown is set to 'all'. The 'Period' section shows 'From' and 'Till' fields with date and time inputs. Below the filter area is a table with the following data:

Host	Name	Problems	Ok	Unknown	Graph
A Test Host	CPU Load too high on A Test Host for last 3 minutes	22.4223%	26.5596%	51.0181%	Show

Now a single trigger from all hosts, directly linked to the selected template, is shown. This allows to get a report on, for example, CPU overload on all of your switches in one group.

Notice how in both modes we can use the **Period** section to produce this report on any arbitrary period, which defaults to the 1st date of the current month. Additionally, if the template is linked to many different hosts, we can filter the results by host group in the **Filter Host group** dropdown.

Most busy triggers top 100

We saw one very simple and one slightly more complex report, now let's see another quite simple one—navigate to **Reports | Most busy triggers top 100**.

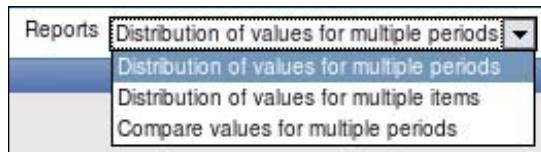
This report allows for little configuration, notably the dropdown at the upper-right corner allows to choose a period, starting from the current moment and it allows a choice of day, week, month, and year periods. A day is the default, but it might not produce too much output, so select **Week** in that dropdown.

REPORT			
Host	Trigger	Severity	Number of status changes
Another Host	One SSH service is down	Average	9
Another Host	CPU Load too high on Another Host for last 3 minutes	Not classified	7
Another Host	SMTP service is down	Average	6
Another Host	WEB service is down	Average	6
A Test Host	Severity 1 missing on A Test Host	Warning	5
Another Host	Severity 1 missing on Another Host	Warning	5
Another Host	Severity 2 missing on Another Host	Average	5
Another Host	Severity 3 missing on Another Host	High	5
Another Host	Testfile is missing	Warning	5
A Test Host	CPU Load too high on A Test Host for last 3 minutes	Not classified	3
A Test Host	Severity 2 missing on A Test Host	Average	3
A Test Host	Severity 3 missing on A Test Host	High	3

As a result, this report provides output of all triggers that have changed their state in the chosen period, sorted by the amount of status changes as listed in the **Number of status changes** column. This can allow you to find out what areas have the most frequent problems in your environment, or maybe even which triggers result in the biggest amount of false positives so that you can fine tune configuration.

Bar reports

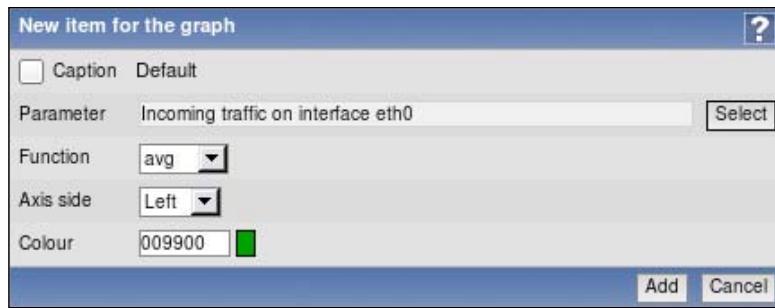
That's about it regarding the so called simple reports, and it's time to talk about bar reports now. They provide more customization options, but they can also take more time to set up. Let's see what reports are available at **Reports | Bar reports** – see the **Reports** dropdown at the upper-right corner.



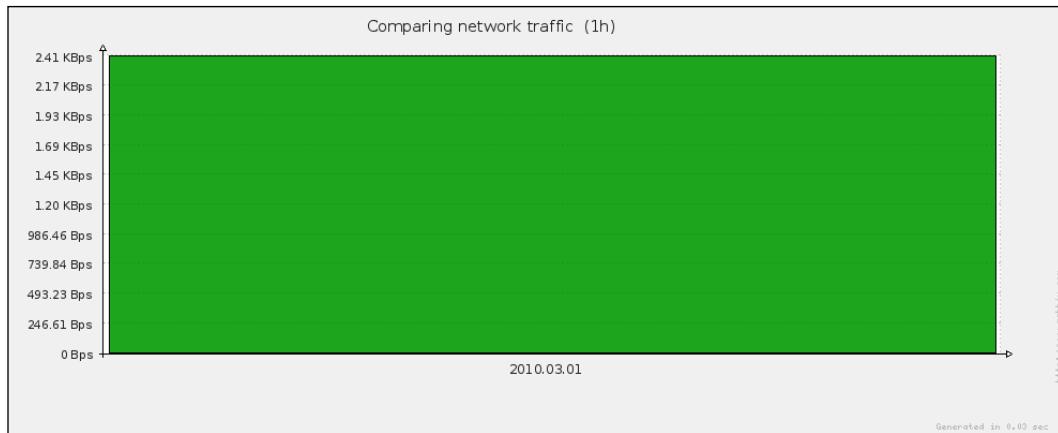
As we can see, three report types are available, but it's not trivial to guess by their names which is useful in which situation, so we'll try to apply each of them.

Distribution of values for multiple periods

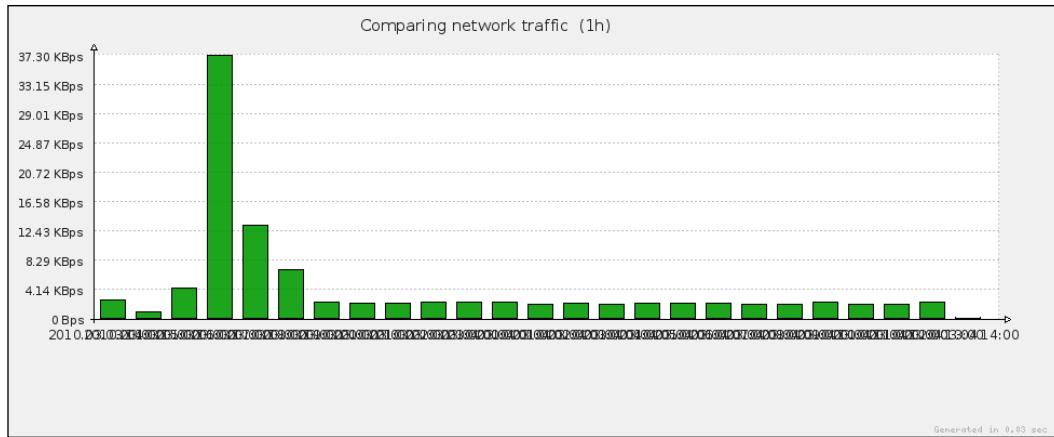
This is the first report and there's no need to switch to another, so dismiss the dropdown by clicking in the **Title** textfield and enter **Comparing network traffic** there. Let's start by getting some output from the report—click on **Add** in the **Items** section. This opens the item configuration dialog, in which you should click the **Select** button next to the **Parameter** field. In the upcoming list, find the **Incoming traffic on interface eth0** item next to **A Test Host** and click on it.



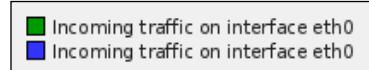
We won't change any other details, so click on **Add**. Getting a report as soon as possible was our main goal, right? Let's click on **Show**.



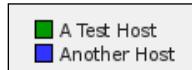
Now that's useless and ugly. There surely must be some way to make this report useful... Take a look at the **Scale** option—it is currently set to **Weekly**. If we compare that to the values in the **Period** section, they are set to one day, thus there's no wonder we got a single huge block. Looking at the **Scale** dropdown, we can see possible scales of hourly, daily, weekly, monthly, and yearly. Our displayed period is one day, so select **Hourly** in this dropdown, then click **Show** to refresh the report.



That's way better. Except maybe the x-axis labels... This again is a minor bug in Zabbix version 1.8.1. Now let's add another item – click **Add** in the **Items** section and again click on **Select** next to the **Parameter** field. Find the item **Incoming traffic on interface eth0**, but this time for **Another Host**, and click on it. Change the color for this item by clicking on the colored rectangle and choosing some shade of blue, then click **Add**. The report is immediately updated and now it looks more useful, we can compare amount of data transmitted for each hour for both hosts. Now, which host had which color? We should enable the legend here, mark the checkbox next to **Legend** title and click on **Show**. Let's look at the legend.



That's not helpful. As both items have the same name, they can't be distinguished in the legend – that has to be fixed. In the item list, click on the first item that has the green color chosen. Notice how this dialog looks very similar to the graph item configuration. We also have choice of axis, color, and function. But there's one additional feature, though – look at the first option, **Caption**. As we saw, the one inserted by default doesn't work that well here, so we will have to change it. Enter in the field **A Test Host**, then click **Save**. Now click on the bluish color item, and enter **Another Host** in thattextfield, then click **Save**. What does the legend look like now?

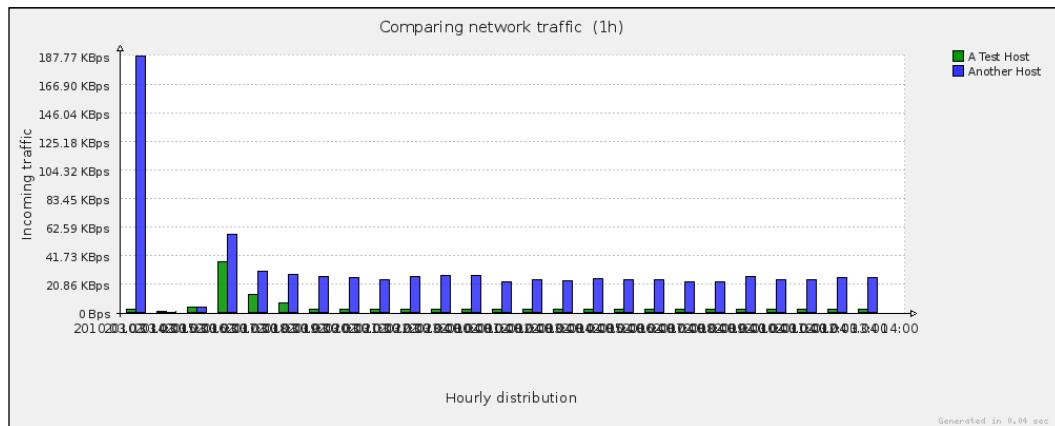


Creating Reports

Wonderful, that can finally be used to figure out which entry belongs to which host. We can still improve this report a bit. Enter **Hourly distribution** in the **X Label** field and **Incoming traffic** in the **Y Label** field. The final result should look like this:

Title	Comparing network traffic										
X Label	Hourly distribution										
Y Label	Incoming traffic										
Legend	<input checked="" type="checkbox"/>										
Scale	Hourly <input type="button" value="▼"/>										
Period	From 03 / 03 / 2010 14 : 36 <input type="button" value="..."/>										
	Till 04 / 03 / 2010 14 : 36 <input type="button" value="..."/>										
Items	<table border="1"><tr><td><input type="checkbox"/> A Test Host</td><td>A Test Host: Incoming traffic on interface eth0</td><td>avg</td><td>Left</td><td>█</td></tr><tr><td><input type="checkbox"/> Another Host</td><td>Another Host: Incoming traffic on interface eth0</td><td>avg</td><td>Left</td><td>█</td></tr></table>	<input type="checkbox"/> A Test Host	A Test Host: Incoming traffic on interface eth0	avg	Left	█	<input type="checkbox"/> Another Host	Another Host: Incoming traffic on interface eth0	avg	Left	█
<input type="checkbox"/> A Test Host	A Test Host: Incoming traffic on interface eth0	avg	Left	█							
<input type="checkbox"/> Another Host	Another Host: Incoming traffic on interface eth0	avg	Left	█							
	<input type="button" value="Add"/> <input type="button" value="Delete selected"/>										
	<input type="button" value="Show"/> <input type="button" value="Reset"/>										

When you are done, click **Show**. Our report appears below with labels for axis added.



With the help of this report it is quite easy to spot busy hours (or days, or weeks, or months, or years depending on what period and scale you choose) or compare data from several items. This is especially useful with many items, as custom graphs become unreadable with many items. Of course, there are practical limits to how many items can be compared with this bar report, but they are still much easier to read than normal graphs. Ability to choose scale allows for a layout that's better suited to spot important information or make a point. As there are

no limits to hosts that items come from, it's possible to create a report that examines several items from a single host or a single item from multiple hosts, or a mix of these in arbitrary combinations.

Distribution of values for multiple items

We found out what the first report does, and it's time to move to the second one. In the **Reports** dropdown select **Distribution of values for multiple items**. This report has a more simple form, at least it looks like that initially. Let's get to creating a new report – enter **Comparing CPU Load** in the **Title** field. Now we have to add periods and items. This will be confusing at first, but you'll get the idea when you see the result.

Let's start with adding some item. Click on **Add** in the **Items** section, then **Select** next to the **Parameter** field in the upcoming pop up. Choose **CPU Load for A Test Host**, then click **Add**. We should now have an item added like this:

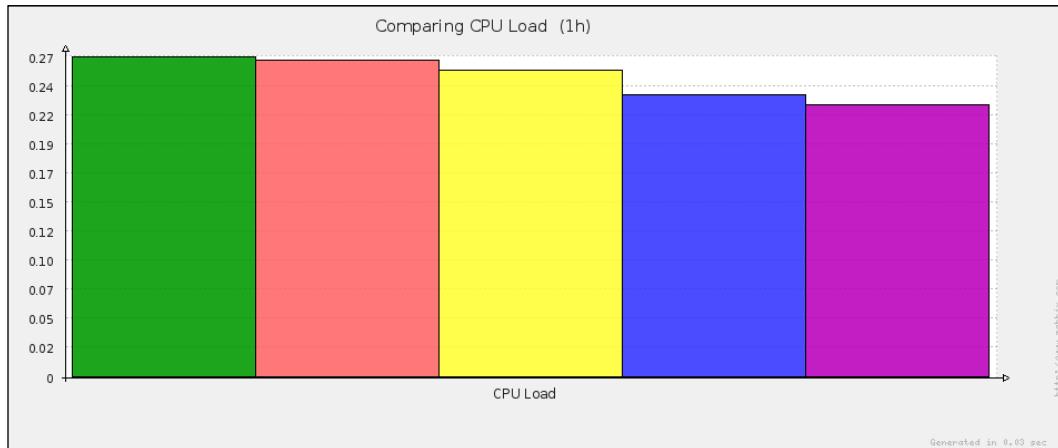
Items	<input type="checkbox"/>	CPU Load	A Test Host: CPU Load	avg
<input type="button" value="Add"/> <input type="button" value="Delete selected"/>				

Now is the time to move to the time periods. Our test installation probably doesn't have huge amounts of data gathered, so we'd be safer choosing smaller time periods like hours. Click on **Add** in the **Period** section, which opens period entering form. Now, set your current date in both date fields and set the latest passed full hour in the **Till** field time selection, and one hour before in the **From** field. If your current time is 14:50, that would mean you would use 14:00 for the **Till** field time value and 13:00 for the **From** field time value. It would look something like this:

Period	
<input type="checkbox"/> Caption	Default
From <input type="text" value="4"/> / <input type="text" value="03"/> / <input type="text" value="2010"/> <input type="text" value="14"/> : <input type="text" value="0"/> <input type="button" value="..."/>	
Till <input type="text" value="4"/> / <input type="text" value="03"/> / <input type="text" value="2010"/> <input type="text" value="15"/> : <input type="text" value="0"/> <input type="button" value="..."/>	
Colour <input type="text" value="009900"/> <input type="color" value="#009900"/>	
<input type="button" value="Add"/> <input type="button" value="Cancel"/>	

When you are done, click **Add**. We have to add more periods for this report, so click **Add** in the **Period** section again. In the period details form set current date just like before, and this time set one hour period just before the previously added period. If you previously added a period of 13:00-14:00, now add 12:00-13:00. Choose another color and click **Add**.

Now repeat this process three more times, each time setting the period one hour before and using a different color. Remember to use current date in both fields as well. Don't worry if you make a mistake. You can easily click any of the added periods and fix it. When you are finished, take a rest, then click **Show** button.



Oh shiny. While your output will look different depending on color choices and actual CPU loads, you'll notice how we can easily compare CPU loads for each defined time period. Sort of. We can observe here the same problem as with the previous report, we can't quite figure out which period is which, so mark checkbox next to the **Legend** title and click **Show**. Take a look at the legend

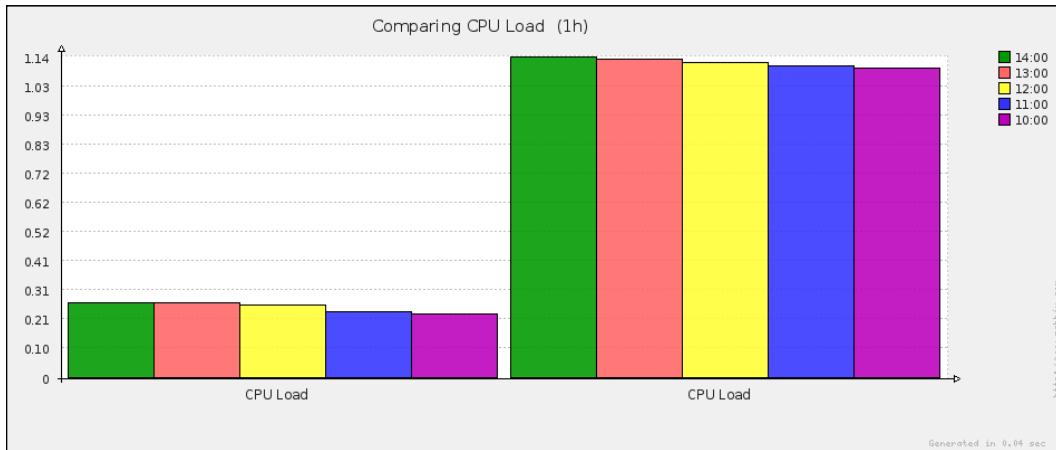
[Color Box]	04 Mar 2010 14:00:00 - 04 Mar 2010 15:00:00
[Color Box]	04 Mar 2010 13:00:00 - 04 Mar 2010 14:00:00
[Color Box]	04 Mar 2010 12:00:00 - 04 Mar 2010 13:00:00
[Color Box]	04 Mar 2010 11:00:00 - 04 Mar 2010 12:00:00
[Color Box]	04 Mar 2010 10:00:00 - 04 Mar 2010 11:00:00

That helped, now we can see which time period each bar represents. The legend is somewhat too verbose though. We know this is a single date and that each period is one hour, so it would be enough if each legend entry would simply list the starting hour. As with the previous report, we can customize captions, so click on the first entry in the **Period** section. Enter the starting hour in the **Caption** field (the one in the **From** field) for this particular period, for example, 13:00, then click **Update**. Now for some tedious work, do the same with all other periods.

[Color Box]	14:00
[Color Box]	13:00
[Color Box]	12:00
[Color Box]	11:00
[Color Box]	10:00

Now our legend is more readable and takes up considerably less space as well.

But we have only added a single item so far, let's add some more. Again, click on **Add** in the **Items** section and click on **Select** next to the **Parameter** field in the upcoming pop up. This time, choose **CPU Load** for **Another Host**, then click **Add**. The report is immediately updated and another block for the newly added item appears.



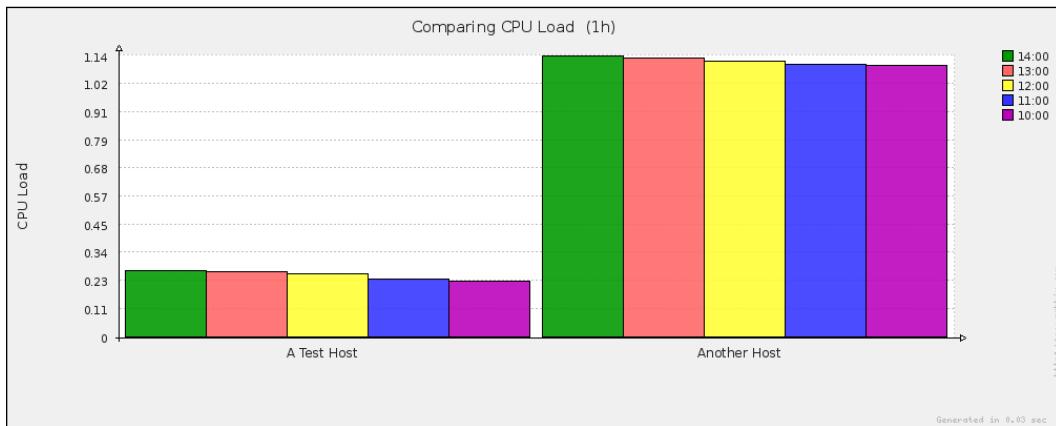
We can now see how the load compares during each period for each item (in this case, same item for different hosts). Oh, but we have again hit the same problem as before—the block titles are identical, so we don't even know which host is which. Luckily, for this report we can enter custom titles both for periods and for items. Click on the first item and enter **A Test Host** in the **Caption** field, then click **Save**. Click on the second item and enter **Another Host** in the **Caption** field, then click **Save**.

There is one more minor polish we could add; enter **CPU Load** in the **Y Label** textfield. The final configuration should look similar to this:

Title	Comparing CPU Load																												
X Label																													
Y Label	CPU Load																												
Legend	<input checked="" type="checkbox"/> <table border="1"> <tr><td><input type="checkbox"/></td><td>14:00</td><td>04 Mar 2010 14:00:00</td><td>04 Mar 2010 15:00:00</td><td>█</td></tr> <tr><td><input type="checkbox"/></td><td>13:00</td><td>04 Mar 2010 13:00:00</td><td>04 Mar 2010 14:00:00</td><td>█</td></tr> <tr><td><input type="checkbox"/></td><td>12:00</td><td>04 Mar 2010 12:00:00</td><td>04 Mar 2010 13:00:00</td><td>█</td></tr> <tr><td><input type="checkbox"/></td><td>11:00</td><td>04 Mar 2010 11:00:00</td><td>04 Mar 2010 12:00:00</td><td>█</td></tr> <tr><td><input type="checkbox"/></td><td>10:00</td><td>04 Mar 2010 10:00:00</td><td>04 Mar 2010 11:00:00</td><td>█</td></tr> </table>				<input type="checkbox"/>	14:00	04 Mar 2010 14:00:00	04 Mar 2010 15:00:00	█	<input type="checkbox"/>	13:00	04 Mar 2010 13:00:00	04 Mar 2010 14:00:00	█	<input type="checkbox"/>	12:00	04 Mar 2010 12:00:00	04 Mar 2010 13:00:00	█	<input type="checkbox"/>	11:00	04 Mar 2010 11:00:00	04 Mar 2010 12:00:00	█	<input type="checkbox"/>	10:00	04 Mar 2010 10:00:00	04 Mar 2010 11:00:00	█
<input type="checkbox"/>	14:00	04 Mar 2010 14:00:00	04 Mar 2010 15:00:00	█																									
<input type="checkbox"/>	13:00	04 Mar 2010 13:00:00	04 Mar 2010 14:00:00	█																									
<input type="checkbox"/>	12:00	04 Mar 2010 12:00:00	04 Mar 2010 13:00:00	█																									
<input type="checkbox"/>	11:00	04 Mar 2010 11:00:00	04 Mar 2010 12:00:00	█																									
<input type="checkbox"/>	10:00	04 Mar 2010 10:00:00	04 Mar 2010 11:00:00	█																									
Period	<input type="checkbox"/>	14:00	04 Mar 2010 14:00:00	04 Mar 2010 15:00:00																									
Items	<input type="checkbox"/>	A Test Host	A Test Host: CPU Load	avg																									
	<input type="checkbox"/>	Another Host	Another Host: CPU Load	avg																									
	<input type="checkbox"/>	Add	Delete selected																										
	<input type="checkbox"/>	Show	Reset																										

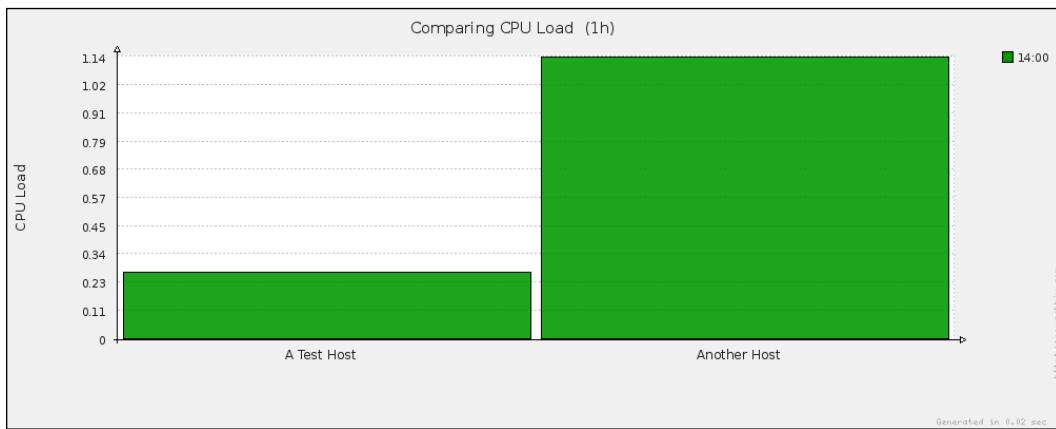
Creating Reports

If it does, click **Show**. Our report appears with all the options set and seems to be quite readable.



While our test installation does not have that much data, this report is useful for large scale comparisons of data—for example, comparing data with periods spanning a year. Smaller periods can be useful to extract seasonal or other periodic patterns. For example, one could create a report with 12 periods, you guessed it, one for each month and observe how seasonal changes impact data. Such a report is much more readable than common graph.

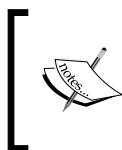
We saw how this report looks like with single item, but would it also be useful with single period? Let's find out. Select all but one checkbox in the **Period** section and click the **Delete selected** button just below them.



Well, those are huge blocks. But it should be clear now that single period configuration is also very useful. We could create a report on a compile farm, comparing the loads of all involved machines during some specific compilation job or compare the amount of Apache processes started on many web servers. Notice one more option appearing in report configuration, which you might have noticed disappear when we added second period – **Sort by**. We have a choice to sort either by name or by value. Name, obviously, will sort alphabetically, while value will sort by each item's value in ascending order. You'll be able to easily figure out which systems are overloaded and which are way too idle, for example.

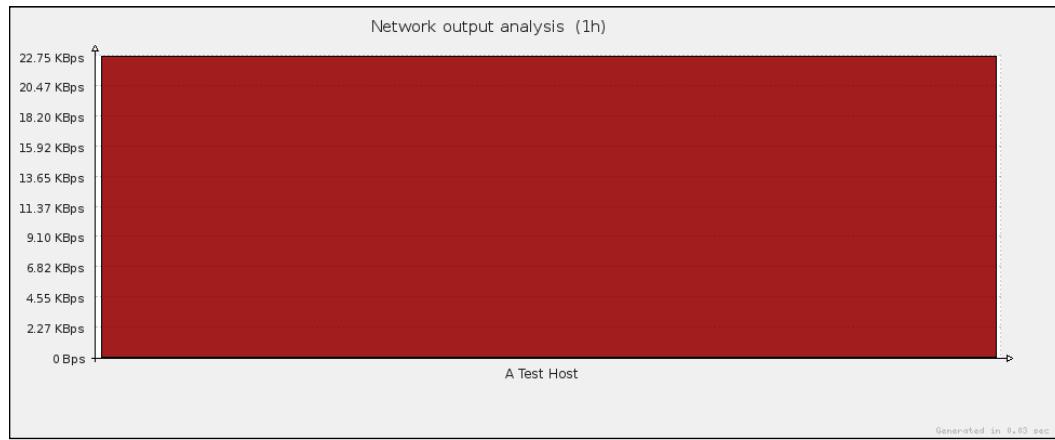
Comparing values for multiple periods

We have now gotten to the third and final report. In the **Reports** dropdown choose **Compare values for multiple periods**. We'll again start by getting some result ready quickly so that purpose of this report is easier to understand. Enter **Network output analysis** in the **Title** field. Now take a look at **Groups** and **Hosts** sections. We'll start with something simple first, so select **A Test Host** in the **Other Hosts** box that is located in the **Hosts** section, then click << button in the same section.



We can choose one or more host groups, and also one or more hosts. Choosing one does not restrict us to using only groups or hosts in addition, so we can easily report on few groups and some individual hosts in one go.

Now we need something to report on—click **Select** next to the **Item** field and click on **Outgoing traffic on interface eth0** for **A Test Host** in the upcoming form. We can try to obtain some output now, click **Show**.





Depending on which day you create this report with the default settings, you might see either one, or two bars – depending on whether the period contains a single week only or spans two weeks.

Let's try to figure out what this represents. To do this we'll take a closer look at report options.

A screenshot of a software interface showing report configuration options. It includes fields for 'Period' (From: 03/03/2010 15:27, Till: 04/03/2010 15:27), 'Scale' (Weekly), and 'Average by' (Daily).

Period	From <input type="text" value="03"/> / <input type="text" value="03"/> / <input type="text" value="2010"/> <input type="text" value="15"/> : <input type="text" value="27"/> <input type="button" value="..."/>
Till	<input type="text" value="04"/> / <input type="text" value="03"/> / <input type="text" value="2010"/> <input type="text" value="15"/> : <input type="text" value="27"/> <input type="button" value="..."/>
Scale	Weekly <input type="button" value="▼"/>
Average by	Daily <input type="button" value="▼"/>

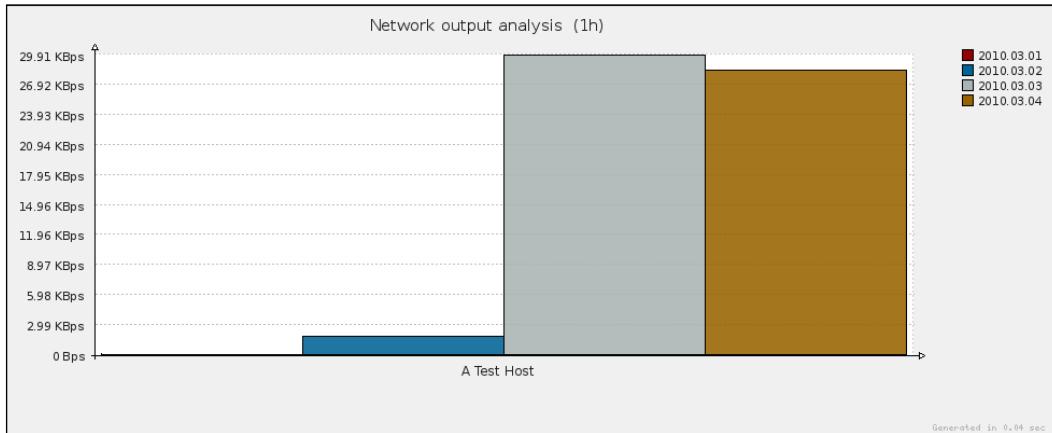
- **Period:** This is quite simple. It is set to be one full day, 24 hours.
- **Scale:** Familiar from the previous report, this determines how fine-grained the report will be. There will be one bar per scale period.
- **Average by:** This is a new option. While scale controls how many bars there will be, this option will determine what each column means. It will show average values per chosen period here.

Let's try to piece all these together. For a period of **Period** as many bars are drawn as **Scale** periods fit in, each representing the average per **Average by** period.

Still confused? Let's dissect our particular example. Let's put the first two options together. Our period is one day, 24 hours. Our scale is weekly, thus one bar will be drawn for each week that fits inside our period. That can be one only, right? Not quite. If these 24 hours pass the weekly border, we would get two bars. So what about the **Average by** option? It controls what actually goes into the report – this option is independent of the **Scale** option. No matter what scale we have chosen, averages will be calculated per the period chosen here.

What's the benefit? That might be more obvious if we created a slightly more meaningful report. While our testing system might not have too much data gathered yet, we might want to find out about traffic during the past few days. But we are interested in average hourly traffic, not daily.

Start by changing the period to be three days in the past, starting from now. In the **Scale** dropdown, choose **Daily**, and in the **Average by** dropdown choose **Hourly**. While we're here, mark the **Legend** checkbox. When done, click **Show**.



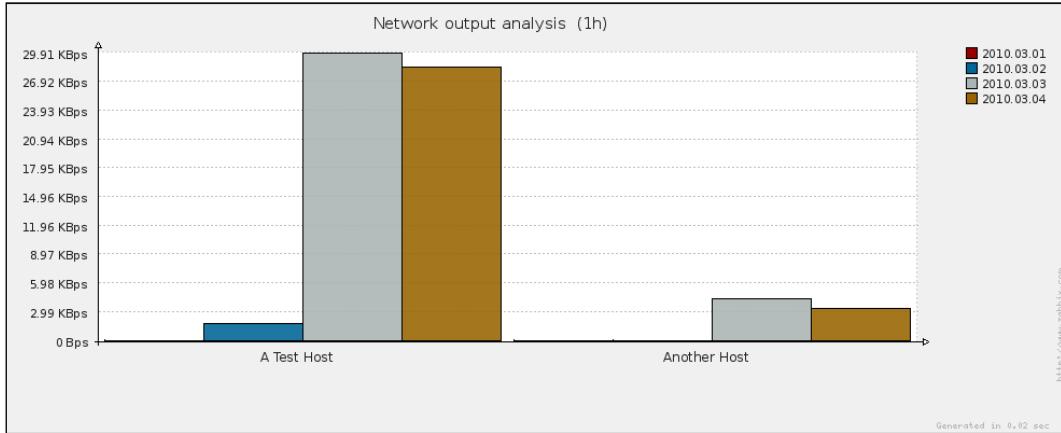
Now, why are we seeing four bars, if we chose three days only? Our period actually spans four days—a part of the current day, two full days and a part of the fourth day. We'll leave it as it is, but for other reports you probably will want to align period to scale starting points.

Looking at these bars, it's important to remember that each of them represents average **hourly** traffic for each of the displayed periods (days in this case), not a full traffic per period displayed. If billing, capacity planning, or other functionality refers to traffic per hour, we could create a report showing average traffic per hour for last few years and find out what exactly was it during a particular year easily.

Of course, nothing prevents you from setting averages to be calculated for the same or even longer period than the scale, although in most cases that would be pointless. Note that if we tuned the period to match the scale, we could reproduce part of the previous report's functionality, with the added ability to choose any averaging period.

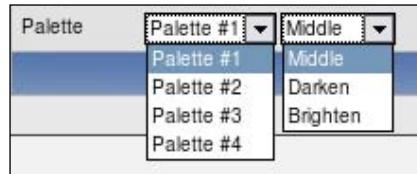
Creating Reports

Now, let's figure out how this report works for multiple hosts. Mark **Linux servers** in the **Other Groups for Groups** section, then click << button in the same section, then click **Show**.

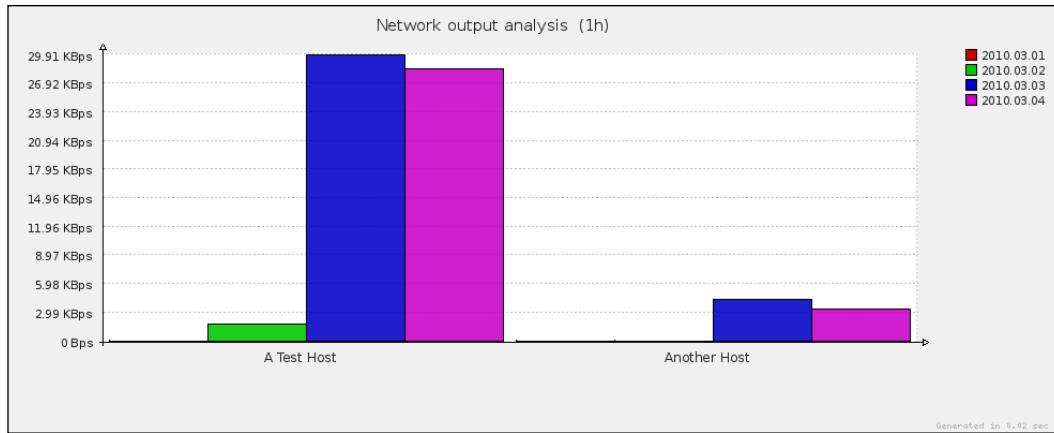


As can be seen here, the new host appears alongside the previous one in the report. The good news is, even though we have **A Test Host** selected both in the **Hosts** section and the **Groups** section, it is appearing only once in the report, thus we can safely create reports on hosts that are in multiple of the chosen groups.

There's a bit more we could configure for this report. While we can't customize coloring of each bar, look at the last option for this report.



The two palette dropdowns provide a choice for which palette to use, and whether coloring should be normal, slightly darkened, or brightened. There are four predefined palettes available, feel free to experiment with these combinations. For example, darkened **Palette #4** would look like this:



Note that you have to click **Show** whenever you change a palette or its brightness.

Summary

In this chapter, we looked at the basic and bar reports that are available in Zabbix. Basic reports allowed us to examine Zabbix state, figure out which triggers fire most often, and create basic availability reports with some charts.

The other category was bar reports, which required more configuration and were more complex, in turn providing very flexible functionality. Most of the bar report usefulness is evident for longer periods like several months or years, although figuring out the effect of some event on a smaller scale can be nice also, like reporting on what happened during slashdotting. With the flexibility of bar reports it can sometimes be hard to figure out which one should be used. We can try to simplify this problem by considering some typical questions that a particular report can answer:

Distribution of values for multiple periods

What amount of web server processes was started on each web server every week last month, grouped by week?

Distribution of values for multiple items

What was the system load for file server in December for the last 5 years? or

What was the system load for all file servers this December, sorted by load or host name?

Compare values for multiple periods

What was the average daily outgoing network traffic during the last three months for all web servers?

Obviously, there are many more reports that can be produced, but a short list like this should help you to choose correct report type without having to try out several reports.

10

Advanced Item Monitoring

Having set up passive and active Zabbix agent items, simple checks like ICMP ping or TCP service checks, or SNMP and IPMI checks, can we go further? Of course we can. Zabbix provides several more item types that are useful in different situations, and we now have to figure out what and when they will be useful.

Aggregate items

We played with stacked graphs, and that allowed us to display the total amount of item collections on a graph, but what if we would like to find out average load on a cluster, or total available disk space on a group of file servers, and not only display that on a graph, but also use it in triggers and notifications?

To find out what we can use in such a situation, go to **Configuration | Hosts**, select **Linux servers** in the **Group** dropdown and click on **Items** next to **A Test Host**, then click **Create Item**. Now we have to figure out what item type to use. Expand the **Type** dropdown and look for an entry named **Zabbix aggregate**. That's the one we need, so choose it and click **Select** next to the **Key** field. Currently the key is listed as `grpfunc`, but that's just a placeholder – click on it. We have to replace it with actual group key – one of `grpsum`, `grpmin`, `grpmax`, or `grpavg`. We'll calculate the average for several hosts, so change it to `grpavg`. This key, or group function, takes several parameters.

- **Group:** As the name says, here goes the host group name. Enter "Linux servers" for this parameter (with plain double quotes).
- **Key:** The key for the item to be used in calculations. Enter `system.cpu.load` here.
- **func:** A function used to retrieve data from individual hosts. While multiple functions are available, in this case we'll want to find out what the latest load is. Enter "last" for this field.

- **param:** A parameter for the function above, following the same rules as normal function parameters (specifying either seconds or value count, prefixed with #). The function we used, `last()`, ignored parameters, so enter "0" here.



All key parameters should be enclosed in plain double quotes.



For individual host data the following functions are supported:

Function	Details
<code>avg</code>	Average value
<code>count</code>	Number of values
<code>last</code>	Last value
<code>max</code>	Maximum value
<code>min</code>	Minimum value
<code>sum</code>	Sum of values

For aggregate items, we have two functions available. They are nested – first, the function specified as the `func` parameter gathers the required data from all hosts in the group. Then, `grpfunc` (`grpavg` in our case) calculates the final result from all the intermediate results retrieved by `func`.



There are no connections made to monitored devices.
Actually **all required data is retrieved from the database only**, thus correct items should be set up and collecting the data for the aggregate item to work.



The final item key should be `grpavg ["Linux servers", "system.cpu.load", "last", "0"]`. To finish item configuration, enter **Average system load for Linux servers** in the **Description** field and choose **Numeric (float)** in the **Type of information** dropdown. To reduce load, also set **Update interval** to **60** and **Keep history** to **7**. The final item configuration should look like this:

Item 'A Test Host:Average system load for Linux servers'

Host	A Test Host	Select
Description	Average system load for Linux servers	
Type	Zabbix aggregate	Select
Key	grpavg["Linux servers","system.cpu.load","last","0"]	
Type of information	Numeric (float)	Select
Units		
Use multiplier	Do not use	Select
Update interval (in sec)	60	
Flexible intervals (sec)	No flexible intervals	
New flexible interval	Delay 50	Period 1-7,00:00-23:59
	<input type="button" value="Add"/>	
Keep history (in days)	7	
Keep trends (in days)	365	
Status	Active	Select
Store value	As is	Select
New application		
Applications	-None-	
<input type="button" value="Save"/> <input type="button" value="Cancel"/>		

When you are done, click **Save**. Now to figure out what this gave us navigate to **Monitoring | Latest data**, make sure all hosts are shown and expand all entries. Look for three values—**CPU Load** both for **A Test Host** and **Another Host**, as well as **Average system load for Linux servers**.



You can enter "load" in the filter above the data and click on **Filter** to see only relevant entries.

Filter

Host	Description	Last check	Last value	Change	History
A Test Host	- other - (2 Items)				
	Average system load for Linux servers	04 Mar 2010 17:46:35	0.600000	+0.01	Graph
	CPU Load	04 Mar 2010 17:47:22	0.350000	+0.05	Graph
Another Host	- other - (1 Items)				
	CPU Load	04 Mar 2010 17:47:31	1.200000	-0.01	Graph

In this case, the system loads for individual hosts are 0.35 and 1.2, but the average is calculated as 0.6. If we calculate it manually, it should be 0.775. Why such a difference? Data for both items that the aggregate item depends on comes in at different intervals, and the aggregate value also is calculated at a slightly different time, thus, while the value itself is correct, it might not match the exact average of values seen at any given time. The aggregate value can easily be used in triggers or graphs.



As the key parameters indicate, aggregate item can be calculated for a host group – there is no way to pick individual hosts. Creating a new group is required if arbitrary hosts must have aggregate item calculated. We discussed other benefits from careful host group planning in *Chapter 5*.

We used the grpavg aggregate function to find out average load for a group of servers, but there are other functions:

Function	Details
grpmax	Maximal value is reported. One could find out what the maximum SQL queries per second are for a group of database servers.
grpmin	Minimal value is reported. The minimal free space for a group of fileservers could be determined.
grpsum	Values for the whole group are summed. Total amount of HTTP sessions could be calculated for a group of web servers.

Nothing limits the usage of aggregate items to servers. They can also be used on any class of devices, like calculating average CPU load for group of SNMP monitored switches.

External checks

All the check categories we explored before cover a very wide range of possible devices, but there's always one which doesn't play well with standard monitoring protocols, can't have agent installed and is buggy in general. A real life example would be UPS that provides temperature information on the web interface, but does not provide this data over SNMP.

For our test we could choose something to query on the local machine – like whether machine has a 64-bit CPU or not. On Linux machines this information can be discovered by looking at /proc/cpuinfo contents. On the Zabbix server, execute:

```
$ cat /proc/cpuinfo
```

This will produce detailed information on all processors installed.

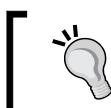
```
$ cat /proc/cpuinfo
processor      : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 5
model name    : Pentium II (Deschutes)
stepping       : 2
cpu MHz       : 349.261
cache size    : 512 KB
fdiv_bug      : no
hlt_bug       : no
foof_bug      : no
coma_bug      : no
fpu           : yes
fpu_exception : yes
cpuid level   : 2
wp            : yes
flags          : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
                  pat pse36 mmx fxsr
bogomips      : 699.66
```

As we can see, this is a fairly old Pentium CPU, which does not have 64-bit capabilities, though the one you have might. The line we are interested in is `flags`. Let's look at example of this line for a more recent CPU.

```
flags          : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
                  pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt lm 3dnowext
                  3dnow pnpi cmp_legacy
```

This unit has considerably more flags, but the flag that denotes 64-bit CPUs is `lm`, which stands for *Long mode*. We can also find out other information from this line alone, like the CPU being produced by AMD because of the `3dnow` flag—which represents an AMD specific instruction set.

Now we have to find some way to pass this information to Zabbix. While it is possible to pass the whole contents of `/proc/cpuinfo` to Zabbix and then look for the desired string, it would be quite inefficient—the Zabbix database would hold information we are not interested in and additional processing would also be required. Instead, we could create a script that would return "0" if the CPU is 32-bit and "1" if it's 64 bit.



On other architectures like PPC and other operating systems you should create different scripts.

But where do we place a script like that? Let's look it up in `/etc/zabbix/zabbix_server.conf` – open it as root and find the option named `ExternalScripts`.

```
# ExternalScripts=/etc/zabbix/externalscripts
```

By default it's commented out and points to a directory in `/etc`. For our situation, change it to the following:

```
ExternalScripts=/home/zabbix/bin
```

When done, restart the Zabbix server. Now to the script itself. As root, execute:

```
# touch /home/zabbix/bin/64bit
# chmod 755 /home/zabbix/bin/64bit
```

Still as root, edit this file to have these contents:

```
#!/bin/bash
[[ "$(grep lm /proc/cpuinfo)" ]] && echo 1 || 0
```

This bash specific one-liner will look for the string `lm` in `/proc/cpuinfo` and print "1" if found, and "0" if not found. You can test it right away to see whether the outcome is as expected. The script is ready, but to receive data we must create item in the Zabbix frontend – open **Configuration | Hosts**, make sure **Linux servers** is selected in the **Group** field, click on **Items** next to **A Test Host**, then click **Create Item**. Enter **CPU architecture** in the **Description** field (but we'll know that this will only differ between 32-bit and 64-bit for x86 architecture, at least for now), and then select **External check** in the **Type** field.

What about key, what should we enter in that field? For external scripts we have to use script name as the key – enter `64bit[]` in the key field.



The square brackets after the script name are mandatory in versions before 1.8.1, even if no parameters are used. They can be omitted in newer versions.



Change **Update interval** to **60**. While that is too low for an item like this (CPU architecture doesn't change too often), it will be useful for our testing. Also change **Keep history** to **7** and click **Save**. Observe the result in the frontend, **Monitoring | Latest data** (reset the filter, if you have used it before).

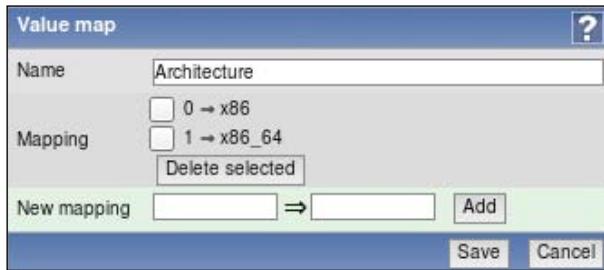
Host ▾	+ Description	Last check	Last value
A Test Host	[- other - (1 items)]		
	CPU architecture	05 Mar 2010 11:21:36	0

At least on this old Pentium machine it works correctly. But there are two problems with this approach—displaying a single number on the frontend isn't too intuitive, and a check like this can check only a single parameter on a single machine per script. Let's try to find out how we could improve both of these drawbacks.

Navigate back to **Configuration | Hosts**, click on **Items** next to **A Test Host**, then click on **CPU architecture**. In the item editing form, click on the link next to the **Show value throw map** field, then click **Create value map**. We already configured value mapping before, and it will be handy in this situation as well. Enter **Architecture** in the **Name** field and add two mappings:

- 0: x86
- 1: x86_64

The form should look like this:



If it does, click **Save**. Close the value mapping window and refresh the item editing form so that our new mapping appears in the dropdown. Choose **Architecture** in the **Show value throw map** dropdown. Notice how we left the parameters empty—now change the **Key** field contents to read `64bit [some parameters]`. When done, click **Save**. Results of our first change should be immediately visible in the frontend—look at **Monitoring | Latest data** again.

Host	Description	Last check	Last value
A Test Host	- other - (1 Items)		
	CPU architecture	05 Mar 2010 11:26:36	x86 (0)

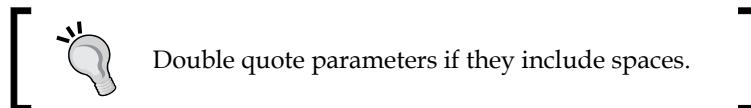
The results from this item should now be much easier to interpret in the frontend. To figure out how we can easily use external scripts for multiple hosts and items we'll have to edit our script a bit. Open `/home/zabbix/bin/64bit` as root in your favorite editor, and modify it to include the following line:

```
#!/bin/bash
echo "$@" >> /home/zabbix/external_script.log
[[ "$(grep lm /proc/cpuinfo)" ]] && echo 1 || echo 0
```

We didn't change the functionality of the script, but we now output all parameters passed to it to the file `/home/zabbix/external_script.log`. Wait a minute or so and check out the contents of this file.

```
127.0.0.1 some parameters
```

We can conclude that Zabbix passes the host IP as the first parameter to the script, and everything we specify as parameters to the key as further positional parameters. This allows us to use a single script for multiple hosts and multiple checks to perform against each host.



There's one problem, though. **External checks can be resource intensive**. It is suggested to use them only as a last resort when all other options to gather the information have failed. In general, you should keep any external checks (be they scripts or directly called binaries) lightweight and fast. Let's find out what happens if the check is slow for some reason.

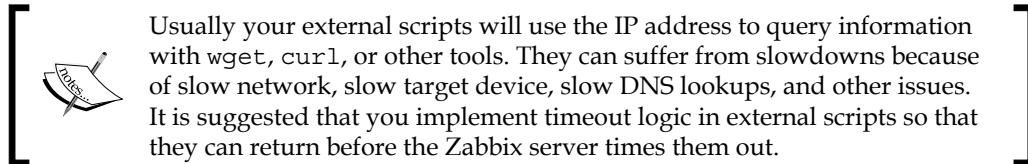
Open `/home/zabbix/bin/64bit` as root and replace the logging line (second one):

```
#!/bin/bash
sleep 6
[[ "$(grep lm /proc/cpuinfo)" ]] && echo 1 || echo 0
```

Wait for a minute, then see **Configuration | Hosts** in the frontend. Click on **Items** next to **A Test Host** and look at the **CPU architecture** item. There's an error icon displayed – move your mouse cursor over it.

External check	Not supported -	X
Zabbix Script <code>/home/zabbix/bin/64bit</code> returned nothing.		✓

That can't be true, can it? Our script did return information, we just introduced a delay for that. Well, the Zabbix server doesn't like scripts tying it up and timeouts after approximately five seconds. As a result, external check scripts should return data almost immediately, otherwise Zabbix will mark those items as unsupported.



Now edit the script again and remove the `sleep` line.

User parameters

We created an external check to gather one metric, but if external checks are executed on the Zabbix server only, they can't be used easily for commands to be executed on remote machines (sans configuring remote shell access).

As they can also cause performance problems on the Zabbix server itself, do we have some functionality available that would solve these problems?

Well, of course we do. A very similar but at the same time quite different functionality is provided by agents – user parameters. Technically behaving the same, these checks are executed by Zabbix agent daemons on whatever machines they reside. Actually, the `1m` string check in the previous section would be better created as a user parameter, as the machine the check is performed on already has Zabbix agent. Let's try to set up some custom parameters.

Just getting it to work

First we'll make sure that we can return any value. User parameters are configured on the agent side – the agent daemon contains the key specification, which includes references to commands. On "A Test Host", as root open `/etc/zabbix/zabbix_agentd.conf` in your favorite text editor and go to the end of the file. There are already some example parameters configured, and a syntax explanation is available:

```
UserParameter=<key>, <shell command>
```

This means that we can freely choose the key name and command to be executed. It is suggested that you keep key names to lowercase alphanumeric characters and dots. For starters, simply uncomment the very first example user parameter:

```
UserParameter=system.test, who | wc -l
```

This check will simply count logged in users on the system. Save the file and restart Zabbix agent daemon. This has to be done after any changes to the configuration file are made. Now we have to create corresponding item in the frontend. Open **Configuration | Hosts**, make sure **Linux servers** is selected in **Group** dropdown and click on **Items** next to **A Test Host**, then click **Create Item**. Fill in these values:

- **Description:** Enter **Users logged in**
- **Type:** Select **Zabbix agent (active)**
- **Key:** Enter `system.test`
- **Update interval:** Enter **60**
- **Keep history:** Enter **7**

We are using an active item type with a user parameter – as a user parameter can tie up server connections if it does not return very quickly, this is the suggested item type for user parameters. Notice how we used the exactly same key name as specified in the agent configuration file. When you are done, click **Save**.

Now check **Monitoring | Latest data**. As this is active item, we might have to wait for the agent to request item list from server, then return the data, which can take up to two minutes in addition to server updating its cache in one minute. Sooner or later the data will appear:

Users logged in	05 Mar 2010 11:38:56	1
-----------------	----------------------	---

We have gotten a basic user parameter to work, but this one replicates the existing Zabbix agent item, thus isn't that useful. The biggest benefit provided by user parameters is the ability to monitor virtually anything, even things that are not natively supported by the Zabbix agent, so let's try some slightly more advanced metrics.

Querying data that Zabbix agent does not support

One thing we might be interested in is the amount of open TCP connections. We can get this data using the `netstat` command. Execute on the Zabbix server:

```
$ netstat -t
```

The `-t` switch tells `netstat` to list TCP connections only. As a result, we get a list of connections (trimmed here).

Active Internet connections (w/o servers)			
Proto	Recv-Q	Send-Q	Local Address
tcp	0	0	localhost:10050
tcp	0	0	localhost:10051
tcp	0	0	localhost:10050

To get the amount of connections we'll use the following string:

```
netstat -nt | grep ^tcp | wc -l
```

Here, `grep` first filters out connection lines, and `wc` utility is used to count them. We could have used many other approaches, including grepping out the header, outputting only lines starting with the third line with `sed` – but this method provides slight performance increase over those methods, and we want our user parameters to be as fast as possible. Additionally, the `-n` flag is passed to `netstat`, which instructs it to perform no resolving on hosts, thus giving another performance boost.

As root, edit `/etc/zabbix/zabbix_agentd.conf` and add the following line at the end of the file:

```
UserParameter=net.conn,netstat -nt | grep ^tcp | wc -l
```

Don't forget to add the trailing newline. In the frontend, go to **Configuration** | **Hosts**, click on **Items** next to **A Test Host**, then click **Create Item** and fill in the following values:

- **Description:** Enter **Open connections**
- **Type:** Select **Zabbix agent (active)**
- **Key:** Enter `net.conn`
- **Update interval:** Enter **60**
- **Keep history:** Enter **7**

When you are done, click **Save**. Did you notice that we did not restart the agent daemon after modifying its configuration file? Do that now. Using such an ordering of events will give us values faster, because the agent queries active items list immediately after startup, and this way server already has item configured when agent is restarted. Feel free to check **Monitoring** | **Latest values**.

Open connections	05 Mar 2010 12:24:46	24
------------------	----------------------	----

[ If item turned into unsupported state before agent got its information, set it to **Monitored** in the item configuration section.]

Flexible user parameters

We now are gathering data on all open connections. But looking at the `netstat` output, we can see connections in different states like `TIME_WAIT` and `ESTABLISHED`.

<code>tcp</code>	<code>0</code>	<code>0 127.0.0.1:10050</code>	<code>127.0.0.1:9167</code>	<code>TIME_WAIT</code>
<code>tcp</code>	<code>0</code>	<code>704 10.1.1.99:22</code>	<code>10.1.1.105:60541</code>	<code>ESTABLISHED</code>

If we would like to monitor connections in different states, would we have to create a new user parameter for each? Fortunately, no. Zabbix supports so called flexible user parameters, which allow us to pass parameters to the command executed.

Again, edit `/etc/zabbix/zabbix_agentd.conf` as root and modify the user parameter line we added before to read:

```
UserParameter=net.conn[*],netstat -nt | grep ^tcp | grep "$1" | wc -l
```

We have made several changes here. First, the addition of `[*]` indicates that this user parameter itself accepts parameters. Second, adding the second `grep` statement allows us to use such passed parameters in the command.

All parameters we would use now for this key will be passed to the script – `$1` substituted for the first parameter, `$2` for the second, and so on. Note the usage of double quotes around `$1`. This way, if no parameter is passed, all output will go to `wc` and the result would be the same as without using `grep` at all.

Restart the agent to make it pick up the modified user parameter.

Back in the frontend **Configuration | Hosts**, click on **Items** next to A Test Host, and click on **Open connections** in the **Description** column, then click the **Clone** button at the bottom of the editing form. Change the following fields:

- **Description:** Open connections in `$1` state
- **Key:** `net.conn[TIME_WAIT]`

Click **Save**. Now click on **Open connections in TIME_WAIT state** in the **Description** column, click **Clone** and modify **Key** field to read `net.conn[ESTABLISHED]`, then click **Save**.



See `man netstat` for a full list of possible connection states.

Take a look at **Monitoring | Latest data**.

Open connections	05 Mar 2010 12:30:55	25
Open connections in TIME_WAIT state	05 Mar 2010 12:30:55	22
Open connections in ESTABLISHED state	05 Mar 2010 12:30:55	4

You'll probably notice that the values don't match – summing open connections in all states doesn't give the same number as all open connections. Firstly, remember that there are more connection states, so you'd have to add them all to get a complete picture. Secondly, as we saw before, all of these values are not retrieved simultaneously, thus one item grabs data, and a moment later the other comes in but the data has already changed slightly.



We are also counting all the connections that we create either by remotely connecting to the server, just running Zabbix server, or by other means.

We are now receiving values for various items, but we had to add a single user parameter only. Flexible user parameters allow us to return data based on many parameters. For example, we could provide additional functionality to our user parameter. If we make a simple modification like this:

```
UserParameter=net.conn[*],netstat -nt | grep "$1" | grep ^tcp | grep "$2" | wc -l
```

We added another grep command on the second parameter, again using double quotes to make sure the missing parameter won't break anything. Now we can use IP address as a second parameter to figure out amount of specific connection type to a specific host. In this case, item key might be `net.conn[TIME_WAIT,127.0.0.1]`.

Note that item parameter ordering (passing state first, IP second) in this case is completely arbitrary. We could swap them and get the same result, as we are just filtering the output by two strings with grep. If we were to swap them, the description would be slightly incorrect then, as we are using positional item key parameter references in it.

Level of the details monitored

There are almost unlimited combinations of what details one can monitor on some target. It is possible to monitor every single detailed parameter of a process, like detailed memory usage, the existence of PID files, and many more things, and it is also possible to simply check whether a process is running.

Sometimes single service can require multiple processes running, and it might be enough to monitor whether a certain category of processes is running as expected, trusting some other component to figure that out. One example could be Postfix, the e-mail server. Postfix runs several different processes, including `master`, `pickup`, `anvil`, `smtpd`, and others. While checks could be created against every individual process, often it would be enough to check whether the init script thinks that everything is fine.

We would need an init script that has `status` command support. As init scripts usually output a textual string like `Checking for service Postfix: running`, it would be better to return only a numeric value to Zabbix that would indicate service state. Common exit codes are "0" for success and non-zero if there is a problem. That means we could do something like:

```
/etc/init.d/postfix status > /dev/null 2>&1 || echo 1
```

That would call the init script, discard all `stdin` and `stderr` output (because we only want to return single number to Zabbix), and return "1" upon a non-successful exit code. That should work, right? There's only one huge problem – parameters should never return empty string, which is what would happen with such a check if Postfix was running. If the Zabbix server were to check such an item, it would assume the parameter is unsupported and deactivate it as a consequence. We could modify this string so that it becomes:

```
/etc/init.d/postfix status > /dev/null 2>&1 && echo 0 || echo 1
```

This would work very nicely, as now a boolean is returned and Zabbix always gets valid data. But there's a possibly better way. As the exit code is 0 for success and non-zero for problems, we could simply return that. While this would mean that we won't get nice boolean values only, we could still check for non-zero values in a trigger expression like this:

```
{hostname:item.last()}>0
```

As an added benefit, we might get a more detailed return message if the init script returns more detailed status with non-zero exit codes. As defined by the Linux Standard Base, the exit codes for status command are the following:

Code	Meaning
0	Program is running or service is OK
1	Program is dead and <code>/var/run</code> pid file exists
2	Program is dead and <code>/var/lock</code> lock file exists
3	Program is not running
4	Program or service status is unknown

There are several reserved ranges that might contain other codes, used by a specific application or distribution – those should be looked up in the corresponding documentation.

For such a case our user parameter command becomes even simpler, with the full string being:

```
UserParameter=service.status[*],/etc/init.d/$1 status > /dev/null  
2>&1; echo $?
```

We are simply returning the exit code to Zabbix. To make the output more user friendly, we'd definitely want to use value mapping. That way each return code would be accompanied on the frontend with an explanatory message like the above. Notice the use of `$1`. This way we can create a single user parameter and use it for any service we desire. For an item like that, the appropriate key would be `service.status[postfix]` or `service.status[nfs]`.

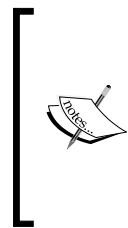
In open source land, multiple processes per single service are less common, but they are quite popular in proprietary software, in which case a trick like this greatly simplifies monitoring such services.

Environment trap

Let's try to find out what other interesting statistics we can gather this way. Looking at the Zabbix agent daemon configuration file, `/etc/zabbix/zabbix_agentd.conf`, we can see other user parameter examples; this time involving MySQL checks. While several look interesting, a good target is `mysql.qps`, queries per second – that should be interesting. There's one slight problem though. This parameter uses MySQL's internal query per second counter, and this counter isn't quite what most users would expect. That particular value is calculated for the whole startup time MySQL has, which means it's quite useful, though only for the first few minutes. As it is calculated for the whole uptime period, longer running MySQL instances have this number approaching some value and only slightly fluctuating. When graphed, the queries per second graph gets more and more flat as time progresses.

The flexibility of Zabbix allows us to use a different metric. Another item in the predefined MySQL parameters is `mysql.questions`, which returns the total amount of queries executed since the startup of the MySQL server. Let's see how can we get useful data out of that.

Configuration file suggests to use `-p` flag for `mysql` utility, but if we use that then the password would be visible for all other users on the machine. Even if there are no other users, that's still a bad practice.



On some platforms some versions of the MySQL client will mask the passed password. While that is a nice gesture from MySQL's developers, it won't work on all platforms and with all software, so such approach should be avoided just to make it a habit. The password in such a case is likely to be written to the shell history file, making it available to attackers even after software itself has been closed.

Having a user without a password also isn't acceptable. Fortunately, MySQL can read the password from a file which we could secure with permissions. Modify the `mysql.questions` user parameter line to look like this (make sure to uncomment it):

```
UserParameter=mysql.questions,mysqladmin -uzabbix status|cut -f4 -d":"|cut -f1 -d"S"
```

We'd better use the Zabbix user instead of root for such connections, although you could also create separate user without any access to databases. The MySQL client looks at multiple places for files that could contain the password, one of them being the `.my.cnf` file in the user's home directory. The Zabbix agentd runs as the user `zabbix`, so let's create such a file. On the Zabbix server execute as `zabbix` user:

```
$ touch ~zabbix/.my.cnf  
$ chmod 600 ~zabbix/.my.cnf
```

Now edit this file and enter the following content:

```
[client]  
password=<password>
```

Use the password that the Zabbix user has. You can remind yourself what it was by taking a look at `/etc/zabbix/zabbix_server.conf`.

Now restart the Zabbix agent daemon. Let's test the parameter first – a very good practice with user parameters, which have more things that could go wrong than built-in ones. From Zabbix server, execute:

```
$ telnet localhost 10050
```

Then enter the key we are interested in, `mysql.questions`, and press enter.



Alternatively, you can also use `zabbix_get` utility – `zabbix_get -s localhost -k mysql.questions`.

```
ZBXDZBX_NOTSUPPORTEDConnection closed by foreign host.
```

Well then, why is that happening? Let's test basic MySQL command – as `zabbix` user (you can become one by executing `su - zabbix` as root), run:

```
$ mysqladmin -uzabbix status | cut -f4 -d":" | cut -f1 -d"$"
```

But that works, a number is returned. Why is the Zabbix agent telling us it's not supported?

There are no environment variables set for user parameter commands. This includes several common variables, and one we are quite interested in – `HOME`. This variable is used by the MySQL client to determine where to look for the `.my.cnf` file. If the variable is missing, this file (and in turn, the password) can't be found. Does that mean we're doomed? Of course not, we wouldn't let such a minor problem stop us. We simply have to tell MySQL where to look for this file, and we can use a very simple method to do that. Edit `/etc/zabbix/zabbix_agentd.conf` and change the `mysql.questions` user parameter line to read:

```
UserParameter=mysql.questions,HOME=/home/zabbix mysqladmin -uzabbix  
status|cut -f4 -d":"|cut -f1 -d"S"
```

We are now setting a specific directory and that should allow the MySQL client to find the configuration file which specifies the password. Again, restart the Zabbix agent and then from the Zabbix server, run:

```
$ telnet localhost 10050
```

Enter mysql.questions and press enter.

```
ZBXD  
32596310 Connection closed by foreign host.
```

You'll see a different value, but finally we can see the item is working. But how do we get the number of queries per second out of this? Let's return to the frontend and figure that out. Navigate to **Configuration | Hosts**, click on **Items** next to **A Test Host**, then click **Create Item** button. Fill in these values:

- **Description:** Enter **MySQL queries per second**
- **Type:** Choose **Zabbix agent (active)**
- **Key:** Enter `mysql.questions`
- **Type of information:** Select **Numeric (float)**
- **Units:** Enter `qps`
- **Update interval:** Enter `60`
- **Keep history:** Enter `7`
- **Store value:** Choose **Delta (speed per second)**

When you are done, click **Save**. Again, we might have to wait for up to three minutes for any data to arrive. If you are impatient, feel free to restart the Zabbix agent daemon (but make sure to do so after server has refreshed its item cache).

MySQL queries per second	05 Mar 2010 12:56:01	6.48 qps
--------------------------	----------------------	----------

The data is coming in nicely and we can see that our test server isn't too overloaded. How did we get useful data out from that parameter? The biggest impact was provided by **Store value** option. The data provided by MySQL – it's just a counter. It just increases and storing it inside Zabbix as a **Delta (speed per second)** makes Zabbix calculate the time difference between the two values received, the delta between values themselves, and from that change in one second. Sounds familiar? Right, it's the same as with network traffic, only in this case we used an outside parameter, not one supported by the Zabbix agent directly.

Another important change is the information type **Numeric (float)**. Despite the fact that the counter itself is an integer, data calculated by Zabbix will be stored as a decimal, thus we had to change this option. Using qps for units we can more easily figure out what a value means, and it also will result in strings like "1.5 Kqps" used. While not a universal unit, it's still more convenient than raw numbers.

Things to remember about user parameters

We saw that the flexibility of user parameters is basically unlimited. Still, there might be cases when additional measures have to be applied.

Wrapper scripts

Commands to be executed can be specified in the Zabbix agent daemon configuration file on a single line only. Pushing whole scripts there can be very messy and sometimes it can be hard to figure out quotation. In such cases, a wrapper script has to be written. Such a script can be useful if parsing data requires more complex actions or if parsing out multiple different values cannot be easily done with flexible user parameters.

Another use case for wrapper scripts is gathering resource-intensive information in a single, crontab scheduled run, then saving it into temporary file(s) which can be queried individually later. This also applies to cases where the actual script is complex and takes a lot of time to run.

It is important to remember that using user parameters and custom scripts requires these being distributed on all monitored hosts—that involves the scripts themselves and changes to the Zabbix agent daemon's configuration file.



Distributing and managing such configuration on a large scale is best done with a dedicated system like Puppet (<http://reductivelabs.com/products/puppet/>).



This can soon become hard to manage. Various systems will require different user parameters, thus you'll either end up with a messy agent configuration file containing all of them, or a myriad of different combinations. There's a quite widespread feature to help with this problem – configuration file including. You can specify the inclusion of individual files by adding to `/etc/zabbix/zabbix_agentd.conf` entries like:

```
Include=/etc/zabbix/userparameters/zabbix_lm_sensors.conf  
Include=/etc/zabbix/userparameters/zabbix_md_raid.conf
```



Configuration of a management system like the aforementioned Puppet can also help with this problem by adding required entries depending on system role. Some Zabbix users also use another approach. By placing included configuration files in centralized file services, they essentially become manageable from the server without the need for configuration management system.

If such a file is missing, Zabbix will complain, but will still start up. Inclusions can be nested – you can include one file which in turn includes several other and so on.

It's also possible to include whole directories – in that case all files placed there will be used. This method allows other packages to place, for example, user parameter configuration in a specific directory, which will then be automatically used by Zabbix.

```
Include=/etc/zabbix/userparameters/
```

Individual packages would now only need to place files like `zabbix_1m_sensors.conf` or `zabbix_md_raid.conf` in the directory `/etc/zabbix/userparameters` and they would be used without any additional changes to agent daemon configuration file.



Configuration file inclusion works for other files as well, like for server or proxy ones.

Other methods to gather data

We have used so many ways to gather data, but there's always that one corner case where a different method might be preferred or required.

Sending in the data

In some cases, it might not be feasible or possible to query data or gather it using Zabbix agents. You might have some system where a Zabbix agent can't be installed, but data can be pushed out instead. Also, the previously mentioned case when retrieving data is resource intensive might benefit from direct data transmitting instead of the use of intermediate files that are queried by agent later. Zabbix provides such a functionality with the `zabbix_sender` utility and trapper type items.

As we have learned, all data that arrives must have a corresponding item set up. In the frontend, go to **Configuration | Hosts**, click on **Items** next to **A Test Host** and click **Create Item**, then fill in the following values:

- **Description:** Enter **Amount of persons in the room**
- **Type:** Select **Zabbix trapper**
- **Key:** Enter `room.persons`

When you are done, click **Save**.

We now have to determine how data can be passed into this item, and this is where `zabbix_sender` comes in. On the Zabbix server, execute:

```
$ zabbix_sender --help
Zabbix Sender v1.8.1 (revision 9702) (27 January 2010)

usage: zabbix_sender [-Vhv] {[-zpsI] -ko | [-zpI] -T -i <file>} [-c <file>]

Options:
  -c --config <File>           Specify configuration file
  -z --zabbix-server <Server>   Hostname or IP address of Zabbix Server
  -p --port <Server port>       Specify port number of server trapper running on the server. Default is 10051
  -s --host <Hostname>          Specify host name. Host IP address and DNS name will not work.
  -I --source-address <ip address>  Specify source IP address

  -k --key <Key>               Specify metric name (key) we want to send
  -o --value <Key value>        Specify value of the key

  -i --input-file <input_file>  Load values from input file
                                Each line of file contains space delimited: <hostname> <key> <value>
  -T --with-timestamps          Each line of file contains space delimited: <hostname> <key> <timestamp> <value>
                                This can be used with --input-file option

  -v --verbose                 Verbose mode, -vv for more details

Other options:
  -h --help                     Give this help
  -V --version                  Display version number
```

Looking at the output, we can figure out which parameters are mandatory.

- `-z` to specify Zabbix server
- `-s` to specify hostname, as configured in Zabbix
- `-k` for key name
- `-o` for value to send

As you might remember, this is the very same command we used to pass SNMP traps from `snmptrapd` to Zabbix.

Let's try to send some value to Zabbix server. Execute:

```
$ zabbix_sender -z localhost -s "A Test Host" -k room.persons -o 1
```

This command should succeed and show the following output:

```
Info from server: "Processed 1 Failed 0 Total 1 Seconds spent
0.044244"
sent: 1; skipped: 0; total: 1
```

Let's send another one—again, execute from the Zabbix server:

```
$ zabbix_sender -z localhost -s "A Test Host" -k room.persons -o 2
```

This one should also succeed, and now we should take a look at **Monitoring | Latest data** over at the frontend. We can see that the data has successfully arrived and the change is properly recorded.

Description	Last check	Last value	Change
- other - (1 items)			
Amount of persons in the room	05 Mar 2010 13:01:09	2	+1

Now we could try being smart. Let's pass a different data type to Zabbix. On Zabbix server, execute:

```
$ zabbix_sender -z localhost -s "A Test Host" -k room.persons -o nobody
```

We are now trying to pass a string to the Zabbix item even though in the frontend its data type is set to integer.

```
Info from server: "Processed 0 Failed 1 Total 1 Seconds spent
0.006084"
sent: 1; skipped: 0; total: 1
```

Zabbix didn't like that, though. The data we provided was rejected because of the data type mismatch, thus it is clear that any process, that is passing the data, is responsible for the data contents and formatting.

Now, security concerned people would probably ask—who can send data to items of the trapper type? A `zabbix_sender` can be run on any host by anybody. But it's possible to restrict this somewhat—see **Configuration | Hosts**, select **Items** and click on **Amount of persons in the room** in the **Description** column. Look at the one of the last few properties, **Allowed hosts**. We can specify an IP address or DNS name here, and any data for this item will be allowed from the specified host.

Allowed hosts	127.0.0.1
---------------	-----------

As that is the only protection provided by Zabbix, trapper items should be used with care in hostile environments.



Current Zabbix versions do not support encryption or secure authentication, thus sensitive information should not be exposed without protecting it by tunneling or other means.



Using custom agents

As the Zabbix protocol is fairly trivial, in some cases it might be feasible to implement custom agents. To a Zabbix server those would look just like Zabbix agents, just providing different data.

One example of such an agent is Zapcat. Zapcat provides Java application monitoring, using internal Java interfaces. As a result it improves performance when compared to command line utilities and exposes a wide range of monitored metrics. If you have Java applications and you want to monitor their internal well being, you'll definitely want to try out Zapcat. It supports polled items and sending data as active items (or traps), and there are tutorials to get it working with Tomcat, Oracle IAS, and other Java application servers. For its homepage, which is available on SourceForge.net, visit <http://sourceforge.net/projects/zapcat/>.

With the simplicity of the protocol, you might be tempted to write your own, custom agents, especially for appliances where Zabbix agent cannot be installed. In its simplest form, such an agent could even be written as a shell script that is launched from a standard `inetd` daemon—although that would most likely result in terrible performance.

Summary

In this chapter, we looked at more advanced ways to gather data.

Aggregate items allowed us to calculate particular values like minimal, maximal, and average for items over a host group. This method is mostly useful for cluster or cluster-like systems where hosts in the group are working to provide a common service.

External checks and **user parameters** provided a way to retrieve nearly any value – at least any that can be obtained on the command line. While very similar conceptually, they also have some differences that we'll try to summarize now.

External checks	User parameters
Are executed by the Zabbix server process	Are executed by the Zabbix agent daemon
Are executed on the Zabbix server	Are executed on the monitored hosts
Can be attached to any host	Can be only attached to the host where the Zabbix agent daemon runs
Can reduce server performance	Have no notable impact on server performance

As can be seen from this comparison, external checks should be mostly used with remote systems where the Zabbix agent cannot be installed, because they can be attached to any host in the Zabbix configuration. Given the possible negative performance impact, it is suggested to use user parameters in most situations.

Note that it is suggested for user parameters to have active Zabbix agent type. That way server connection is not tied up in case executed command fails to return in a timely manner. We also learned that we should take note of the environment the agent daemon runs in, as it is not initialized and that these commands should always return some value, they may not return empty string.

Armed with this knowledge we should be able to gather any value that traditional methods like Zabbix agents, SNMP, IPMI, and other built-in checks can't retrieve.

11

Monitoring Windows and Web Pages

In this chapter we will look at two somewhat different things:

- Monitoring of web pages, which have built-in support in Zabbix. We will check different sections of a webpage and monitor it for failures, and monitor download speeds, and response times.
- Monitoring of Windows systems; additional support for this is available in Zabbix, including support for a native Windows monitoring subsystem, performance counters, and installing Zabbix agent on Windows.

Monitoring web pages

The internet is important in every aspect of modern life. Socializing, business, entertainment, and everything else happens over the wire. With all the resources devoted to this network, many are tasked with maintaining websites – no matter whether it's an internally-hosted site, or one trusted to an external hosting provider, we will want to know at least its basic health status. Zabbix provides native web monitoring support, which we will try out by monitoring a real life website.

Creating web monitoring scenario

Before we rush into the creation of a web monitoring scenario, we first have to satisfy a couple of prerequisites – web scenarios in Zabbix require an existing host and application to be linked against. While we could link a created scenario to any of the existing hosts, that wouldn't correctly depict what the scenario is monitoring, so we will create a dedicated one. To create these, navigate to **Configuration | Hosts**, click **Create Host** and fill in these values:

- **Name:** Enter **OpenStreetMap**

- **Groups:** Mark **Linux servers** in the **In Groups** box, click on the **>>** button
- **New group:** Enter **Web pages**

We don't have to change any other value, so click **Save**. Now for the application part—select **Web pages** in the **Group** dropdown and click on **Applications** next to **OpenStreetMap** then click the **Create application** button. For application we have to enter the name only, so let's name this application "WEB" and save it. We're now ready to create the scenario itself—open **Configuration | Web**, make sure **OpenStreetMap** is selected in the **Host** dropdown, and click **Create scenario**. In the scenario properties, enter these values:

- **Application:** Click on **Select** next to this field, then choose the only available application, **WEB**
- **Name:** Enter **Map**
- **Update interval:** Change to **360**
- **Agent:** We wouldn't want to skew the OpenStreetMap statistics, now would we? Let's change this to the **Lynx** entry

 When a web browser connects to the web server, it usually sends along a string identifying itself. This string includes browser name, version, operating system, and often other information. While this information can be used for good purposes like gathering statistics or making a specific portion of site work better in some browser, sometimes it is used to deny access or limit experience on the site. Zabbix web monitoring checks also send user agent string to web servers, thus this option can be used to mask as another browser if some filtering is performed.

Now we have to add steps. Steps for web monitoring are the actual queries performed on the web server. Click **Add** in the **Steps** section and fill in these values in the new pop up:

- **Name:** Enter **First page**, because we'll start by testing whether the main page opens at all
- **URL:** Enter **http://www.openstreetmap.org/**
- **Required:** Enter **OpenStreetMap is a free editable map of the whole world**. This field will search for a particular string in the returned page and this step will fail if such a string is not found. We can use POSIX regular expressions here
- **Status codes:** Enter **200**. Here acceptable HTTP return codes can be specified, separated with commas. Again, if return code doesn't match, this step will be considered a failure. A status code of 200 means "OK"

This form should look like this:

Step of scenario	
Name	First page
URL	http://www.openstreetmap.org/
Post	
Timeout	15
Required	OpenStreetMap is a free editable map of the whole world
Status codes	200
<input type="button" value="Add"/> <input type="button" value="Cancel"/>	

If it does, click **Add**. This was the first step in web site testing, and it was a fairly simple one—let's do something fancy now. We will attempt to log in to OpenStreetMap and see whether that succeeds. If you have an account, you can use that one, if not what are you waiting for? Go and register an account now. With the account information on hand, we can start configuring the next step. Before proceeding, though, let's find out what the **Variables** text area is for.

When Zabbix performs further steps, it can use variables that are specified here. When logging in, we will need to pass username and password, which both seem like good candidates to be included in such variables. Enter two lines in that field:

```
{user}=user@some.domain
{password}=secret_password
```

Of course, replace the e-mail and password entries with valid values. Now we are ready to configure logging in step—click on **Add** in the **Steps** section and fill in these values:

- **Name:** Enter **Log in**.
- **URL:** Enter `https://www.openstreetmap.org/login`.
- **Post:** This is the field that will allow us to actually perform the logging in. On this site, login happens with HTTP POST variables, and Zabbix web monitoring allows us to specify such variables here. Enter in this area `[email]={user}&user [password]={password}`. Notice how we are using the previously defined variables—it allows us to specify some data only once. Additionally, it is worth paying attention to the syntax when specifying multiple variables. They are separated with &.

- **Required:** Enter **home**. When a user is logged in, a link named "home" appears, which centers the map view on the user's home location, thus we can use that as an indicator of a successful login.
- **Status codes:** Enter **200**

When you are done, click **Add**. We are now checking two things – whether the main page can be opened, and whether we can log in. Let's also check whether the GPS traces pages can be accessed. Again, click on **Add** in the **Steps** section and enter these values:

- **Name:** Enter **Traces**
- **URL:** Enter `http://www.openstreetmap.org/traces/mine`
- **Required:** Enter **Your GPS traces**
- **Status codes:** Enter **200**

In the **Required** field we entered text that user can see if login is successful and own traces page is accessible. If the login process was unsuccessful, the login screen will be opened instead, thus this works as a double check for a successful login. When you are done, click **Add**.

The final web scenario configuration should look like this. Notice that we have a handy overview of all steps and their checks available here.

The screenshot shows the 'Scenario' configuration dialog box. At the top, it has fields for 'Application' (WEB), 'Name' (Map), 'Basic authentication' (None), 'Update interval (in sec)' (360), 'Agent' (Lynx 2.8.4.rel.1 on Linux), and 'Status' (Active). Below these, there is a 'Variables' section containing two entries: [user]=user@some.domain and [password]=secret_password. The main area is titled 'Steps' and contains a table with three rows:

Name	Timeout	URL	Required	Status	Sort
<input type="checkbox"/> First page	15 sec	http://www.openstreetmap.org/	OpenStreetMap is a free editable map of the whole world	200	Down
<input type="checkbox"/> Log in	15 sec	https://www.openstreetmap.org/login	home	200	Up Down
<input type="checkbox"/> Traces	15 sec	http://www.openstreetmap.org/traces/mine	Your GPS traces	200	Up

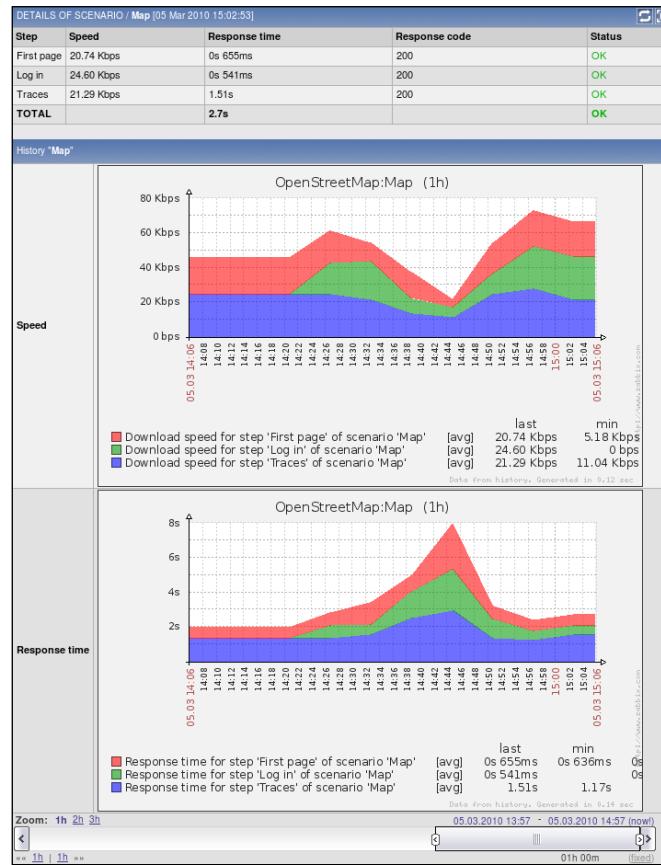
At the bottom of the dialog box are buttons for 'Add' (highlighted in blue), 'Delete selected', 'Save' (highlighted in blue), and 'Cancel'.

If everything looks fine, click **Save**. Expand the action details at the top of the page and look at the list of things that happened. In the long list of actions, you should be able to find the addition of the scenario itself but there also are lots of entries for items added. That looks interesting, but first we will look at what web monitoring visually looks like. Open **Monitoring | Web**. **OpenStreetMap** is the only host with a web scenario attached, so expand the **WEB** application to see high-level overview. It looks like all steps were completed successfully, thus we can consider the monitored

website to be operating correctly. Or at least the parts we are monitoring, because the **Status** column happily says **OK**. As with plain items we can see when the last check was performed, and additionally more detailed information on the status of this scenario is displayed, like time when the next invocation will be started:

Name	Number of steps	State	Last check	Status
WEB (1 Scenarios)				
Map	3	Idle till 05 Mar 2010 15:02:53	05 Mar 2010 14:56:53	OK

We also have an overview of how many steps each scenario contains, but that's all very vague. Click on **Map** in the **Name** column—maybe there's more information. Indeed there is. Here we can see statistics for each step like the speed, response time, and response code. And, if that's not enough, there are nice predefined and pretty graphs for the speed and response time. Note that these are stacked graphs, so we can identify moments when all steps together take more time. If we pay attention to the lower area of the graphs, we can notice those familiar timescale controls—the scrollbar, zoom, and calendar controls, so these graphs provide the same functionality as anywhere else, including clicking and dragging to zoom in.



We can see how logging in and first page access were fastest operations for this particular user account, while opening the GPS traces page was notably slower, and all three operations on average take slightly less than three seconds.

Note that for the last step to succeed, the user must be logged in and we do that in the second step. For such a thing to work, **Zabbix keeps all received cookies for latter steps during the whole scenario.**

While this view is very nice, it isn't too flexible. If only we had direct access to underlying data... wait, what were those items we saw added when the scenario was created? If there are items, there must be data. Let's visit **Monitoring | Latest data** to find out. Select **Web pages** in the **Group** dropdown and **OpenStreetMap** in the **Host** dropdown and take a close look at available data (expand **WEB** application, if it is collapsed). We can directly access information about every step like the download speed and response time, which allows us to create whatever graphs we please – maybe we want a pie chart of response times for each step, or a non-stacked graph of download speeds. Of course, as with all items, we get simple graphs without any additional configuration.

+ Description	Last check	Last value	Change	History
WEB (11 Items)				
Download speed for scenario 'Map'	05 Mar 2010 16:03:32	27.61 Kbps	+13.50 Kbps	Graph
Download speed for step 'First page' of scenario 'Map'	05 Mar 2010 16:03:31	20.59 Kbps	+17.13 Kbps	Graph
Download speed for step 'Log in' of scenario 'Map'	05 Mar 2010 16:03:31	25.46 Kbps	+6.03 Kbps	Graph
Download speed for step 'Traces' of scenario 'Map'	05 Mar 2010 16:03:32	36.79 Kbps	+17.34 Kbps	Graph
Failed step of scenario 'Map'	05 Mar 2010 16:03:32	0	-	Graph
Response code for step 'First page' of scenario 'Map'	05 Mar 2010 16:03:31	200	-	Graph
Response code for step 'Log in' of scenario 'Map'	05 Mar 2010 16:03:31	200	-	Graph
Response code for step 'Traces' of scenario 'Map'	05 Mar 2010 16:03:32	200	-	Graph
Response time for step 'First page' of scenario 'Map'	05 Mar 2010 16:03:31	0s 660ms	3.26s	Graph
Response time for step 'Log in' of scenario 'Map'	05 Mar 2010 16:03:31	0s 523ms	0s 162ms	Graph
Response time for step 'Traces' of scenario 'Map'	05 Mar 2010 16:03:32	0s 872ms	0s 777ms	Graph

Two additional items are the download speed for whole scenario and failed step, which returns "0" if none of the steps failed. While former is handy to figure out how long all the steps take together, the latter is even more important. If we know that value is "0" when everything is fine, then we can check for this value not being "0" in a trigger, and alert based on that. Let's create a trigger that warns us when any one of the steps fails; to create a trigger we will need the item key. Where can we find it?

Let's try to look at the item list. Go to **Configuration | Hosts**, click on **Items** next to **OpenStreetMap** host. No items are shown. Weird, we saw item creation in the details. Well, these items are special – they are Zabbix internal items and thus are not available for manual configuration, so we'll have to find the item key somewhere else.

Click on **Triggers** in the header, then click **Create Trigger**. In the trigger editing form, enter these values:

- **Name:** Enter **{HOSTNAME} website problem**.
- **Expression:** Click on **Select**, then click on **Select** next to the **Item** field in the upcoming pop up. Select **Web pages** in the **Group** dropdown and **OpenStreetMap** in the **Host** dropdown, then click on **Failed step of scenario Map** in the **Description** column. We have to find out when this item is not returning zero, but we wouldn't want to fire an alarm after the first failure – failures can be caused by many elements in the path to the servers and in the server infrastructure itself, thus in cases like this it is better to use trigger expression that is more resilient to flapping. Let's say, we want to know about 3 failures in a row. Choose **Maximal value for period of time T NOT N** in the **Function** dropdown. Next to the **Last of** field, choose **Count** in the dropdown, enter **3** in the textfield, and click **Insert**. Final trigger expression should be like this:

```
{OpenStreetMap:web.test.fail[Map].max(#3)}#0
```

When you are done, click **Save**. We can see how the item key `web.test.fail[Map]` was used, thus web scenario items are very much like normal items. They have names and keys, even though they can't be seen in item configuration view. This way we can create triggers for all web scenario items like response time and download speed to also spot performance issues, or for return codes to spot exact steps that fail.

If a web monitoring step fails, Zabbix stops and does not proceed to the next step. If the website you are monitoring has multiple sections that can work independently of one another, you should create separate scenario for each. For OpenStreetMap, such a situation is possible, as main site might have problems, while the wiki was still working, or the opposite. If we want to monitor these independently, we'll have to create a separate scenario – open **Configuration | Web**, make sure **OpenStreetMap** is selected in the **Host** dropdown, and click **Create scenario**. In the scenario properties, enter these values:

- **Application:** Click on **Select** next to this field, then choose **WEB**
- **Name:** Enter **wiki**
- **Update interval:** Enter **300**
- **Agent:** Change this to the Lynx entry

Now on to the scenario steps. Click **Add** in the **Steps** section and fill in these values in the new pop up:

- **Name:** Enter **wiki frontpage**, in case we decide to monitor additional wiki pages later
- **URL:** Enter <http://wiki.openstreetmap.org/>
- **Status codes:** Enter **200**

Click **Add**, then click **Save**. This scenario is a very simple one, it contains only a single step, and verifies that the wiki's front page can be opened. Feel free to expand it by checking out whether it is possible to log in; check the wiki first page for the **Up and running** string in the **Platform Status** section, and maybe additional tests—that will help you to learn how to configure web testing for production systems.

For production systems, there usually will be way more applications, scenarios, and steps. Web monitoring can be used for many different purposes, the most popular being site availability and performance, but there are many different cases one could monitor, including things like watching the Slashdot frontpage for a company name and replacing the usual first web page with a more simple one to withstand the coming load easier.

Windows-specific monitoring

While the things we learned about using Zabbix agents, creating items and yes, even user parameters are useful, there are some things that are specific to Windows system monitoring which we will look at now. For this section you will need a Windows machine that is accessible from the Zabbix server.

Installing Zabbix agent for Windows

To install the agent, it first has to be obtained. On Windows, compiling software is less common and most users get binary distributions, which is exactly what we will do now.

The Windows build of the Zabbix agent can be obtained from two official locations—either from the download page at <http://www.zabbix.com/download.php>, or from the source archive. This time it is suggested to use the one included inside the archive to make sure the versions match. The agent executable is located in the subdirectory `bin/win32` or `bin/win64`—choose the one that is appropriate for your architecture and place it on some directory in the Windows machine.

For simplicity, we'll use `C:\zabbix` this time, but you are free to use any other directory. We will also need the configuration file, so grab the example provided at `misc/conf/zabbix_agentd.win.conf` and place it in the same directory. Before

we continue with the agent itself let's figure out whether we need to alter the configuration in any way. Open `C:\zabbix_agentd.win.conf` in your favorite text editor and look for any parameters we might want to change. Firstly, the log file location isn't quite optimal – it's set to `c:\zabbix_agentd.log`, so let's change it to read:

```
LogFile=c:\zabbix\zabbix_agentd.log
```



You can use both forward and backward slashes on the Windows Zabbix agent daemon command line and in the configuration file.

We have already learned that the server line, which currently reads `Server=127.0.0.1`, will have to be changed. Replace the `127.0.0.1` part with the IP address of your Zabbix server. As this is not a Zabbix server, replace the `Hostname` directive to read:

```
Hostname=Windows box
```

Now let's try to start the agent up. Execute:

```
C:\zabbix>zabbix_agentd.exe -c c:/zabbix/zabbix_agentd.win.conf
```

While the agent daemon does start up, it doesn't like being started like that and prints a warning:

```
zabbix_agentd.exe [1464] :
```

```
!!!!ATTENTION!!!! ZABBIX Agent started as a console application.  
!!!!ATTENTION!!!!
```

So what can we do to make it happier? First stop it by pressing `Ctrl+C`. The agent daemon executable on windows has additional options that can be passed to it, so execute in the command prompt (when located in the directory where `zabbix_agentd.exe` resides):

```
C:\zabbix>zabbix_agentd.exe --help
```

While the start of the output does not differ notably, we can see several additional parameters at the end of the help.

```
ZABBIX Agent Win32 (service) v1.8.1 (revision 9692) (27 January 2010)
```

```
usage: zabbix_agentd.exe [-Vhp] [-idsx] [-m] [-c <file>] [-t <metric>]
```

```
Options:
```

```
-c --config <file>      Specify configuration file. Use absolute path
-h --help                give this help
-V --version              display version number
-p --print                print supported metrics and exit
-t --test <metric>       test specified metric and exit
```

Note that -t and -p switches do not work with user parameters. Use zabbix_get instead.

Functions:

```
-i --install             install Zabbix agent as service
-d --uninstall           uninstall Zabbix agent from service
-s --start               start Zabbix agent service
-x --stop                stop Zabbix agent service
-m --multiple-agents    service name will include hostname
```

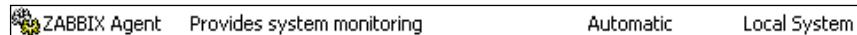
The Zabbix agent daemon for Windows includes functionality to install it as a standard Windows service, which is controlled by the last set of options. Unless you are simply doing some testing, you'll want to properly install it, so let's do that now.

```
C:\zabbix>zabbix_agentd.exe -c c:/zabbix/zabbix_agentd.win.conf -i
```

The Zabbix agent daemon is installed as a Windows service using the configuration file, specified by the -c flag.

```
zabbix_agentd.exe [4376]: Service "ZABBIX Agent" installed
successfully.
zabbix_agentd.exe [4376]: Event source "ZABBIX Agent" installed
successfully.
```

You can verify in the Windows Control Panel, Services section, that the Zabbix service has indeed been installed:



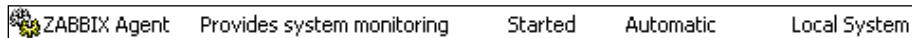
While it has been set to start up automatically, it is stopped now. We can start it by either right clicking the **Zabbix Agent** service entry and choosing **Start**, or by using command line switch to zabbix_agentd.exe. Let's try the latter method now:

```
C:\zabbix>zabbix_agentd.exe -c c:/zabbix/zabbix_agentd.win.conf -s
```

Supposedly service was started successfully.

```
zabbix_agentd.exe [5608]: Service "ZABBIX Agent" started successfully.
```

Again, let's verify in the services list that the Zabbix service has started up:



It looks like everything is fine on the monitored host, which we will now have to configure in the frontend. Open **Configuration | Hosts**, and click **Create Host**, then fill in the following values:

- **Name:** Enter **Windows box**.
- **Groups:** Select **Windows servers** in the **Other Groups** box, then click on the << button. If there's any other group in the In Groups box, remove it.
- **IP address:** Enter the IP address of that host.

When you are done, click **Save**. Now select **Windows servers** in the **Group** dropdown and click on **Items** next to the **Windows box**, then click **Create Item**. Enter these values:

- **Description:** Enter **CPU Load**
- **Key:** Enter `system.cpu.load`
- **Type of information:** Select **Numeric (float)**
- **Update interval:** Enter **60**
- **Keep history:** Enter **7**

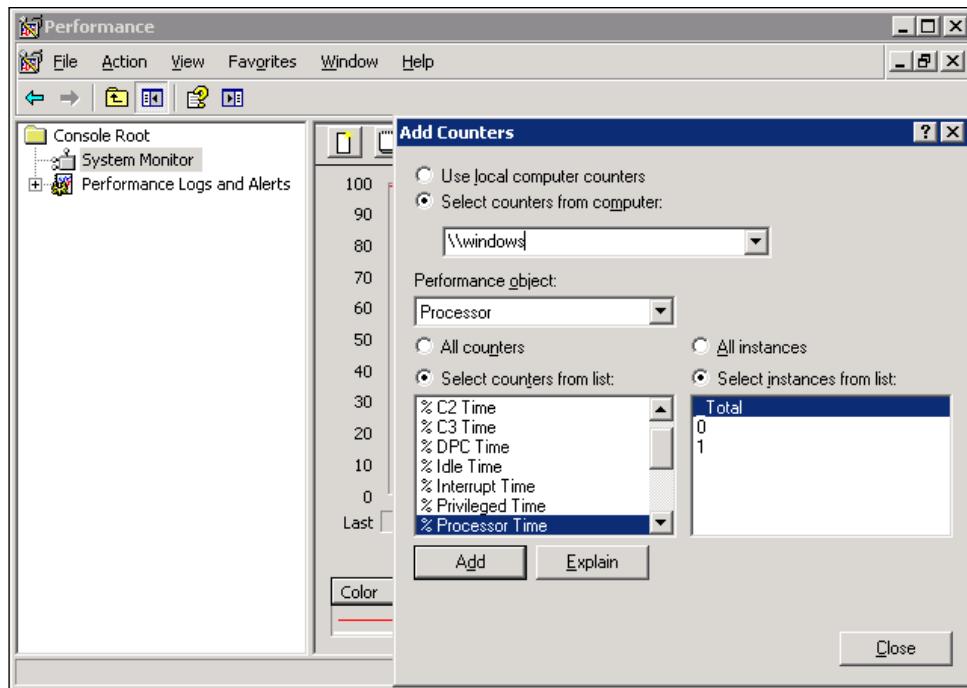
When you are done, click **Save**. We can now check out incoming data at **Monitoring | Latest data**—select **Windows servers** in the **Group** dropdown.

ITEMS		Group	Windows servers	Host	Windows box
Filter					
+ Description	Last check	Last value	Change	History	
- other - (1 Items)					
CPU Load	05 Mar 2010 18:18:59	0.220000	+0.22	Graph	

We have now successfully retrieved data on the CPU load for this Windows machine. Notice how the key syntax is the same as for Linux. This is true for several other keys, and you can check out the Zabbix documentation to determine which keys are supported on which platform.

Querying performance counters

While many keys match between platforms, there's a whole category that is specific to Windows. Zabbix supports Windows' built-in metrics gathering system, **performance counters**. People who are familiar with Windows probably know that these can be found at **Control Panel | Administrative Tools | Performance**, with a lot of counters to add. In this window, we can click on the + icon in the child toolbar, or press **Ctrl+I** to see available counters.



In this dialog, we can gather the information required to construct performance counter string. First, the string has to start with a backslash, \. The **Performance object** follows, in this case, **Processor**. Then we have to include the desired instance in parenthesis, which would make our string so far \Processor(_Total) (notice the leading underscore before Total). The counter string is finished by adding individual counter string from the **Select counters from list** list box, again separated by a backslash. So the final performance counter string looks like this:

```
\Processor(_Total)\% Idle Time
```

Now that we have constructed it, what do we do with it? Create an item, of course. Back in the frontend, navigate to **Configuration | Hosts**, click on **Items** next to the **Windows box** and click **Create Item**. Fill in these values:

- **Description:** Enter **CPU idle time, %**
- **Key:** This is where things get more interesting, although the principle is quite simple – the `perf_counter` key has to be used with the performance counter string like the one we constructed before as a parameter, thus enter `perf_counter[\Processor(_Total)\% Idle Time]` here
- **Type of information:** Select **Numeric (float)**
- **Update interval:** Enter **60**
- **Keep history:** Enter **7**

When you are done, click **Save**. This item should show us the total time all CPUs spend idling on the machine, so let's look at **Monitoring | Latest data** with **Windows box** selected in the **Host** dropdown. We can see that the data is directly fetched from the built-in performance counter.

CPU idle time, %	05 Mar 2010 18:29:00	50.000000
------------------	----------------------	-----------

Looking at the list of available performance objects and corresponding counters, we can see many different metrics. Navigating this window is cumbersome at best, thanks to small widgets, no filtering or searching capabilities, and the fact that constructing the required string to be used as key is a manual typing job as entries can't be copied. Luckily there's a solution available on recent versions of Windows – the command line utility `typeperf.exe`. To see how it can help us, execute:

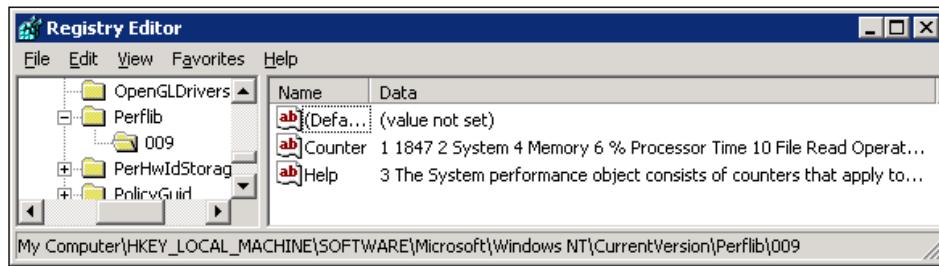
```
C:\zabbix>typeperf -qx > performance_counters.txt
```

This will direct all output of this command to be saved in the file `performance_counters.txt`. Open that file with a text editor and observe the contents. You'll see lots and lots of performance counter strings, covering various software and hardware information. There is no need to struggle with that clumsy dialog any more, we can easily search for and copy these strings.

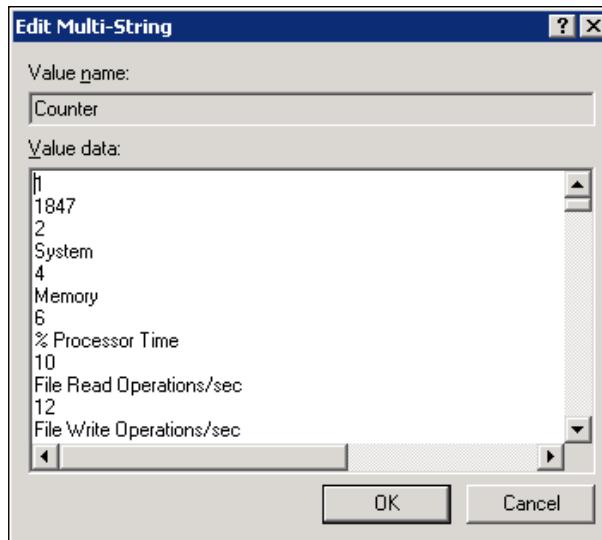
Using numeric references to performance counters

If you have a localized Windows installation, you have probably noticed by now that all performance counters are in the localized language, not in English. This becomes especially cumbersome to handle if you have to monitor several Windows machines with different locales configured for them. For example, a counter that on an English Windows installation is `\System\Processes`, would be `\Système\Processes` in a French one. Would it be possible to use some other, more universal method to refer to the performance counters? Indeed it is, we can use numeric references, but first we have to find out what they are.

Launch `regedit` and look for the key `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib`. Under this key you'll see one or more entries, with one being `009`, which is entry for the English language. Select this entry and pay attention to the **Counter** key, which has suspiciously similar contents to performance counter names.



Double click this value to see its contents in a somewhat more manageable form:



So each performance counter string can be translated to a number. But figuring out exact conversion in this tiny window is awfully hard, so let's copy all contents and save them to some file, which we'll be able to search then—name it `numeric.txt`. To see how this works, let's translate performance counter string we used before—`\Processor(_Total)\% Idle Time`.

First we have to translate the performance object, Processor. While it is possible to search these contents in any text editor, it soon becomes cumbersome, especially if we have to translate lots of values. In that case we can turn to the basic GNU tools like `grep`, which you might have installed on the Windows machine—if not, copy this file over to the Zabbix server:

```
$ grep -B 1 "^Processor$" numeric.txt
```

This command will search for a line containing the string `Processor` exactly and will also output the line immediately before it, which contains the numeric ID of this performance object.

```
238
Processor
```



Numeric values might differ between Windows versions, so make sure to use values found in your file.

If you are using `grep` on the Zabbix server, saved file might contain Windows style newlines and you might get no output. In that case, convert newlines by executing:

```
$ sed -i 's/\r//' numeric.txt
```

Now that we have the numeric value for the first part, do the same for the second part of the performance counter.

```
$ grep -B 1 "^\% Idle Time$" numeric.txt
```

```
1482
% Idle Time
```

We now have numeric values for all parts of the performance counter except the `_Total`—how can we translate that? We don't have to, this string is used as is on all locales. Our resulting performance counter would then look like this:

```
\238(_Total)\1482
```

As we already have an item gathering this information, we won't add another one. Instead let's test with the `zabbix_get` utility. On the Zabbix server, execute:

```
$ zabbix_get -s <Windows box IP> -k "perf_counter[\238(_Total)\1482]"
```

This should return the same data as `\Processor(_Total)\% Idle Time` key does.

49.998400



Additional software can add additional performance counters, and numeric values for such counters can differ between systems. In some cases software modifies existing performance counters, for example, adding the firewall software vendor's name to a network interface.

Using aliases for performance counters

Another method to unify item keys that are using Zabbix configuration (so that a single template could be used for all hosts) is to specify performance counter aliases. To do that, add `Alias` directive to Zabbix agent configuration file. For example, if we wanted to refer to the performance counter we used, `\Processor(_Total)\% Idle Time`, as `cpu.idle_time`, we would add the following:

```
Alias = cpu.idle_time:perf_counter[\Processor(_Total)\% Idle Time]
```

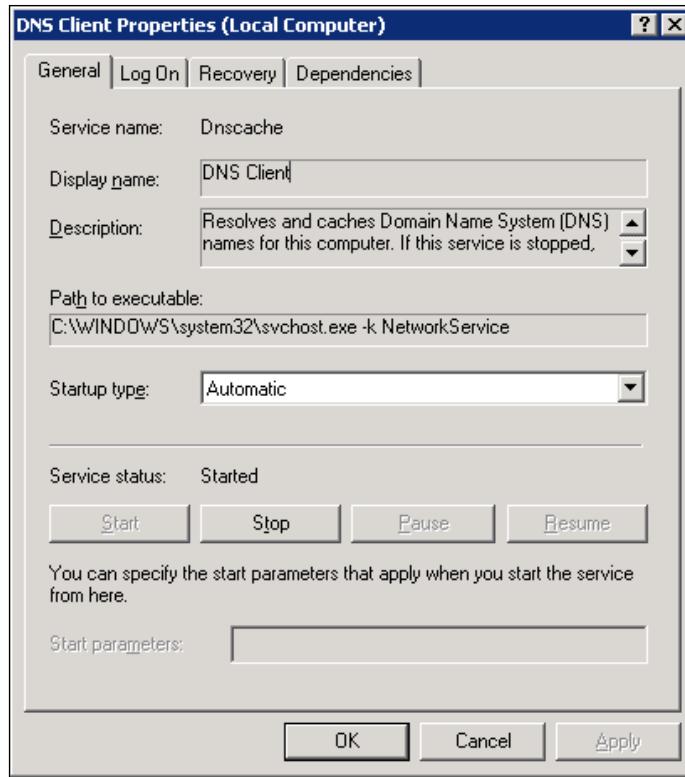


Do not forget to restart the agent after making the changes.

Systems with another locale alias would use a different performance counter, but from now on we can use the same item key for all systems — `cpu.idle_time`.

Monitoring Windows services

Performance counters are a Windows-specific metric category, and there's another one as well — a dedicated key for Windows service state monitoring. Let's try to monitor some service now. First we have to figure out how to refer to this service. For that, open the services list and then open up the details of some service — let's choose **DNS Client**.



Look at the top of this tab. **Service name** is the name we will have to use, and we can see that it differs notably from the display name – instead of using **DNS Client**, the name is **Dnscache**. Let's create item now – navigate to **Configuration | Hosts**, click on **Items** next to the **Windows** box, then click **Create Item**. Enter these values:

- **Description:** Enter **DNS client service state**
- **Key:** Key `service_state` allows access to information about system services, thus enter `service_state [Dnscache]` here
- **Update interval:** Enter **60**
- **Keep history:** Enter **7**

[ Service names are case insensitive.]

When you are done, click **Save**, open **Monitoring | Latest data**, and look for our newly added item.

DNS client service state	05 Mar 2010 18:57:01	0
--------------------------	----------------------	---

So data is gathered, and the state is "0". That's probably normal, but how could we know what the state number means? Back in **Configuration | Hosts**, select **Items** in the first dropdown and click on **DNS client service state** in the **Description** column. Look at our old friend, the **Show value throw map** property. Click on the **throw map** link and examine the mapping with most entries.

Windows service state	0 => Running 1 => Paused 3 => Pause pending 4 => Continue pending 5 => Stop pending 6 => Stopped 7 => Unknown 255 => No such service
---------------------------------------	---

It turns out, there's already a predefined mapping for Windows service states available. Close this window and choose **Windows service state** in the **Show value throw map** dropdown, then click **Save**. Back at **Monitoring | Latest data**, verify that service state is now displayed in a much more user friendly way:

DNS client service state	05 Mar 2010 19:00:01	Running (0)
--------------------------	----------------------	-------------

Now we will be able to easily identify different service states in the frontend.

Checking whether an automatic service has stopped

Sometimes we are not interested in exact details about every service, and we would have to configure each of them manually. Instead we might want to see a high-level overview, for example, whether any of the services that are started automatically have stopped. Since version 1.8.1, Zabbix provides an item that allows to make such a comparison very easily—`services[automatic,stopped]`. It allows us to retrieve lists of services based on different parameters, including ones that should be started automatically and are stopped. How can we use this?

An item should be added with the following key:

```
services[automatic,stopped]
```



For a list of all supported services key parameters consult the Zabbix manual.



This will take care of getting the needed data. Whenever a service that is set to start automatically is stopped, it will be listed in the data from this item. But how do we check for that in a trigger? If the list is empty, the Zabbix agent returns 0. As a result, by simply checking whether last value was zero we can trigger whenever an automatically started service is stopped. A trigger expression for such a check would be:

```
{Windows box:services [automatic,stopped] .last(0)}#0
```

Of course, you can apply some method to only fire the trigger after it has been non-zero for more than a single check.

```
{Windows box:services [automatic,stopped] .count (#3,0)}=0
```

Such a trigger expression will only fire if there has been at least one such stopped service in all of the last three checks.

Summary

In this chapter, we looked at two different things, both starting with "W".

First, we learned to monitor web pages based on various parameters, including response time, transfer speed, HTTP return code, and text, contained in the page itself. We also found out about how to set up multiple scenarios and steps in them, as well as set up variables to be used in all steps. With this knowledge it should be possible to monitor most of the functionality web pages have.

On the other hand, we learned the specifics of monitoring Windows machines, like setting up a Zabbix agent on Windows as a system service and querying Windows-specific values, like built-in performance counters and system service states. Coupled with the generic monitoring and reporting knowledge we have by now, this should allow to efficiently monitor Windows installations as well.

12

Using Proxies to Monitor Remote Locations

While a direct connection between server and agent is the easiest to set up and use, it is not always possible—firewalls and other devices can get in the way. This chapter will detail one solution, Zabbix proxies.

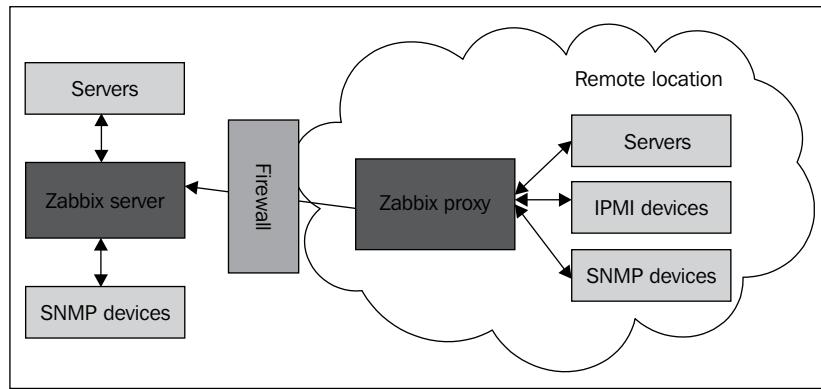
Zabbix proxies provide a remote data collection solution (although not restricted from being used locally), and in this chapter we will figure out:

- How to set up and configure a Zabbix proxy instance
- How to gather data using a proxy
- What benefits a Zabbix proxy provides
- How to determine whether the Zabbix proxy itself is available and working

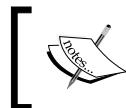
When proxies are useful

A server is not always able to connect directly to the monitored machines. We could use active items everywhere (where Zabbix agents connect to a Zabbix server, discussed in *Chapter 3*), but that would require access from all of the monitored hosts to the Zabbix server. Additionally, that would not work for SNMP, IPMI, and other monitoring methods. This is common when devices from other organizations have to be monitored, or when such restrictions are in place in a large corporate network.

This is where Zabbix proxy comes in. When set up, only connections to the Zabbix server come from the proxy, which, in turn, does all the monitoring on behalf of the Zabbix server.



In this example, there's no need for individual connections from Zabbix to individual devices in the remote location or the other way around – single firewall rule to allow Zabbix proxy connections to the Zabbix server is sufficient.



An alternative to Zabbix proxies could be port forwarding or tunneling, but that is usually much harder to set up and maintain, especially if you are restricted by security policies.



So how does the proxy work? Let's repeat: **Only the Zabbix proxy connects to the Zabbix server**. This bit of information is crucial to understanding how it actually works. When the proxy connects to the server, it requests configuration information about what it has to monitor, then goes and does just that. As data is gathered, proxy sends it back to the server. Does that sound familiar? It should, because that part sounds just like an active agent, except that the proxy can gather data from different systems pretty much like the Zabbix server can. It sounds almost like a super agent. Actually, it is said that the proxy was at first internally named "super agent", which confirms the similarity.

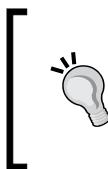
The Zabbix proxy is a very useful feature, the first stable Zabbix version to introduce it was 1.6, back in 2008. Given the usefulness of the Zabbix proxy, it is surprising that for a long time no other solutions provided something comparable. Today, Zabbix proxies continue to lead the way by having and improving features that are considered most useful for remote data collection.

As we will see later, there are other benefits that proxies can provide, so let's get to actually setting one up.

Setting up the proxy

When setting up the proxy for this exercise, it is suggested you use a separate (physical or virtual) machine. If not possible, you can choose to run proxy on the same machine, as hints will be provided for such a setup as well.

Before proceeding, we will have to compile the proxy itself. When compiling the proxy there are a few mandatory parameters, `--enable-proxy` being one. Zabbix proxy also needs a database, so you have to include support for one. While it is not suggested to use SQLite on Zabbix itself because of the performance issue, the proxy might be a good candidate. Unless it is going to monitor lots of systems, SQLite's performance can be satisfactory, and it is trivial to set up because Zabbix server creates and populates the database automatically.



If a Zabbix proxy is going to monitor many devices, it is suggested to consider other database than SQLite. Larger workloads are known to have locking issues, and there are reports of database corruption cases as well. The threshold depends on many parameters, but anything past 50 monitored devices per proxy should use a different database.

Additionally, Zabbix proxy must have support compiled in for things it will be monitoring, like SNMP, IPMI, and web monitoring. Let's say we want to compile Zabbix proxy with SQLite support and the ability to monitor web, SNMP, and IPMI. Obviously, we will need all the relevant support libraries mentioned when we first compiled Zabbix, but this time we'll also need another one—SQLite development headers. On most distributions they will be included in a package named `sqlite-devel` or similar, so make sure to install this package before proceeding. Then, on the machine where you plan to install Zabbix proxy, enter the directory where the Zabbix installation source has been extracted to and execute:

```
$ ./configure --enable-proxy --with-sqlite3 --with-libcurl --with-net-snmp --with-openipmi && make
```

This will both configure and compile Zabbix proxy. We can again use the simple and universal solution to create a more or less proper package by executing as root:

```
# checkinstall --nодoc --install=yes -y --pkgname=zabbix-proxy
```

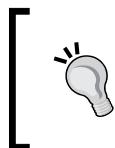
The proxy binary is now compiled and in place, we can put our hands on the configuration now. First things first—it must be placed in the correct location:

```
# cp misc/conf/zabbix_proxy.conf /etc/zabbix
```

Now open `/etc/zabbix/zabbix_proxy.conf` as root and make some changes to get the proxy running:

```
Hostname=proxy
```

Hostname is an important parameter. As with active agents, if it's wrong, nothing will work.



If you are running proxy on the same machine as a Zabbix server, you also have to change the proxy's port, set `ListenPort=10052`. We have to change the port the proxy will be listening on, otherwise the port setting would conflict with the server port.



```
Server=<Zabbix server IP address>
```

Zabbix proxy connects to the Zabbix server, so it must know where to connect to.

```
DBName=/tmp/zabbix_proxy.db
```

We will try to use an SQLite database, thus the full path to the database file must be specified.



On a production system it is suggested to place database file as well as the proxy log file and PID file in location other than /tmp. On some distributions /tmp might be cleared upon reboot, thus removing the log file.



Let's try to start the Zabbix proxy now, as root execute:

```
# zabbix_proxy
```

Wait, we did not create the database like we did when installing the server—the proxy couldn't start up, right? Let's look at the log file—open `/tmp/zabbix_proxy.log`. Paying close attention we can find some interesting log records.

```
27982:20090729:131636.530 Cannot open database file "/tmp/zabbix_
proxy.db": No such file or directory
27982:20090729:131636.531 Creating database ...
```

Wonderful. **Zabbix proxy can automatically create the required SQLite database and populate it.**



As with the Zabbix server, you must ensure that the appropriate pollers are configured to start. For example, if you want to monitor IPMI devices through a proxy, make sure to set the `StartIPMIPollers` parameter in the configuration file to a value other than the default "0".



We could also verify that Zabbix proxy is listening on the port it should be by running:

```
$ netstat -ntl | grep 10051
```

The output should confirm that everything is correct.

```
$ netstat -ntl | grep 10051
tcp      0      0 0.0.0.0:10051        0.0.0.0:*      LISTEN
```

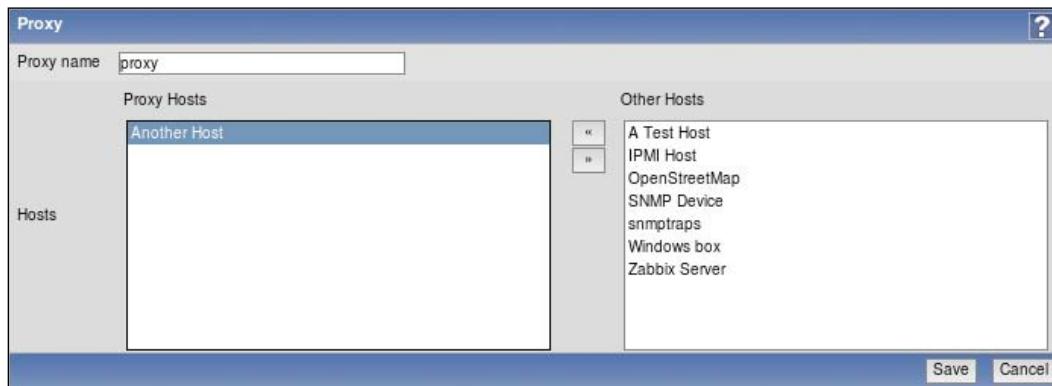
[ If installing on the same machine, check for port 10052.]

Monitoring a host through a proxy

Now that we have the proxy compiled, configured, and running, we have to inform Zabbix about it somehow. To do this, open **Administration** | **DM** in the frontend, then select **Proxies** in the first dropdown. No proxies are listed, so we have to create one—click the **Create Proxy** button and complete the form. Enter **proxy** in the **Proxy name** field.

[ Proxy name we enter here must match the one configured in zabbix_proxy.conf file.]

The section below; **Hosts**, allows us to specify which hosts will be monitored by this proxy. To make one host monitored by proxy, mark **Another Host** in the **Other Hosts** list box and click on the << button.



When you are done, click **Save**.



You can verify whether the proxy can successfully connect to the server by opening **Administration** | **DM** and selecting **Proxies** again. Allow for a couple of minutes to pass after adding the proxy and check that the **Last access** column for the new proxy has updated. If it has not, check that both the Zabbix server and proxy are running, and that you can open a connection from the proxy host to Zabbix server, port 10051.

This server should now be monitored by the proxy, right? Again, not quite. As you might remember, we were setting the Zabbix server IP address in the agent configuration files so that agents would accept connections from these servers and send active check data to them. When we change a host to be monitored by a proxy, all connections will be made from the proxy, thus the proxy address must be added to the agent daemon's configuration file.

As root, edit `/etc/zabbix/zabbix_agentd.conf` on Another Host and change the `Hostname` line to have the Zabbix proxy IP address instead of the Zabbix server IP address. Save the file, then restart the agent daemon. Now this agent will send all data to the proxy only, no connection will be made to or from Zabbix server.



We could also add the IP address of the proxy server instead of replacing it (separated by a comma) but note that in such a case active items will keep on reporting to the Zabbix server directly. Remember, agent processes active checks only from the first IP in this configuration parameter. If we did that, incoming requests would be allowed both from Zabbix server and Zabbix proxy hosts.

If, on the other hand, you have the proxy running on the same host as Zabbix server, agent will not notice that it is now queried by the proxy, not by the server, because the source IP would not have changed. The agent is still requesting and sending active checks to port 10051, Zabbix server port. To change this for the proxy port, edit `/etc/zabbix/zabbix_agentd.conf` on Another Host as root and change the `ServerPort` line to read:

```
ServerPort=10052
```

Don't forget to remove the leading hash mark, then restart the agent daemon.



Change `ServerPort` only if you have the Zabbix proxy running on the same machine as Zabbix server.

With the host monitored solely by the proxy, let's check whether there are any indications of that in the frontend. Open **Configuration | Hosts**, make sure **Linux servers** is selected in the **Group** dropdown and take a look at the **Name** column. As can be seen, **Another Host** is now prefixed by the proxy name and reads **proxy:Another Host**.

Name
proxy:Another Host
A Test Host
IPMI Host

Of course, we are not limited in our proxy name choice, we could have named it "Nothing to see here" for all that the Zabbix server cares.



When having multiple proxies, it is a common practice to name them by location name—for example, "proxy-London" or "Paris-proxy".



But do we have to always go to **Administration | DM** whenever we want to make a host monitored by proxy? Click on **Another Host** in the **Name** column, and observe the available properties. There's one dropdown available, **Monitored by proxy**. Using this dropdown we can easily assign a host to be monitored by the chosen proxy (remembering to change server IP address in the agent daemon configuration file, if using active checks).



Proxy benefits

But how independent is a proxy really? What happens if the server goes down? If we look at `/etc/zabbix/zabbix_proxy.conf`, we can find a configuration parameter named `ProxyOfflineBuffer`. This parameter controls how long the proxy will keep data in its local database (in hours) in case Zabbix server cannot be contacted. Being one hour by default, this means the Zabbix server can be down or unreachable for up to one hour, and we would not lose any data as long as it came back online before the hour was up. Of course, you can increase this period but make sure you have enough free disk space, especially if running on embedded hardware.



There are more proxy-specific configuration items available, they are listed later in this chapter.



There's one caveat—a proxy is a data collector only. This means that **proxy does not process events, send out alerts, or execute remote commands**. You need to pass the data to the Zabbix server so that it can do all those things.

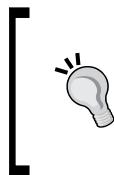
Thus proxies can gather data while the Zabbix server is down or unreachable for some reason and buffer this data for the configured period. If the Zabbix server was down for two hours, we would still lose one hour worth of data.

Another benefit is performance gain. This is not something to ignore; as your Zabbix installation grows, the load on the database can bring it down quite easily, especially if you are not careful with monitored items and their interval choices. As the proxy deals with all the hard work of polling, keeping an eye on when it has to do checks and providing Zabbix server with a nice, prepared chunk of data to insert into the database, in some configurations adding a proxy can notably reduce the load on the Zabbix server. Still, the proxy has to grab the configuration from Zabbix server now and then, and this process isn't as lightweight. The server has to figure out whether the configuration for the systems the proxy is responsible for contain data that has changed since the last synchronization, and this can be fairly resource intensive. If the configuration is very unbalanced, it can even decrease server performance. For example, having a proxy that synchronizes the configuration very often but only monitors a few items would usually make little sense.

By default, proxies synchronize the configuration once per hour, and this period can be set in `/etc/zabbix/zabbix_proxy.conf` configuration file. Look for parameter named `ConfigFrequency`, which by default will look like this:

```
# ConfigFrequency=3600
```

This means that a Zabbix proxy can lag in configuration up to an hour, which might sound scary but once a production installation settles, the configuration usually doesn't change that often. While testing, you might wish to decrease this period, but in a stable production setup it is actually suggested to increase this value.



If you must have configuration changes pushed to a proxy immediately, simply restart the proxy process, but remember that this can also result in lots of metrics being queried. If you still don't have any data gathered for **Another Host** since you assigned it to be monitored by the proxy, restart proxy now.



Talking about the benefits a proxy provides, we already discussed the third big one, the ability to have only one way connections, even when monitoring things that can't have Zabbix agent installed such as various network devices, printers, and UPSes. That can help with firewalls allowing connections only in one direction, as well as requiring only a single port to be open instead of dozens for various services and protocols.

Proxy reliability

With all the benefits that a proxy brings, how can one resist embracing it wherever possible – especially on remote locations? Well, first, a Zabbix proxy doesn't work on thin air, it needs a server. If all you have on that remote location is SNMP, IPMI, or other protocols capable devices that have no way to install a Zabbix proxy, you'll have to live without this useful daemon.

Even if there is a machine that you can use for a Zabbix proxy, problems with that host will impact the monitoring of all other devices. As a proxy goes down, neither normal, active, SNMP, nor any other check would be processed. So one really must have a reliable host on the other end. But even reliable hosts can have problems.

As we now know, all connections are initiated from the proxy (configuration synchronization and data transfer including), which essentially means Zabbix server has no idea whether proxy is still there. Well, this is not quite true, as a Zabbix server actually can find out whether something really bad has happened with a proxy. Let's try to figure out how we can determine when a proxy was last seen reporting. Open **Configuration | Hosts**, make sure **Linux servers** are selected in the **Group** dropdown and click on **Items** next to **A Test Host** then click on **Create Item**. Fill in the following values:

- **Description:** Enter **Last proxy access**
- **Type:** Select **Zabbix internal**
- **Key:** Enter `zabbix[proxy,proxy,lastaccess]`
- **Units:** Enter `unixtime`
- **Keep history:** Enter `7`



In the key here the second parameter is the proxy name. Thus, if your proxy was named "kermit", key would become `zabbix[proxy,kermit,lastaccess]`.

When you are done, click **Save**. Notice how we used special unit type—`unixtime`. Now what would it do? To find out, navigate to **Monitoring | Latest data**, make sure **A Test Host** is selected in the **Host** dropdown. Expand the filter and enter **proxy** there, then click the **Filter** button. Look at the way data is presented here—we can see very nicely in a human-readable form when the proxy last contacted server.

+ Description	Last check	Last value
[- other - (1 items)]		
Last proxy access	05 Mar 2010 21:39:02	2010.03.05 21:38:40

So this item will be recording the time when proxy last contacted Zabbix server. That's great, but hardly enough to notice problems in everyday routine—we already know quite well that a trigger is absolutely needed. Here, already familiar `fuzzytime()` function comes to rescue. Navigate to **Configuration | Hosts**, select **Triggers** from the first dropdown, then click the **Create Trigger** button.

Let's say we have some fairly loaded and critical proxy—we would like to know when three minutes have passed without the proxy communicating back. In such a case, a trigger expression like this could be used:

```
{host:zabbix[proxy,proxy,lastaccess].fuzzytime(180)}=0
```

That's very nice, but if we recall when the proxy connects to the server, there were two cases—it either synchronized the configuration, or sent data. What if, for some reason, all occurrences of both these events are further apart than these three minutes? Nothing to fear. Zabbix proxy is a witty beast, and it also has a heartbeat process, which reports back to the server at regular intervals. Even better, this timing is configurable. Again, take a look at `/etc/zabbix/zabbix_proxy.conf`, this time looking for the `HeartbeatFrequency` variable, which by default looks like this:

```
# HeartbeatFrequency=60
```

Being specified in seconds, this value means that every 60 seconds the proxy will report back to the server, even if there are no new values to send, thus checking on proxy. The `lastaccess` item is quite a reliable way to figure out when a proxy is most likely down or at least inaccessible, even if it should not be sending data for a longer period of time. As usual, the heartbeat interval can be easily customized in the said configuration file; remember to uncomment the line after changing its value.

So for our trigger, fill in the following values:

- **Name:** Proxy \$2 not connected for 3 minutes
- **Expression:** {A Test Host:zabbix[proxy,proxy,lastaccess].fuzzytime(180)}=0

When done, click **Save**.

Remember, we talked about how you need a reliable server to run a Zabbix proxy on? There's also another option, which is more like turn-key solution. You could set up an appliance-like system, some simple, low spec device, that would explicitly run Zabbix proxy. Unless there's a large volume of data being processed, it could be really simple and use an automatically created SQLite database, thus plugging a cheap headless proxy device into some remote network could provide you with nice monitoring access.

Tweaking proxy configuration

While many configuration parameters for a proxy are the same as for the server (the pollers to start, port to listen on, and so on), and some are same as for the agent daemon (hostname), there are some proxy specific parameters. Knowing about these can be helpful when diagnosing a proxy-related problem, or when the proxy must be deployed in a specific environment.

Option	Description
ProxyLocalBuffer	Proxy will keep data in the local database for this many hours. By default all data that is synchronized to Zabbix server is removed.
ProxyOfflineBuffer	Proxy will keep data for this many hours if the Zabbix server is unavailable. By default, data older than one hour is discarded.
HeartbeatFrequency	By default Zabbix proxy sends a heartbeat message to the Zabbix server every minute. This parameter allows to customize that.
ConfigFrequency	By default Zabbix proxy retrieves new configuration from server once an hour. As discussed above, you might want to increase this for large, fairly static setups, or maybe decrease for smaller, more dynamic installations.
DataSenderFrequency	This parameter specifies how often the proxy pushes collected data to Zabbix server. By default it's one second. As all the trigger and alert processing is done by server, it is suggested to keep this value low.



An up-to date proxy configuration option list can be found in the official Zabbix manual and the example proxy configuration file. These also include information on allowed option ranges.

Summary

We have now learned one useful method that will help us with monitoring remote locations that can't be reached directly. Let's rehash the main benefits that a Zabbix proxy provides:

- Only connections from the Zabbix proxy to the Zabbix server on a single port are made, thus allowing us to monitor devices behind a firewall or devices that are inaccessible because of network configuration
- Zabbix server is offloaded from keeping track of checks and actually performing them, thus increasing performance
- Local buffering on the proxy allows it to continue gathering data while the Zabbix server is unavailable, transmitting it all when connectivity problems are resolved

With the listed benefits and being easy to install, a proxy is a very tempting part of the monitoring solution, but it is advisable to keep track of it by using the `lastaccess` Zabbix internal item.

When changing a host to be monitored by a proxy, make sure to change `Server` parameter in `zabbix_agentd.conf` on the monitored host, and also `ServerPort`, if applicable. Otherwise the proxy won't be allowed to connect to the agent, and the agent would still send active check data to the Zabbix server. This doesn't apply to agent-less monitored systems, where moving them to a proxy is as easy as flipping a configuration parameter in the frontend.

13

Working Closely with Data

Using a web frontend and built-in graphing is nice and easy, but sometimes you might want to perform some nifty graphing in an external spreadsheet application, or maybe feed data into another system. Sometimes you might want to make some configuration change that is not possible or is cumbersome to perform using the web interface. While that's not the first thing most Zabbix users would need, it is handy to know when the need arises. Thus, in this chapter we will find out how to:

- Get raw data about monitored metrics from the web frontend or database
- Perform some simple, direct database modifications
- Use XML export and import to implement more complex configuration

Getting raw data

Raw data is data as it's stored in the Zabbix database, with minor, if any, conversion performed. Retrieving such data is mostly useful for analysis in other applications.

Extracting from the frontend

In some situations it might be a trivial need to quickly graph some data together with another data that is not monitored by Zabbix (yet, you plan to add it soon of course), in which case a quick hack job of spreadsheet magic might be the solution. Now, the easiest way to get data to be used outside of the frontend is actually frontend itself.

Let's find out how we can easily get historical data for some item. Go to **Monitoring** | **Latest data** and select **A Test Host** from the **Host** dropdown. We'd like to plot load on this system against some external data in external application, so click on **Graph** next to **CPU Load**. That gives us a well known graph interface. Well, that wasn't what we wanted, now was it? But this interface allows us access to raw data easily using the dropdown at the top-right corner, **Values**.

[ If you just want to quickly look at the last few values, choose **500 latest values** instead. It will get you the data with less clicks.]

Here, one thing worth paying attention to is the time period controls at the bottom, which are the same as the ones available for graphs, screens, and elsewhere. Using the scrollbar, zoom, move, and calendar controls, we can display data for any arbitrary period. For this item, the default period of one hour is fine; for some items that are polled less frequently we will often want to use much larger periods.

A Test Host: CPU Load		Values	As plain text
Timestamp	Value		
2010.Mar.08 18:51:52	0.01		
2010.Mar.08 18:51:22	0.02		
2010.Mar.08 18:50:52	0.04		
2010.Mar.08 18:50:22	0.07		
2010.Mar.08 18:49:52	0.03		
2010.Mar.08 18:49:22	0.05		
2010.Mar.08 18:48:52	0.09		
2010.Mar.08 18:48:22	0.03		
2010.Mar.08 18:47:52	0.05		
2010.Mar.08 18:47:22	0.08		
2010.Mar.08 18:46:52	0		
2010.Mar.08 18:46:22	0		
2010.Mar.08 18:45:52	0		
2010.Mar.08 18:45:22	0		

Zoom: [1h](#) [2h](#) [3h](#) [6h](#) [12h](#) [1d](#) [1w](#) [2w](#) [1m](#) 08.03.2010 17:52 - 08.03.2010 18:52
[!\[\]\(18a4464e535687c52071544f51d0f611_img.jpg\)](#) [1m](#) [1w](#) [1d](#) [12h](#) [1h](#) | [1h](#) [12h](#) [1d](#) [1w](#) [1m](#) [!\[\]\(f2a38732cb8e14b6209687108786624f_img.jpg\)](#)
[!\[\]\(c66cb529ced877624196f517a21adc23_img.jpg\)](#) [01h 00m](#) [\(fixed\)](#)

While we could copy data out of this table with a browser that supports HTML copying, then paste it into some receiving software that can parse HTML, that is not always feasible. A quick and easy solution is in the upper-right corner—just click the **As plain text** button.

This gives us the very same dataset, just without all the HTML-ish surroundings like the Zabbix frontend parts and the table. We can easily save this representation as a file or copy data from it and reuse in a spreadsheet software or any other application. Additional benefit this data provides—all entries have corresponding Unix timestamps listed as well.

Querying the database

Grabbing data from the frontend is quick and simple, but this method is unsuitable for large volumes of data and hard to automate—parsing the frontend pages can be done, but isn't the most efficient way of obtaining data. Another way to get to the data would be directly querying the database. Let's find out how historical data is stored. Launch a MySQL command line client (simply called `mysql`, usually available in the path), and connect to the database `zabbix` as user `zabbix`.

```
$ mysql -uzabbix -p zabbix
```

When prompted, enter the `zabbix` user's password (which you can remind yourself of by looking at the contents of `/etc/zabbix/zabbix_server.conf`) and execute the following command in the MySQL client:

```
mysql> show tables;
```

This will list all tables in the `zabbix` database—exactly 88 in Zabbix version 1.8.1. That's a lot of tables to figure out, but for our current need (getting some historical data out) we will only need a few. First the most interesting ones—tables that contain actual data. All historical data is stored in tables whose names start with `history`. As you can see, there are many of those with different suffixes—why is that? Zabbix stores retrieved data in different tables depending on data type. The relationship between types in the Zabbix frontend and database is as follows:

- `history` – **Numeric (float)**
- `history_log` – **Log**
- `history_str` – **Character**
- `history_text` – **Text**
- `history_uint` – **Numeric (unsigned)**

So to grab the data we first have to find out the data type for that particular item. The easiest way to do that is to open **Configuration | Hosts** in the frontend, then choose **Items** in the first dropdown. Open filter, click on **Select** next to the **from Host** field and choose **A Test Host**, then click **Filter**. In the list, click on **CPU Load** in the **Description** column and observe the **Type of information** field value—it's set to **Numeric (float)**. So, according to our table-type mapping, data for this item should be in the `history` table. Back in the MySQL client, we can try peeking at the contents of the `history` table by retrieving all fields for the first three records:

```
mysql> select * from history limit 3;
```

The output will show us that each record in this table contains three fields (your output will have different values):

```
mysql> select * from history limit 3;
+-----+-----+-----+
| itemid | clock      | value   |
+-----+-----+-----+
| 22177 | 1256811187 | 0.0000 |
| 22162 | 1256811202 | 0.0000 |
| 22177 | 1256811217 | 0.0000 |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

The last field, `value`, is quite straightforward—it contains the retrieved value. The `clock` field contains the timestamp in Unix time—the number of seconds since the so-called Unix epoch, 00:00:00 UTC on January 1, 1970.



An easy way to convert the Unix timestamp to a human-readable form that does not require an internet connection is using the GNU date command—`date -d@<timestamp>`. For example, `date -d@1234567890` would return `Sat Feb 14 01:31:30 EET 2009`.

The first field, `itemid` is the most mysterious one. How can we determine which ID is for the item we are interested in, CPU Load on A Test Host? Again, the easiest way is to use the help of the frontend. You should still have the item properties page open in your browser, so take a look at the address bar. Along with other variables you'll see part of the string that reads like `itemid=22162`. Great, so we already have the `itemid` on hand. Let's try to grab some values for this item only from the database:

```
mysql> select * from history where itemid=22162 limit 3;
```



Use the `itemid` that you obtained from the page URL!

```
mysql> select * from history where itemid=22162 limit 3;
+-----+-----+-----+
| itemid | clock      | value   |
+-----+-----+-----+
| 22162 | 1256811202 | 0.0000 |
| 22162 | 1256811232 | 0.0000 |
| 22162 | 1256811262 | 0.0000 |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

The resulting set contains only CPU load values from the exact item, as evidenced by `itemid` field.

One usually will want to retrieve values from a specific period. Guessing Unix timestamps isn't entertaining, so we can again use `date` command to figure out the opposite—a Unix timestamp from a date in human-readable form.

```
$ date -d "2010-08-13 13:13:13" "+%s"
1281694393
```



The `-d` flag tells the `date` command to show the specified time instead of the current time, and the `%s` format sequence instructs it to output in Unix timestamp format. This fancy little command also accepts more freeform input like "last Sunday" or "next Monday".

As an exercise, figure out two recent timestamps half an hour apart, then retrieve values for this item from the database. Hint—the SQL query will look similar to this:

```
mysql> select * from history where itemid=22162 and clock > 1250158393
and clock < 1250159593;
```

You should get back some values. To verify the period, convert the returned clock values back to a human-readable format. The obtained information can be now passed to any external applications for analyzing, graphing, or comparing.

With `history*` tables containing the raw data, we can get a lot of information out of them. But sometimes we might want to give a bigger picture only and that's when table `trends` can help. Let's find out what exactly this table holds. In the MySQL client, execute:

```
mysql> select * from trends limit 2;
```

We are now selecting two records from the `trends` table.

itemid	clock	num	value_min	value_avg	value_max
18440	1239786000	93	754.0000	1361.3610	4608.6250
18440	1264579200	115	120.0000	4096.9594	58706.0000

2 rows in set (0.00 sec)

Here we find two familiar friends, `itemid` and `clock`, whose purpose and usage we just discussed. The last three values are quite self-explanatory—`value_min`, `value_avg`, and `value_max` contain minimal, average, and maximal values of the data. But for what period? The `trends` table contains information on one hour periods. So if we would like to plot the minimal, average, or maximal values per hour for one day in some external application, instead of recalculating this information, we can grab data for this precalculated data directly from the database. But there's one field we have missed, `num`. This field stores how many values were there in the hour that is covered in this record. It is useful if you have hundreds of records each hour in a day that are all more or less in line, but for one hour data is missing, except a single extremely high or low value. Instead of giving the same weight to every hour values when calculating daily, weekly, monthly, or yearly data, we can more correctly calculate the final value.



If you want to access data from the database to reuse in external applications, beware of the retention periods - data is removed from `history*` and `trends` tables after days, specified in the `Keep history` and `Keep trends` field for the specific items.

Using data in a remote site

That covered data retrieval on the Zabbix server. But what if we have a remote site, a Zabbix proxy, a powerful machine, and a slow link? In situations like that, we might be tempted to extract proxy data to re-use it. However, the proxy stores data in a different way than Zabbix server.

We had set up our Zabbix proxy with an SQLite database located at `/tmp/zabbix_proxy.db`. But by default, proxy only stored data for a moment in there until the data was synchronized to the Zabbix server. What do we do now? Open `/etc/zabbix/zabbix_proxy.conf` on the Zabbix proxy server as root and look for the two configuration parameters named `ProxyLocalBuffer` and `ProxyOfflineBuffer`. While we covered the latter in the previous chapter, the former is the interesting one here. This parameter controls how many hours the Zabbix proxy keeps data in the database after it is already synchronized with the server, and by default it is 0—so no synchronized data is kept. Change this line to read:

```
ProxyLocalBuffer=1
```

Restart the Zabbix proxy. Now the proxy will keep data for one hour even after it has been synchronized back to the server.

But where exactly? Let's try to look at the data contained in the database. For that we will need to open the SQLite command line client, which usually is located in the package `sqlite`. On the Zabbix server, execute:

```
$ sqlite3 /tmp/zabbix_proxy.db
```

This opens the specified database. We can look at what tables are present by using the `tables` command:

```
sqlite> .tables
```

We will see lots of tables, so it's like some quest to find the data there. While we are playing with the proxy, let's make life harder for it—stop the Zabbix server. Now we should be able to see the effect of both buffering configuration parameters. First we have to know how the Zabbix proxy knows what data is synchronized. In the SQLite client, run:

```
sqlite> select * from ids;
```

From this table we can find out some interesting information.

```
0|proxy_history|history_lastid|92
```

The second field shows us the table where the proxy actually stores its data—in a table named `proxy_history`. The last field, the ID, is the ID that is synchronized. Of course, in your database the ID will be different. That wasn't crystal clear, now was it? To get some more clarity, execute:

```
sqlite>select * from proxy_history;
```

This will output a bunch of quite cryptic rows. In these rows, the first value is ID, thus we can figure out how far the proxy had got with the synchronization, before we took Zabbix server from it.

```
1|22177|1250255828|0||0|0.140000|0
2|22175|1250255828|0||0|0.000000|0
...
91|22242|1250256162|0||0|0|0
92|22246|1250256166|0||0|0|0
93|22195|1250256175|0||0|0|0
94|22177|1250256187|0||0|0.000000|0
...
282|22195|1250256985|0||0|0|0
283|22171|1250256991|0||0|0|0
```

From a quick glance it looks like in this example we have 283 records, of which all records up to number 92 were synchronized to the Zabbix server. If we had not set `ProxyLocalBuffer` in the configuration file, records before number 92 would have already been removed from the database. But how will the Zabbix proxy determine when to remove locally buffered records? A quick look at the table fields might help.

Field	Usage
<code>id</code>	Record ID, used to determine which records have been synchronized back to the server.
<code>itemid</code>	This is item ID in question as it appears on the Zabbix server.
<code>clock</code>	Unix time of the record, using proxy host time.
<code>timestamp</code>	Relevant for Windows eventlog monitoring only, timestamp as appears on the monitored machine.
<code>source</code>	Relevant for Windows eventlog monitoring only, eventlog source.
<code>severity</code>	Relevant for Windows eventlog monitoring only, eventlog severity.
<code>value</code>	This is actual value of the monitored metric.
<code>logeventid</code>	Relevant for Windows eventlog monitoring only, event ID.

Thus we can see that the `id` field will be used to determine when records should be purged because they are already synchronized to the server and `ProxyLocalBuffer` time period has passed, or `ProxyOfflineBuffer` time has passed, even if the records are not synchronized.



The proxy doesn't have much information on items, you'll need to snatch that from the Zabbix server if you are doing remote processing. For example, item descriptions are not available in the proxy database.



Diving further in the database

With some knowledge on how to extract historical and trends data from the tables, we might as well continue looking at other interesting, and relatively simple things which we can find and maybe even change directly in the database.

Managing users

We saw how managing users was an easy task using the frontend. But what if you have forgotten the password? What if some remote installation of Zabbix is administered by local staff, and the only Zabbix super admin has left for a month long trip without a phone and nobody else knows the password? If you have access to the database, you can try solving such problems. Let's find out what and how exactly Zabbix stores data regarding users. In the MySQL console, execute:

```
mysql> select * from users limit 2;
```

This way, we are listing all data for two users.

userid	alias	name	surname	passwd	url	autologin
1	Admin	Zabbix	Administrator	5fce1b3e34b520afeffb37ce08c7cd66		0
2	guest	Default	User	d41d8cd98f00b204e9800998ecf8427e		0

2 rows in set (0.00 sec)



Please note that this screenshot is trimmed at the right edge.

That's a lot of fields. We'd better find out what each of them means.

Field	Usage
userid	Quite simple, that's a unique, numeric ID.
alias	More commonly known as a username or login name.
name	User's name, usually populated by given name.
surname	This surely can't be anything else but surname.
passwd	Password hash is stored here. Zabbix stores MD5 hashes for authentication.
url	URL after login is stored in this field.
autologout	Whether auto-logout for this user is enabled. Non-zero values indicate timeout.
lang	Language for the frontend.
refresh	Page refresh in seconds. If zero, page refresh is disabled.
theme	Frontend theme to use.
attempt_failed	How many consecutive failed login attempts there have been.
attempt_ip	IP of the last failed login attempt.
attempt_clock	Time of the last failed login attempt.
rows_per_page	How many rows per page are displayed in long lists.

As we can see, many of the fields are options that are accessible from user profile or properties page, although some of these are not directly available. We mentioned password resetting before, let's look at simple method to do that. If passwords are stored as an MD5 hash, we must obtain that first. A common method is command line utility `md5sum`. Passing some string to it will output the desired result, so we can try executing:

```
$ echo "somepassword" | md5sum
531cee37d369e8db7b054040e7a943d3 -
```

The MD5 hash is printed, along with a minus sign, which denotes standard input. If we had run `md5sum` on a file, the filename would have been printed there instead.

Now, the problem is, if we try to use this string as a password hash, it would fail. In this case, the hash is calculated on the passed string, including the newline at the end. For a correct version, we have to pass the `-n` flag to `echo`, which suppresses the trailing newline:

```
$ echo -n "somepassword" | md5sum  
9c42a1346e333a770904b2a2b37fa7d3 -
```

Notice the huge difference in the resulting string. Great, now we only have to reset the password:



The following statement changes the Zabbix administrative user password. Do not perform this on a production system, except in an emergency situation.



```
mysql> update users set passwd='9c42a1346e333a770904b2a2b37fa7d3' where user_id='1';  
Query OK, 1 row affected (0.01 sec)  
Rows matched: 1 Changed: 1 Warnings: 0
```

From now on you should be able to log into Zabbix frontend as admin/`somepassword` – try it out. Feel free to change the password back after that.

Well, there's actually an easier method available. MySQL has built-in syntax for calculating MD5 hash, so all this trickery could be replaced with a simpler approach:

```
mysql> update users set passwd=MD5('somepassword') where alias='Admin';
```

The command line utility provides a nice way to check various sequences. For example, try to figure out what the default guest password hash, `d41d8cd98f00b204e9800998ecf8427e` represents.

We also mentioned making some user Zabbix super admin , if you were to be tasked with remotely making this change. This change is fairly trivial—all we have to do is change single number.

```
mysql> update users set type='3' where alias='wannabe_admin';
```

And that's it, the user `wannabe_admin` will become the Zabbix super admin.

Converting a host to a template

So you did like instructed in the first chapters, and created items directly on a host. Instead of finding out soon enough about templates, you created lots and lots of items. Now you want to make this host available as a template, but can't find such functionality in the frontend. Let's see whether this can be easily done at all.

First we should create a host to play with. In the frontend, open **Configuration | Hosts**, click on **A Test Host** in the name column, then click on **Full clone** button. Change the **Name** field to read **A Test Template**. Mark both groups in the **In Groups** area and click **>>** button, mark **Custom Templates** in the **Other Groups** area and click **<<** button. We have done all that's possible on this form to make it look like a template, so click **Save**.

Now select **Templates** in the first dropdown and make sure **Custom Templates** is selected in the **Group** dropdown. Well, we don't see our new "template" in the list, because it is not a real template yet. To make it appear, we have to directly edit the database. In the MySQL console, which you still should have open, execute:

```
mysql> select host,status from hosts where host='A Test Template' or
host='C_Template_Linux';
```

In the output we can see a great hint:

host	status
A Test Template	0
C_Template_Linux	3

2 rows in set (0.00 sec)

It looks like the `status` field value of 0 denotes a normal host, while 3 denotes a template. But it seems like there should be more values here. We can find a list of the other values in the frontend file `include/defines.inc.php`. Open this file in some editor and search for `HOST_STATUS_MONITORED`. You'll find entries that will explain what each of the constants mean. From that, we can learn the following:

Field	Usage
0	These are monitored hosts.
1	These are hosts that are not monitored. Thus both 0 and 1 denote normal hosts.
2	While file refers to this code as being for unreachable hosts, this is not currently used.
3	As we deduced, these are templates.
4	Deleted hosts. Housekeeper process will take care of them.
5	These are proxies.



There is an important lesson to be learned here—normal hosts, templates, and proxies are basically handled the same way in Zabbix.

Our host currently has status 0, and templates have status 3. Easy to see what we must do. In MySQL console, enter:

```
mysql> update hosts set status=3 where host='A Test Template';
```

That completes successfully.

```
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Back in the frontend, refresh the templates list—hooray, our freshly converted template appears in the list as intended.

Changing existing data

While usually once the data has been gathered you won't have a need to change it, there might be some rare cases when that might be required. Back in *Chapter 3* we created items for network traffic monitoring, and we gathered data in bytes, but in network management usually bits per second are used instead. While it would often be possible for you to simply reconfigure the items and clear the old data, what if you must preserve already gathered values? Directly editing the database might be the only solution.

Before doing that you would have to modify the item in question. If data is coming in bytes, but we want bits, what do we do? Right, we configure multiplier for that item and set the multiplier to be 8. Additionally change units to b (bits) while performing the change.

When performing the change to the item, take a quick look at a clock.

While that would deal with all future incoming values, it would leave us with inconsistent data before that moment. As we can't delete it, we must find some way to fix it. Our problem is twofold:

- We have incorrect data in the database
- We have both incorrect and correct data in the database (old and new values)

This means that we can't simply convert all values, as that would break the new, correct ones.



If you have set any triggers based on traffic amount, do not forget to change those as well.



Finding out "when"

Figuring out the moment when correct information started flowing in can be most easily done by looking at the frontend. Navigate to **Monitoring | Latest data**, click on **History** for that item and then select **Values** or **500 latest values**. Look around the time you changed item multiplier, plus a minute or so, and check for notable change in scale. While it might be hard to pinpoint the exact interval between two checks (network traffic can easily fluctuate over eight times in value between two checks), there should be a pretty constant increase in values. Look at the times to the left of values and choose a moment between first good value and last bad value.

"When" in computer language

But as we now know all time related information in the Zabbix database is stored as Unix timestamps – how can we find out what the timestamp is for this time? For that, the GNU date command can help again. Execute on the Zabbix server the following, by replacing exact time with what you deducted from latest values:

```
$ date -d "2010/02/13 13:13:13" "+%s"
```

That will output Unix timestamp of that moment, which in the case of this example would be 1266059593.

Finding out what

Now that we know the exact time which limits the change, we must also know which item we must modify for it. Wait, but we do know that already? Almost. What we need is the item ID to make changes to the database. The easiest way to find it out is by opening item properties in configuration section and copying ID from URL, like we did before.

Performing the change

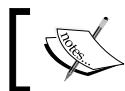
By now we should have two cryptic-looking values:

- Time in Unix timestamp format
- Item ID

What we have to do now? Multiply by 8 all the values for the item ID before that timestamp. With data we have, it is actually quite trivial—in MySQL console we would execute:

```
mysql> update history set value=value*8 where itemid='<our ID>'  
      and clock<'<our timestamp>';
```

We are updating table `history`, not `history_uint`, because data for network traffic is stored as speed per second, not as the direct counter value we receive (the one that always increases). This single query should be enough to convert all the old data to bits.



If you have lots of historical data in total and for this item, such a query can take some time to complete.



Using XML import/export for configuration

The web frontend is an acceptable tool for making configuration changes to a Zabbix server, unless you have to do lots of modifications that are not made easier in the frontend with methods like mass update. While Zabbix 1.8 introduced API, as of this writing there are no complete tools that take advantage of it. Additionally, the Zabbix wiki at <http://www.zabbix.com/wiki/> has various contributed templates available, as well as scripts to automatically generate XML configuration files. Let's look at how a simple roundtrip would work.

Exporting initial configuration

Before we can proceed, let's export the configuration file. In the frontend, open **Configuration | Export/Import** and make sure **Custom Templates** is selected in the **Group** dropdown. In this view, mark the checkbox next to **C_Template_Email_Server**. Make sure the action dropdown at the bottom says **Preview** and click **Go**. In the next screen, notice how all elements to be exported are nicely listed, select **Export** in the action dropdown and click **Go** again. You are offered to save a file named `zabbix_export.xml`—save it somewhere on your local machine

Modifying configuration

Now, with the file in hand, we can modify this configuration. This method gives us free reign on host and host attached information, so modifications are limited only by Zabbix's functionality and our imagination.

XML export format

Open the saved XML export in your favorite editor. In this file you'll see all the data that configuration host has, and the file will start like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<zabbix_export version="1.0" date="09.03.10" time="08.53">
<hosts>
    <host name="C_Template_Email_Server">
        <proxy_hostid>0</proxy_hostid>
        <useip>1</useip>
        <dns></dns>
        <ip>127.0.0.1</ip>
        <port>10050</port>
        <status>3</status>
        <useipmi>0</useipmi>
        <ipmi_ip>127.0.0.1</ipmi_ip>
        <ipmi_port>623</ipmi_port>
        <ipmi_authtype>0</ipmi_authtype>
        <ipmi_privilege>2</ipmi_privilege>
        <ipmi_username></ipmi_username>
        <ipmi_password></ipmi_password>
        <groups>
            <group>Custom Templates</group>
        </groups>
        <triggers>
            <trigger>
                <description>WEB service is down</description>
```

As can be seen, it's a basic XML file with hosts coming as the first level and then having respective attached elements listed. So each host is contained in a `<host>` block, which in turn has blocks for all things attached to that host. The format is simple and most things should be obvious simply from taking a glance at the XML, and maybe sometimes by comparing values in XML with values in frontend configuration section. An exception might be values available for each field, although we already looked at some of those, for example, converting a host to a template could also be performed this way, looking at the XML excerpt above we can see `<status>` block, which simply mimics the `status` field we changed in the database. So other way to convert a host to template would be to export that host, change `<status>` value, and import it back.

While we look at the exported template, we can see the same information that an exported host would have, including template linkage – that's what the `<templates>` block denotes.

Script around the export

While manually making a single change to an exported file can be handy, it's the large changes that expose the benefit of this approach best. While it is possible, and usually easier on large scale, to use specific XML tools or libraries, for a quick hack job on XML we can use simple shell scripts.

One very common problem is a switch or router monitoring. Such devices tend to have many ports, and manually clicking through the interface to create 48 or more items, multiplied by metrics to be monitored on each port soon becomes physically impossible. Using XML export, we can script a file to be imported, thus easily solving such configuration nightmares.

The easiest scenario would be to create all items for a single port, then export this example host and write a quick script that loops over these item values and creates items for the remaining ports. In most cases it would be enough to change the item key or SNMP OID to contain the corresponding number, although the description might also have to be changed if it was not possible to use positional parameters like \$1, \$2, and so on.

When needed, continue to do the same for triggers and graphs as well. Again, it's best to create all data for single element (like a port), then export it, and examine it to find out how these should be put back together.

Other large scale problems that can be solved by an XML roundtrip:

- **Adding lots of devices**

If you are given a large list of switches with IP addresses, adding them all through the interface is a monstrous task. With XML, it becomes a very easy and quick one instead. To do that, simply create single host, linked against the previously created template or several (which used XML export to create items for all the ports), then export it to get some sort of a template. In this export, you'll basically have to change a couple of values only—notably `<ip>` and `name` inside the `<host>` element. Then just proceed to create a loop that creates new `<host>` entries with corresponding IP and host name data. Note that you can only specify host information in this file—all items, triggers, and graphs will be attached, based on information that is contained in the template or templates, specified in `<templates>` block.

- **Creating many graphs with lots of arbitrary items**

Sometimes it might be required to create not only one graph per port, like we might do in the above switch example, but also graphs, grouping items from several devices and other arbitrary collections. Again, export an example host (you can only mark graphs for export) and script in a loop graph items—these are located in `<graph_elements>` block.



A graph with a huge amount of items can soon become unreadable. Don't overdo items on a single graph.

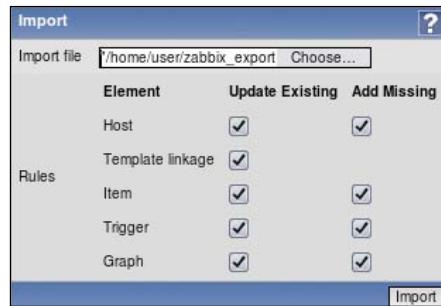
As you might have guessed, these actions are more or less possible to do by directly editing the database and using the new API, but XML export and import has been around longer than API and thus is better documented. Editing the database directly is very likely to corrupt it, often in ways not visible immediately after the corruption occurs. Slightly abusing the XML process is the safest way, at least for now.

Importing modified configuration

For our XML export, we won't do large scripting. Instead, let's make a simple modification. In the saved `zabbix_export.xml`, find item with key `vfs.file.exists[/tmp/severity3]`. Copy all elements for this item block and insert them below (you can safely skip SNMP related entries), then change item key to read `vfs.file.exists[/tmp/severity4]`. A minimum item entry in this case could look like this:

```
<item type="0" key="vfs.file.exists[/tmp/severity4]" value_type="3">
  <description>Link $1</description>
  <delay>30</delay>
  <history>90</history>
  <trends>365</trends>
  <status>0</status>
  <data_type>0</data_type>
</item>
```

Save this file. Now on to the actual import process. Back in the frontend, in the **Configuration | Export/Import** section, choose **Import** in the upper-right hand dropdown. In this form, click **Choose** next to the **Import file** field and choose the saved `zabbix_export.xml`. Feel free to explore the **Rules** section, although the defaults will do for us. The only value we are interested in is missing items, and the checkbox in the **Add Missing** column next to **Items** is already marked.



Now just click **Import** to proceed. This should complete successfully, so click on **Details** in the upper-left corner. While all other records will be about updating, find the entries mentioning `severity4`. These will be the only ones who will actually add an item, and as we are adding it for a template, it also gets in turn added to all other hosts and templates that are linked against this one.



Let's verify that this item was successfully added. Navigate to **Configuration | Hosts**, make sure **Linux servers** is selected in the **Group** dropdown, and click on **Items** link next to the **Another Host** entry. Our new item should be visible in the item list, showing that it has been correctly added to the linked host. Remember, we only added it to the upstream template in our import process.



Summary

In this chapter we dived deeper into the internal data structures Zabbix uses. While that's still just a small part of a different database and other information, it should help with some of the common problems users encounter at first.

We figured out how to get raw data from the frontend, which is the easiest method for small datasets. For bigger amounts of data, we learned to grab data from different history tables, depending on data type. We also found out how to reuse data in a remote location from a proxy, especially making sure that the proxy keeps data in the database after having it synchronized back to the Zabbix server—configuration parameter `ProxyLocalBuffer` helped there. All this should be of use whenever data gathered by Zabbix has to be reused in other applications.

For situations when less precision is needed, we learned about the trends table and the calculation of hourly minimal, maximal, and average values that are stored there.

And in the end we looked at a robust way to add large amount of elements like items, triggers, graphs, or hosts to Zabbix configuration by using XML export and import functionality. We looked at XML format in brief and learned about basic elements that are useful when creating files for re-importing later.

14

Upgrading Zabbix

While some users choose to use distribution package systems for Zabbix, some features and bug fixes in latter versions make for compelling reasons to upgrade more frequently.

Even for users that stick to distribution packages, it is useful to have a deeper knowledge of the upgrade process and possible consequences. In this chapter we will look at:

- General upgrade and versioning policy – what versions can be upgraded to what, do version numbers mean anything, and so on.
- What a patchlevel upgrade includes (that is, upgrading, for example, from 1.8.1 to 1.8.2).
- What should be done for a minor or major version upgrade (for example, going from 1.6 to 1.8 or from 1.8 to 2.0).
- Compatibility between different Zabbix components. We'll find out whether we can run different versions of components together, and if yes, what exactly.

General policy

Most software packages have general guidelines when it comes to their upgrade cycle, sort of a policy. To discuss them further, let's look at Zabbix version number scheme first.

Zabbix versions

Zabbix version numbers follow the following numbering scheme:

`major.minor.<change/patch>`

This results in a three number version string, for example, 1.6.6 or 1.8.0. Since version 1.3/1.4, Zabbix follows the odd/even numbering system that denotes development versions with odd numbers and stable versions with even version numbers. For example, 1.7 would be a development version that later becomes 1.8 as the stable version.

Version upgrades

When upgrading Zabbix, this process can vary depending on what level of upgrade is performed. If a minor or major version change happens, it will require at least upgrading the Zabbix server executable, and frontend files, applying a database patch and, most likely, also upgrading any Zabbix proxy instances.

So what's a database patch? As time passes, Zabbix developers create new features and improve existing ones. Sometimes these changes require changes to the underlying database structure. When users upgrade, they want to keep the existing configuration and data, thus a patch is provided that brings the old database to the new form. While minor version upgrades always require a database patch, **change level version upgrades never require database patching**.

While sometimes SQL queries might be provided that add indexes to improve performance, they are not mandatory for Zabbix functioning. As a result, change level upgrades should be very easy, safe, and require no downtime.

As can be seen from the current version number, Zabbix has not advanced a major version number yet, but you can be sure that there will be a database patch to apply when it does.

Upgrading Zabbix

Let's see what an actual full upgrade involves. As the upgrade process differs depending on what level upgrade is performed, let's look at what steps are required in each case.

Change level upgrade

This is the simplest case. As mentioned, no database patching is required, so only the Zabbix code has to be updated. To perform an upgrade, follow these steps:

1. Review the release notes. This is important, because even a change level upgrade can introduce some minor difference that might be good to know beforehand. Look for suggested database changes like adding new indexes or updating existing ones, and determine whether those would be of interest to you.

2. Retrieve the new version from the Zabbix homepage.
3. Compile Zabbix, create a package. Follow the same steps as we did when first installing Zabbix.
4. Backup the database. While this upgrade should be safe enough, it is always a good idea to create a backup.
5. Stop the server – this might not be required with some distribution packages, but it's better to be safe here.
6. Upgrade the Zabbix package. Use distribution specific package management tools for this.
7. If decided, add database indexes or other minor improvements.
8. Start Zabbix server.
9. Replace the frontend files.



When using MySQL, it is suggested to pass the `--single-transaction` flag to `mysqldump` command. This way `mysqldump` will get a consistent view of the database.

Now, without touching the database in the majority of cases, you should have successfully upgraded and should be able to access the new frontend. In this whole process, two steps might create confusion that we haven't looked at in detail yet.

Adding the indexes

Step 7 was about adding the indexes, if they are mentioned in the release notes and considered useful enough. For a first time user those few lines can be confusing, so let's look at ways to apply them. If presented in the notes, these instructions will usually look like some simple lines, for example:

```
CREATE INDEX graphs_items_2 on graphs_items (graphid);
```

This is one line from the real life release notes for version 1.6.5. Now, to apply such changes to our MySQL database we would launch a MySQL command line client and indicate that we want to work with the Zabbix database:

```
mysql> use zabbix
```



Do not perform these commands on your test installation. This is just an example.

Then we would simply copy and paste the commands, given in the release notes:

```
mysql> CREATE INDEX graphs_items_2 on graphs_items (graphid);
```

Note the trailing semicolon, it is mandatory and important.



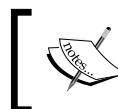
You can also skip the use command and add database directly into the command, like `CREATE INDEX zabbix.graphs_items_2 on graphs_items (graphid);`.



Another alternative would be to copy and paste all mentioned commands to a file, which we could call, for example, `zabbix_database_indexes`. Then, to apply the changes, one would execute from the shell:

```
$ mysql zabbix -u Username -p < zabbix_database_indexes
```

The string `Username` would have to be replaced by a username that has write permissions on the Zabbix database. The string `zabbix` that appears directly after `mysql` command is the name of the database on which to perform the changes, contained in the file.



Remember that you do not have to apply database patches in `upgrade/dbpatches` directory when upgrading from one patch level version to another. It will most likely break your database.



Replacing frontend files

With the main executables (and optionally, database) dealt with, a frontend upgrade is important as well. When performing step 9 during the change level upgrade, it is advised to keep both the old and new frontend version. So if your relative frontend path is `zabbix/`, rename that directory to `zabbix-<old_version>`. Place the new frontend config files in `zabbix-<new_version>` and simply create a symlink so that you don't have to use a different URL whenever upgrading. Let's say we are upgrading from version 1.8.1 to 1.8.2, so we would execute the following in the web server documents directory:

```
# mv zabbix zabbix-1.8.1
# ln -s zabbix-1.8.2 zabbix
```

Next we have to preserve the old frontend configuration file and copy it in place for the new frontend version:

```
# cp zabbix-1.8.1/conf/zabbix.conf.php zabbix-1.8.2/conf/zabbix.conf.php
```

That should be enough. Now you can refresh the Zabbix frontend and check the page footer – the new version number should be visible there.

This approach with keeping old frontend versions is useful if a new version turns out to have a problem and you would like to test whether the old version also had the same problem – just load up URL in your browser that points to the old frontend directory. If the problem indeed turns out to be a regression, simply change the symlink to point to the old directory and you have reverted to the old version.



If you have modified the `defines.inc.php` file, make sure to perform the same modifications on the new version of the file.



Minor or major level upgrades

With change level upgrades covered, at some point you will happen to be in a situation where a new minor level release becomes available, or even a new major level one. Although it probably won't matter that much – as we will see, a so-called minor level upgrade actually will involve one or more significant changes. Although more involved than a change level upgrade, this process still consists of quite simple steps.

1. Review the release notes. These will outline new functionality and possible large architectural and compatibility changes.
2. Retrieve the new version from the Zabbix homepage.
3. Compile Zabbix, create a package. Follow the same steps as we did when first installing Zabbix.
4. Backup the database. This is mandatory.
5. Stop the Zabbix server.
6. Upgrade the Zabbix package. Use distribution specific package management tools for this.
7. Patch the database.
8. Start the Zabbix server.
9. Replace the frontend files.

While the process is very similar to the process outlined earlier, there are two steps with additional importance.

Patching the database

Patching the database is probably the most important step here. So what is it and why is it needed?

While the Zabbix team works hard to keep change level upgrades without database changes, between minor releases it's open season. Changes to the database schema and its contents are made to accommodate new features, improve performance, and increase flexibility. For some reason, users do not appreciate it if they can't keep gathered data and created configuration in the name of new greatness, so with each new upgrade a database patch is provided. These patches bring the schema in line with the new requirements and migrate the data where appropriate and possible. This may include adding new tables, new rows, removing tables and rows, and changing the data layout.

Sometimes an upgrade might involve removing database indexes. As that is not easy to do in an automated fashion, in such cases the release notes will inform you that you must remove them manually. That process is very similar to the previously discussed one for adding an index. Usually you will be instructed to ignore errors during index removal, as these indexes might be missing in your database but that is a completely normal situation. To find out whether you should remove some indexes, always carefully examine the release notes.

Let's look at a snippet of an actual patch that is used to upgrade an existing Zabbix 1.6 installation to version 1.8.

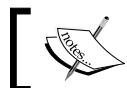
```
alter table users add rows_per_page integer DEFAULT 50 NOT NULL;
alter table usrgrp add api_access integer DEFAULT '0' NOT NULL;
alter table usrgrp add debug_mode integer DEFAULT '0' NOT NULL;
```

These specific lines are here because of added functionality. Before version 1.8 it was not possible to configure lines per page if there are many entities to display (remember the user preferences we looked at in the beginning). Then next two statements deal with additional properties or rights that user groups can have.

Oh, did you notice step 4, the mandatory database backup? That's no joke. While upgrades are extensively tested, it is not possible for the developers to test all scenarios. What has worked for a thousand people might break in some obscure way for you. Additionally, interrupting the upgrade process because of hardware or electricity failure is basically guaranteed to leave your database in a broken state. You have been warned, so get that backup ready.

Of course, you are strongly encouraged to test the upgrade on a test installation, preferably using a production dataset (maybe with trimmed history and trends data, if the available hardware does not permit testing with a full copy).

With a fresh backup created we are ready to engage the patching. Database patches for Zabbix are shipped in the `upgrades/dbpatches` directory. We can see subdirectories like `1.4`, `1.6` and `1.8` in that directory. Each directory denotes the version that the patch moves to from the previous stable version. So if you are tasked with upgrading Zabbix `1.6`, you would only apply the database patch from the `1.8` directory. If you instead needed to upgrade version `1.1`, you would first apply patch from the `1.4` directory, then the one from the `1.6` directory—the order is important here. Once the next version after `1.8` is released, one would use the patch denoted with that version.



Do not perform these commands on your current test installation. This is just an example for a case when an actual upgrade needs to be performed.



To apply the patch, change to the directory containing it on the Zabbix server. For example, if you are using MySQL and upgrading from Zabbix version `1.6` to `1.8` that would be `upgrades/dbpatches/1.8/mysql/patch`. In this directory, one would execute the following command:

```
$ mysql zabbix -u zabbix -p < patch.sql
```

This would instruct the MySQL client to apply `patch.sql` to the database named `zabbix`. Note that patches are for upgrading **to** the specified version, not upgrading from it.



The Zabbix server must be stopped at this point. Leaving it running during the upgrade is very likely to cause garbage data to be inserted into the database.



This process should complete without a single error. When upgrading past several versions, each patch must complete successfully. If a patch fails on your production system, don't think twice—note down the error, restore your system from the backup, and figure out what happened later. While fixing the database might be possible, that's not something you would want to perform with the system being unavailable.

Usually the patching should happen without any errors, I am just scaring you here. Once the process finishes, and this can take hours on large installations, you are ready to start the Zabbix server back up again.

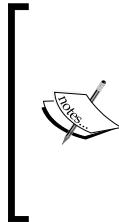
Gathering the data during the upgrade

A normal database patch is much longer than those three lines we looked at before; with 342 lines in total for 1.4 upgrade to 1.6 and 1002 lines for 1.1 upgrade to 1.4 (MySQL version). The database upgrade from 1.6 to 1.8 has 367 lines in total.

As we can see, there are many changes to be made to the database, and this process tends to take a long time, especially when performed on large collected datasets. Sometimes it might be required to keep gathering data even during the Zabbix upgrade, but how could we achieve that if Zabbix server can't be running during the database patching step?

Remember the additional Zabbix process, the proxy? It was able to gather data and buffer it for sending to Zabbix server, so no data was lost even if server was down for some time, which sounds pretty much like our situation. If all your actual monitoring is already done by Zabbix proxies, you are already covered – simply upgrade the server and the data will flow in later.

If you have items that are polled directly by server, you might want to set up a temporary proxy installation, maybe even on the same server, that would be running during the Zabbix server upgrade and removed later. To do this easily, use the mass update functionality in the **Configuration | Hosts** section in the frontend and set the **Monitored by proxy** option. Make sure the proxy can actually gather data by testing it first with a single host.



Setting up a temporary proxy installation will be notably harder if you are using active items. Reconfiguring all Zabbix agents would be required as they retrieve active check lists and send in the data to the first host in the Server configuration parameter. On the other hand, active agents do buffer data for a short while themselves, thus a quick server upgrade might not miss that data as well.

Note that version 1.6 proxies are actually not compatible with a version 1.8 server because of incompatible changes in the communication protocol, thus this method would mostly be usable when upgrading from one patchlevel release to another. As it is not known whether future versions will be compatible, it is suggested to find that out on a release by release basis and through testing as well.

Frontend configuration file

While sometimes upgrading the frontend between minor versions also might happen as easy as with change level upgrades, there's one possible caveat – the frontend configuration file's syntax might have changed. This might or might not be mentioned in the release notes and often the fronted would continue to work, only having some small problems or minor missing features. Thus it is suggested for minor version upgrades to copy the frontend files into place and walk through the frontend installer again. At the end, save the configuration file and compare it to the old one – has anything changed?

Alternatively, compare your existing configuration file to the example file provided in Zabbix frontend directory and look for configuration directive changes.

This way you will get a new configuration file that you will be able to copy over for further change level upgrades.

Compatibility

We mostly talked about Zabbix server upgrading and rightly so, as the server is the most important part. But what happens when you upgrade the server, do you have to upgrade all the agents at the same time? Maybe you can upgrade the agents first?

Again, this depends on upgrade level. **With change level upgrades you can upgrade either server or agent**, and it will work. So you can upgrade some agents now, upgrade server later, and some agents a month later.

With minor Zabbix version upgrades, there's a dual compatibility rule:

- Old versions of agents are supported
- New versions of agents are not supported

Plain and simple, isn't it? The first rule is very helpful when upgrading – we can leave agents in place, only upgrading the server. Agents can be upgraded later when we have time, access, or simply feel like it. It is perfectly fine to use 1.6, 1.4, and even 1.1 version agents with a 1.8 server, as long as you are satisfied with their functionality and capabilities.

But newer versions of agents usually have more useful features like active check buffering and more supported metrics – can we upgrade the agents and leave the server as it is? Not quite. While in some cases a new agent version might work with an older server, that is not supported (as per second rule), and can break monitoring of that device. For example, active check buffering on a new agent would not work with an older server that does not support this feature.

If you absolutely must upgrade some agent to use with an older server, it is up to you to check the release notes for known compatibility issues and to test for unknown ones.

Is that all? What about Zabbix proxies? As proxies aren't that old, there is not much of a track record for them yet. While proxies in one minor version are compatible with all other components, a minor version upgrade could theoretically change that. An upgrade from 1.6; the version where proxies first appeared, to 1.8 did break the compatibility (heartbeat is still processed, but data is not). As proxies, and their protocol, mature, this might change. As usual, check the release notes of new versions for details.

Summary

With new features and bug fixes being added to Zabbix with each new version, it is suggested, if possible, to keep up with them. We just learned the general steps to perform when upgrading to take advantage of both small incremental updates (change versions) and larger improvements, which are folded into minor version numbers for Zabbix.

It should be reminded once more – back up your database before upgrading, especially when performing a minor (or major) version upgrade. Even with all the testing that goes into database patches, that one time when something unexpected happens will pay for all the backups.

We also discussed the compatibility of various components and how that affects upgrading. With change level upgrades it was very easy – all components including the server, proxy, and agent are compatible with each other. Let's try to visualize the minor upgrade level compatibility matrix:

		Newer		
		Older	Agent	Proxy
Agent			Y	Y
Proxy		N		N*
Server		N	N*	

As mentioned before, so far proxy compatibility has not been preserved between minor versions, but that might change in future versions.

15

Taking Care of Zabbix

By now you should be ready to set up a Zabbix system that looks after your servers, applications, switches, various IP devices, and lots of other things. Zabbix will dutifully gather historic information about their well-being and inform responsible persons in case any problem is found. It will send e-mail messages, SMS messages in some more pressing situations, open tracker tickets, and maybe restart a service here or there. But who's going to take care of the poor, little Zabbix? Let's look at what can we do to make Zabbix running happily, and what measures we can use to prevent or solve problems:

- Monitoring the internal health of your Zabbix system
- What other things can we change to improve performance, including some MySQL specific advice and other methods
- Finding and using Zabbix internal audit logs
- Making sure that Zabbix keeps on running and data is not lost by creating backups and even more importantly, restoring from them

Internal items

While Zabbix can monitor nearly anything about external systems, it can be useful to actually know what it takes to do that, and how many things are covered. This is where internal items can help. We already briefly looked at some global data on our Zabbix server – where was that?

In the frontend, open **Reports | Status of Zabbix**. Here we can observe high-level information like whether the Zabbix server is running and values like the number of hosts, items, triggers, and users online. As we might remember, the value next to **Required server performance, new values per second** was important when determining how powerful hardware we would need.

Parameter	Value	Details
Zabbix server is running	Yes	-
Number of hosts (monitored/not monitored/templates)	54	7 / 1 / 46
Number of items (monitored/disabled/not supported)	65	64 / 0 / 1
Number of triggers (enabled/disabled)[true/unknown/false]	17	17 / 0 [1 / 7 / 9]
Number of users (online)	4	1
Required server performance, new values per second	1.0297	-
Updated: 09:30:09		

This report is useful if we want to take a quick look to determine server condition, report on configuration for the management, or brag on IRC, but we can't see here any data like if the amount of new values per second has increased recently, and how that correlate's to the number of active items we have.



Let's remind ourselves what new values per second means. This was a parameter that Zabbix server calculated internally based on item configuration – basically, active item count and item intervals influenced it. Most new values required several selects and several updates to the database, so this value increasing has a serious impact on the performance.

Let's try to see how we could store these values for feel-good graphs later, shall we? Navigate to **Configuration | Hosts**, click on **Items** for **A Test Host**, then click on **Create Item** button. In this form, start by clicking on **Select** next to the **Key** field, then change **Type** dropdown to **Zabbix internal**. This presents us with a nice list of available internal items. While the list provides short descriptions, it might be useful to know when each is useful.

- **zabbix[history]**: Provides a list of values in the `history` table, this metric tells us how many values in total we have gathered for the numeric (float) data type.
- **zabbix[history_str]**: Lists the amount of values in the `history_str` table – the one containing data for character data type. Having access to all the history data can be useful in the long run to determine how particular data type has grown over time.

- **zabbix[items]**: Lists the number of items in the Zabbix database. We can see the same in Zabbix status report, only now we can create graphs showing how the item count has increased (or maybe decreased) during Zabbix's lifespan.
- **zabbix[items_unsupported]**: Lists the number of unsupported items in the Zabbix database—the same list we can see in Zabbix status report. Not only might this be interesting to look at for historical purposes, it also could be useful to have a trigger on this value notably increasing—that would be an indicator of some serious problem.
- **zabbix[log]**: As a counterpart to the log files, this item could be used as a source for triggers to notify on some specific condition the Zabbix server has encountered. Obviously, log file is going to contain better information regarding database problems, as this item would want to insert such information in the very same database.
- **zabbix[queue]**: Shows the total number of items placed in the queue. A notable increase can mean either some problem on the Zabbix server itself, or some connectivity problem to the monitored devices.
- **zabbix[trends]**: Lists the amount of values in the trends table. Long term data storage requirements for numeric metrics can be quite precisely determined, as the size of each record does not fluctuate much from the average.
- **zabbix[triggers]**: The total number of triggers configured. When compared with amount of items, this value can provide an insight into whether this installation is more geared towards historical data collection, or reaction on problems and notifying about those. If the relative amount of triggers compared with items is small, this installation is gathering more data than it is reacting on changes in that data and vice versa.

There are also some other internal items, not mentioned in this list:

- **zabbix[boottime]**: The time when the Zabbix server was started. This value is stored in Unix time format, thus displaying this value as-is would be quite useless. For this key, it is suggested to use the `unixtime` unit, which will convert the timestamp into human-readable form.
- **zabbix[uptime]**: Shows how long the Zabbix server has been running. Again, this is stored in Unix time format, so for meaningful results the `uptime` unit should be used.
- **zabbix[requiredperformance]**: Visible in the Zabbix status report and the one indicative of performance requirements. As it is an important indicator, gathering historic information is highly desirable.

- **zabbix[rcache]**: As Zabbix server caches monitored host and item information, that cache can take up lots of space in larger installation. This item allows you to monitor various parameters of this cache. It can be a good indicator as to when you would have to increase cache size. Consult the Zabbix manual for a list of the available modes for this item.
- **zabbix[wcache]**: This item allows you to monitor how many values are processed by the Zabbix server, and how used the write cache is, which contains item data that is pooled to be written to the database. High cache usage can indicate database performance problems. Again, consult the Zabbix manual for available cache monitoring parameters.

Remember that we created an item to monitor the time when the proxy had last contacted server – that's another Zabbix internal item, although not mentioned in this list.

While with the `boottime` and `uptime` metrics there isn't much choice about how to store them, with other keys we could use different approaches depending on what we want to see.

For example, the `zabbix[trends]` value could be stored as is, thus resulting in a graph, showing the total amount of values. Alternatively, we could store it as delta, displaying the storage requirement change over time. Positive values would denote a need for increased storage, while negative values would mean that the amount of values we have to store is decreasing. There's no hard rule on which is more useful, so choose the method depending on which aspect interests you more.

Let's try to monitor some Zabbix internal item them. In the frontend, open **Configuration | Hosts**, then click on **Items** link next to **A Test Host**, then click **Create Item**. We are choosing this host because it is our Zabbix server and it makes sense to attach such items to it. Fill in these values:

- **Description**: Enter **Zabbix server uptime**
- **Type**: Select **Zabbix internal**
- **Key**: Enter `zabbix[uptime]`
- **Units**: Enter `uptime`
- **Update interval**: Change to **120**
- **Keep history**: Change to **7**

When you are done, click **Save**, then go to **Monitoring | Latest data**. Make sure **A Test Host** is selected in the **Host** dropdown, enter **uptime** in the filter field and click **Filter**. You should be left with a single item in the list, which shows in a human-readable format for how long Zabbix server process has been running.

+ Description	Last check	Last value
- other - (1 Items)		
Zabbix server uptime	09 Mar 2010 09:37:30	14:55:05

This item could be used for problem investigation regarding the Zabbix server itself, or it could be placed in some screen by using **Plain text** resource and setting it to a show single line only.

While useful, this metric does not provide nice graphs in most cases, so let's add another item. Navigate to **Configuration | Hosts**, click on **Items** next to **A Test Host** and click **Create Item**. Enter these values:

- **Description:** Enter **New values per second**
- **Type:** Select **Zabbix internal**
- **Key:** Enter **zabbix[requiredperformance]**
- **Type of information:** Select **Numeric (float)**
- **Update interval:** Change to **120**
- **Keep history:** Change to **7**

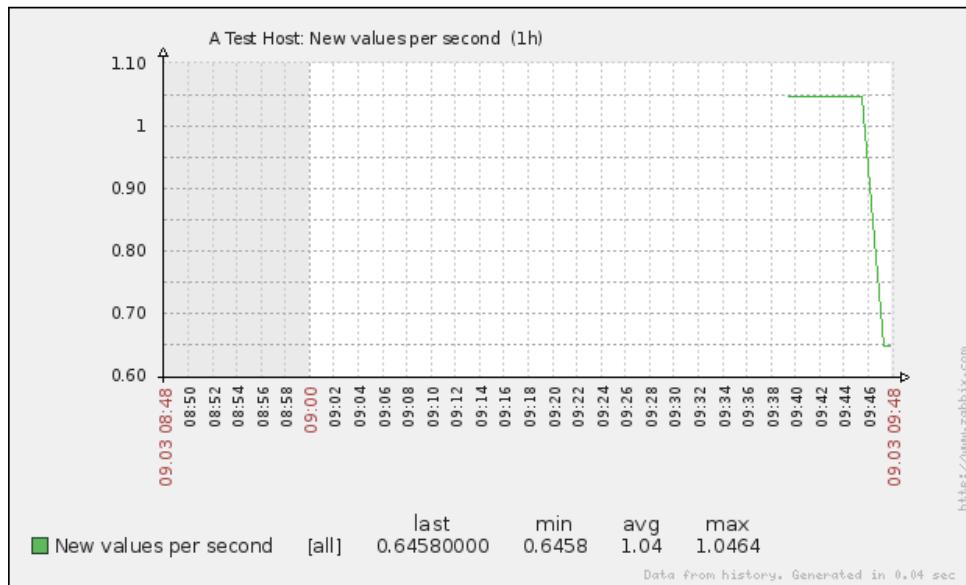
When you are done, click **Save**. Again, review the result in **Monitoring | Latest data**. In the filter field, enter **values** and click **Filter**. You will see an amount of new values per second, which will be almost the same as the one visible in Zabbix status report except that here it will be rounded to two decimal places, and in the report it will have precision of four. Our test server still doesn't have a large number of items to monitor.

+ Description	Last check	Last value
- other - (1 Items)		
New values per second	09 Mar 2010 09:39:34	1.046400

Now let's do something radical. Go to **Configuration | Hosts**, make sure **Linux servers** is selected in the **Group** dropdown, and mark the checkboxes next to **IPMI Host** and **Another Host**, then choose **Disable selected** from the action dropdown at the bottom and click **Go**, then confirm the pop up. That surely has to impact new value per second count – to verify, open **Monitoring | Latest data**. Indeed, we now are receiving less data per second.

+ Description	Last check	Last value	Change
- other - (1 Items)			
New values per second	09 Mar 2010 09:47:31	0.645800	-0.40

That should also be visible in the graph—click on **Graph** in the **History** column. Indeed, the drop is clearly visible.



Such graphs would be interesting to look at over a longer period, showing how a particular Zabbix instance has evolved. Additionally, a sharp increase might indicate misconfiguration and warrant attention—for example, a trigger expression that would catch if the new values per second increased by ten during last two minutes for the item we created could look like this:

```
{A Test Host:zabbix[requiredperformance].last(#1)}-{A Test Host:zabbix[requiredperformance].last(#2)}>10
```

As it could also indicate valid large proper configuration additions, you probably don't want to set severity of such trigger very high.

With this item set up, let's switch those two hosts back on—go to **Configuration | Hosts**, mark checkboxes next to **IPMI Host**, and **Another host**, then choose **Activate selected** in the action dropdown and click **Go**.

Performance considerations

It is good to monitor Zabbix's well-being and what amount of data it is pulling in or receiving, but what do you do when performance requirements increase and the existing backend does not fully satisfy those anymore? The first thing should always be reviewing actual Zabbix configuration.

- **Amount of items monitored.** Is everything that has an item created really useful? Is it ever used in graphs, triggers, or maybe manually reviewed? If not, consider disabling unneeded items. Network devices tend to be over-monitored at first by many users—there's rarely a need to monitor all 50 ports on all your switches with four or six items per port, updated every 10 seconds. Also review items that duplicate relevant information, like free disk space and free disk space percentage—most likely, you don't need both.
- **Item intervals.** Often having a bigger impact than absolute item count, setting item intervals too low can easily bring down a Zabbix database. You probably don't have to check the serial number of a few hundred switches every minute, not even every day. Increase item intervals as much as possible in your environment.

Still, the amount of things to look after tends to grow and there's only that much to gain from tweaking items while still keeping them useful. At that point it is useful to know common performance bottlenecks. Actually, for Zabbix the **main performance bottleneck is database write speed**. This means that we can attack the problem on two fronts—either reducing the amount of query writing, or increasing writing performance.

Reducing the query count

Reducing the query count to the database would improve the situation, and there are a couple of simple approaches we can use.

- **Method one:** Use active items where possible. Remember, active agents gather a list of checks they have to perform and the Zabbix server does not have to keep track of when each check should be performed. This reduces the database load because just scheduling each check on the server requires multiple reads and at least one write to the database, and then the retrieved values have to be inserted into the database. Active item data is also buffered on the agent side, thus making data inserts happen in larger blocks. See *Chapter 3* for reminder of how to set up active agents.
- **Method two:** Use proxies. These we discussed in detail, and proxies have even greater potential to reduce Zabbix server load on the database. Zabbix proxies can check lots of things that Zabbix agents can't, including SNMP, IPMI, and websites, thus lots of work can be offloaded from the server. We discussed proxies in *Chapter 12*.

Increasing write performance

Another helpful area is tuning and configuring database to increase its performance. That is a mix of science and art, which we have no intention to delve deep into here, so we'll just mention some basic things one might look into more detail.

Method one: Tune database buffering. For MySQL with InnoDB engine there are some suggested basic parameters.

- **Buffer pool size:** Parameter `innodb_buffer_pool_size` controls size of in-memory cache InnoDB uses. This is something you will want to set as high as possible without running out of memory. MySQL's documentation suggests around 80% of available memory on dedicated database servers, so on a server that is shared by the database, frontend, and server you might want to set it somewhat below this percentage. Additionally, if you don't have a swap configured, by setting this variable high you are increasing chances that all memory might become exhausted, so it is suggested to add some swap so that at least rarely accessed memory can be swapped out to accommodate database needs.
- **Log file size:** The parameter `innodb_log_file_size` controls the InnoDB log file size. Increasing log files reduces the frequency at which MySQL has to move data from logs to tables. It is suggested that you back up the database before performing this operation. Additionally, increasing this size must be performed offline by following simple steps:
 1. Stop the MySQL server
 2. Move the log files somewhere else. They are named like `ib_logfile0`, `ib_logfile1` and so on.
 3. As root, edit `/etc/my.cnf` and increase `innodb_log_file_size`. There is no specific size you should choose, but setting it to at least 32M might be reasonable.
 4. Start the MySQL server.

Note that there's a caveat to this change – the bigger log files, the longer recovery takes after an unclean shutdown, such as a MySQL crash or hardware problems.

- **Temporary tables.** A typical Zabbix database will require constant use of temporary tables, which are created and removed on the fly. These can be created in memory or on disk, and there are multiple configuration parameters depending on your MySQL version to control temporary table behavior, so consult the MySQL documentation for your version. Slow temporary tables will slow down the whole database considerably, so this can be crucial configuration. For example, attempting to keep the database files on an NFS volume will pretty much kill the database. In addition to MySQL parameters that allow tuning sizes when temporary tables are kept in memory or pushed to disk, there's also one more global parameter—`tmpdir`. Setting this in `/etc/my.cnf` allows you to place temporary on-disk tables in arbitrary locations. In the case of NFS storage, local disks would be a better location. In the case of local storage, a faster disk like a flash-based one would be a better location. In all cases, setting the temporary directory to a `tmpfs` or `ramdisk` will be better than without. This approach also works around MySQL internals and simply pushes temporary tables into RAM.



One major difference between `tmpfs` and `ramdisk` is that `tmpfs` can swap out less used pages, while `ramdisk` will keep all information in memory.

Method two: Splitting the data. There are different methods that would allow you to split data over physical devices where parallel access to them is faster.

- **Separating tables themselves.** By default, InnoDB storage places all tablespace data in large, common files. Setting the MySQL option `innodb_file_per_table` makes it store each table in a separate file. The major gain from this is the ability to place individual tables on separate physical media. Common targets for splitting out in a Zabbix database are the tables that are used most often—`history`, `history_str`, `history_uint`, and `items`. Additionally, `functions`, `items`, `trends`, and `triggers` tables also could be separated.
- **Using built-in database functionality like partitioning.** This is a very specific database configuration topic which should be consulted upon database documentation.
- **Using separate servers for Zabbix components.** While small Zabbix installations easily get away with a single machine hosting server, database, and frontend, that becomes infeasible for large ones. Using a separate host for each component allows you to tailor configuration on each for that specific task.

Method three: Increasing hardware performance. This is the most blunt approach, and also requires financial investment, this can make quite some difference. Key points when considering Zabbix hardware:

- **Lots of RAM.** The more the better. Maybe you can even afford to keep the whole database in RAM with caching.
- **Fast I/O subsystem.** As mentioned, disk throughput is the most common bottleneck. Common strategies to increase throughput include using faster disks, using battery backed disk controllers, and using appropriate disk configurations. What would be appropriate for Zabbix? RAID 10 from as many disks as possible would be preferred because of the increased read, and most importantly write performance and decent availability.

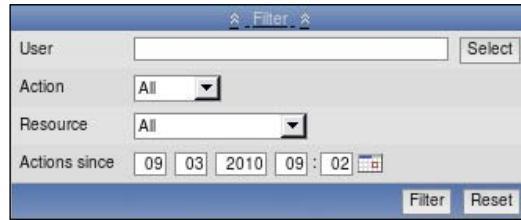
These are just pointers on how one could improve performance. For any serious installation both the hardware and the database should be carefully tuned according to the specific dataflow.

Who did that?

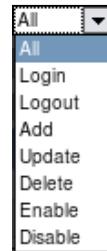
"Now who did that?" – a question occasionally heard in many places, IT workplaces included. Weird configuration changes, unsolicited reboots. Accountability and trace of actions help a lot to determine that the questioner was the one who made the change and then forgot about it. For Zabbix configuration changes, an internal audit log is available. Just like most functionality, it is conveniently accessible from the web frontend. During our configuration quest we have made quite a lot of changes – let's see what footprints we left. Navigate to **Administration | Audit**, set the filter field **Actions since** to be about week ago, and click **Filter**. We are presented with a list of things we did. Every single one of them. In the output table we can see ourselves logging in and out, and adding and removing elements, and more.

Time	User	IP	Resource	Action	Id	Description	Details
09 Mar 2010 09:49:52	Admin	192.168.3.42	Host	Updated	10051	IPMI Host	hosts.status: 1 => 0
09 Mar 2010 09:49:52	Admin	192.168.3.42	Host	Updated	10048	Another Host	hosts.status: 1 => 0
09 Mar 2010 09:46:06	Admin	192.168.3.42	Host	Updated	10048	Another Host	hosts.status: 0 => 1
09 Mar 2010 09:46:06	Admin	192.168.3.42	Host	Updated	10051	IPMI Host	hosts.status: 0 => 1
09 Mar 2010 09:41:34	Admin	192.168.3.42	Host	Updated	10051	IPMI Host	hosts.status: 1 => 0
09 Mar 2010 09:41:31	Admin	192.168.3.42	Host	Updated	10048	Another Host	hosts.status: 1 => 0
09 Mar 2010 09:41:15	Admin	192.168.3.42	Host	Updated	10048	Another Host	hosts.status: 0 => 1
09 Mar 2010 09:41:14	Admin	192.168.3.42	Host	Updated	10047	A Test Host	hosts.status: 1 => 0
09 Mar 2010 09:40:45	Admin	192.168.3.42	Host	Updated	10051	IPMI Host	hosts.status: 0 => 1
09 Mar 2010 09:40:45	Admin	192.168.3.42	Host	Updated	10047	A Test Host	hosts.status: 0 => 1
09 Mar 2010 09:39:26	Admin	192.168.3.42	Item	Added	22291	New values per second	
09 Mar 2010 09:33:40	Admin	192.168.3.42	Item	Added	22290	Zabbix server uptime	

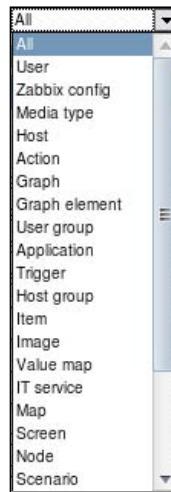
But pay close attention to the filter. While we can see all actions in a consecutive fashion, we can also filter for a specific user, action, or resource.



How fine-grained are those action and resource filters? Expand each and examine the contents. We can see that the action list has actions both for logging in and out, as well as all possible element actions like adding or updating.



The resource filter has more entries. The dropdown can't even show them all simultaneously – we can choose from nearly anything you could imagine while working with Zabbix frontend, starting with hosts and items, and ending with value maps and scripts.



Exercise: Find out at what time you added the **Restart Apache** action.



In the first Zabbix 1.8 releases some actions are not registered in the audit log. Such issues are expected to be fixed in near future.

While in this section, let's remind ourselves of another logging area – the action log that we briefly looked at before. In the upper-right dropdown, choose **Actions**. Here, all actions performed by the Zabbix server are recorded. This includes sending e-mails, executing remote actions, sending SMS messages, and executing custom scripts. This view provides information on what content was sent to whom, whether it was successful, and any error messages, if it wasn't. It is useful for verifying whether Zabbix has or has not sent a particular message, as well as figuring out whether the configured actions are working as expected.

Together, the action and log audit sections provide a good overview of internal Zabbix configuration changes as well as debugging help to determine what actions have been performed.

Real men make no backups

And use RAID 0, right? Still, most do make backups, and for a good reason. It is a lucky person who creates backups daily and never needs one, and it is a very unfortunate person who needs a backup when one has not been created, so we will look at the basic requirements for backing up Zabbix.

Backing up the database

As almost everything Zabbix needs is stored in the database, it is the most crucial part to take care of. While there are several ways to create a MySQL database backup, including file level copying (preferably from an LVM snapshot) and using database replication, most widely used method is creation of database dump using native MySQL utility, `mysqldump`. Using this approach provides several benefits:

- It is trivial to set up
- The resulting backup is more likely to work in a different MySQL version (file system level file copying is not as portable as the SQL statements, created by `mysqldump`)
- It can usually create a backup without disturbing the MySQL server itself

Drawbacks of this method include:

- Heavily loaded servers might take too long to create backup
- It usually requires additional space to create data file in

While it is possible with several backup solutions to create backup from `mysqldump` output directly, without the intermediate file (for example, backup solution Bacula (<http://www.bacula.org>) has support for FIFO, which allows to backup dump stream directly; consult Bacula documentation for more information), it can be tricky to set up and tricky to restore, thus usually creating an intermediate file isn't such a huge drawback, especially as they can be compressed quite well.

To manually create a full backup of the Zabbix database, you could run:

```
$ mysqldump -u zabbix -p zabbix > zabbix_database_backup.db
```

This would create SQL statements to recreate the database in a file named `zabbix_database_backup.db`. This would also be quite ineffective and possibly risky, so let's improve the process. First, there are several suggested flags for `mysqldump`:

- `--add-drop-table`: Will add table dropping statements so we won't have to remove tables manually when restoring.
- `--add-locks`: Will result in faster insert operations when restoring from the backup.
- `--extended-insert`: This will use multi-row insert statements and result in a smaller data file, as well as a faster restore process.
- `--single-transaction`: Uses a single transaction for the whole backup creation process, thus a backup can have a consistent state without locking any tables and delaying the Zabbix server or frontend's processes. As Zabbix uses transactions for database access as well, the backup should always be in a consistent state.
- `--quick`: This option makes `mysqldump` retrieve one row at a time, instead of buffering all of them, thus it speeds up backups of large tables. As Zabbix history tables usually have lots of records in them, this is a suggested flag for Zabbix database backups.

Then, it is suggested to compress the dump file. As it is a plaintext file containing SQL statements, it will have a high compression ratio, which not only requires less disk space, but often actually improves performance—often it is faster to write less data to the disk subsystem by compressing it and then writing smaller amount of data. So the improved command could look like:

```
$ mysqldump zabbix --add-drop-table --add-locks --extended-insert --single-transaction -quick -u zabbix -p | bzip2 > zabbix_database_backup.db.bz2
```

Here we used `bzip2` to compress the data before writing it to the disk. You can choose other compression software like `gzip` or `xz`, or change compression level, depending on what you need more—disk space savings or a less-taxed CPU during the backup. The great thing is, you can run this backup process without stopping the MySQL server (actually, it has to run) and even Zabbix server. They both continue running just like before, and you get a backup of the database as it looked like at the moment when you started the backup.

Now you can let your usual backup software grab this created file and store it on a disk array, tape or some other, more exotic media.

There are also other things you might consider for backing up—Zabbix server, agent and proxy configuration files, web frontend configuration file, and any modifications you might have made to the frontend definitions file, `includes/defines.inc.php`.



[On a large database and powerful server you might want to consider using a different utility—`mk-parallel-dump` from Maatkit project (<http://www.maatkit.org>). It will dump database tables in parallel, quite likely resulting in a faster backup. Note that for a parallel restore you would have to use the companion utility, `mk-parallel-restore`.]

Restoring from backup

Restoring such a backup is trivial as well. We pass the saved statements to the MySQL client, uncompressed them first, if necessary:



[Zabbix server must be stopped during the restore process.]

```
$ bzcat zabbix_database_backup.db.bz2 | mysql zabbix -u zabbix -p
```



[Use `zcat` or `xzcat` as appropriate if you have chosen a different compression utility.]

Of course, backups are useful only if it is possible to restore them. As required by any backup policy, the ability to restore from backups should be tested. This includes restoring the database dump, but it is also suggested to compare the schema of the restored database and the actual one, as well as running a copy of Zabbix server on a test system. Make sure to disallow any network connections by the test server, though, otherwise it might overload the network or send false alerts.

Separating configuration and data backups

While we can dump whole database in a single file, it is not always the best solution—sometimes you might have somehow messed up the configuration beyond repair. With the data left intact, it would be nice to restore configuration tables only, as that would be much faster. To prepare for such situations, we can split tables in two groups—those required for Zabbix configuration and those not required, or configuration and data tables.



While strictly speaking, many configuration tables also contain bits and pieces of runtime data, for backup purposes that is usually not relevant.



We can consider the list of the following tables as data tables, and all other tables—configuration tables.

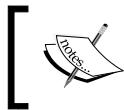
- alerts
- auditlog
- events
- history
- history_log
- history_str
- history_str_sync
- history_sync
- history_text
- history_uint
- history_uint_sync
- node_cksum
- proxy_dhistory
- proxy_history
- service_alarms
- services_times
- trends
- trends_uint

Now we can update our backup command to include a specific set of tables only:

```
$ mysql zabbix -add-drop-table -add-locks -extended-insert --single-transaction -quick -u zabbix -p -tables alerts auditlog events history history_log history_str history_str_sync history_sync history_text history_uint history_uint_sync node_cksum proxy_dhistory proxy_history service_alarms services_times trends trends_uint | bzip2 > zabbix_data_backup.db.bz2
```

This way we are creating a separate backup of data tables in a file named `zabbix_data_backup.db.bz2`.

Exercise—determine the list of other configuration tables.



The data backup will be much larger than the configuration backup on any production Zabbix installation.



Summary

After Zabbix is installed and configured, a moment comes when maintenance tasks become important. In this last chapter we looked at three important tasks:

- Tuning for performance
- Reviewing Zabbix built-in auditing capabilities
- Creating backups

The database performance bottleneck is the one most often reached, and as we found out, we could attack this problem from different angles. We could reduce the write load, distribute it, or increase database performance itself using both software level tuning and hardware improvements.

If you notice a sudden change in Zabbix server behavior like load increase, it can be very helpful to find out what configuration changes have been performed just prior to that. And if there are multiple Zabbix administrators, it is even better, as you can find out who exactly performed a specific change. This is where the built-in auditing capabilities of Zabbix help a lot by providing a change list and also exact change details for many operations.

And reaching one of the most joyful events, a successful backup restore in a case of emergency data loss, we left for the end, where we looked at basic suggestions for Zabbix database backup copy creation and restoration, considering Zabbix's availability during the backup, as well as restore performance.

Of course, both performance and backup suggestions in this chapter are just starting steps, with a goal to help new users. As your database grows and gains specific traits, you will have to apply different methods, but it will be helpful if the design and layout you use from the start will have future scalability and availability taken into account.

A Troubleshooting

Installing and configuring Zabbix can happen without a hiccup for one user and with a constant stream of problems for another. The reasons for the problems can differ from user to user—buggy libraries, bad distribution packaging, unintuitive configuration in Zabbix, or maybe even an occasional bug in Zabbix itself. Here we will look at common problems new users experience when performing various tasks:

- Setting up the initial Zabbix installation
- Working with the web frontend
- Monitoring different devices
- Configuring thresholds and alerting

Installation

There are several common stumbling blocks in the installation process that are sometimes caused by well hidden factors.

Compilation

- Q: I am trying to compile Zabbix on a 64-bit distribution. I have the corresponding development packages installed, but Zabbix claims they are not present.
- A: Double check that development packages are installed for the correct architecture.

Troubleshooting

Q: I am trying to compile Zabbix on CentOS/RedHat. I have curl development packages installed for the correct architecture, but Zabbix complains about them being missing.

A: On some CentOS/RedHat versions such an error can be because one or more of the following development packages are also required:

- e2fsprogs-devel
- krb5-devel
- libgssapi-devel

Q: I am trying to compile Zabbix on SLES, but Zabbix complains about a missing Net-SNMP development library, even though it is installed.

A: There are two possible solutions, depending on SLES version:

- Install the package named tcpd-devel
- Run export CFLAGS=-lssl before configure

Q: I am trying to compile Zabbix on CentOS/RedHat, but Zabbix complains about a missing Net-SNMP development library, even though it is installed.

A: Install the package named openssl-devel.

Q: Compilation fails with:

```
/usr/bin/ld: cannot find -lc
```

A: Install the package named glibc-devel.

Q: I am trying to compile Zabbix from svn, but the configure script fails with an error:

```
syntax error near unexpected token `IKSEMEL,iksemel,'
```

A: Install the pkg-config package and re-run the commands to generate the configure script.

Q: I am trying to compile Zabbix, but it fails.

A: It is useful to reduce the number of possible causes. Verify that you are not compiling with --enable-static that is known to cause compilation problems. If compilation fails without that flag, check config.log file contents in the source directory. It often contains exact error details.

Frontend

- Q: I have installed the Zabbix frontend. What's the default username and password?
- A: The username is `admin`, and the password is `zabbix`.
- Q: I have installed the frontend, but it says at the top of the page "This file is a place-holder.", along with other text.
- A: This is a Fedora/EPEL packages specific problem – remove `conf/zabbix.conf.php` and regenerate it with the installation wizard, or copy `zabbix.conf.php.example` to `zabbix.conf.php` and edit it manually.

Starting services

- Q: I am attempting to start the Zabbix server or agent daemon, but it fails, mentioning semaphores. What's wrong?
- A: This usually happens if you have attempted to start an older version of Zabbix daemon previously. Rebooting the server helps, but is often too drastic – you can list semaphores and shared memory segments with command `ipcs`. With all Zabbix services stopped, you can remove Zabbix-related structures with `ipcrm`. Consult the documentation for these commands for more details.

Frontend

- Q: Zabbix is working correctly, but some/all graphs are not displayed.
- A: See the Apache error log for more detail. Usually this is caused by the PHP script memory limit being too low – if that is the case, increase it by setting the parameter `memory_limit` to a higher value. Another possible cause is a broken `conf/zabbix.conf.php` file – verify that it does not have any weird characters, especially at the end of the file.
- Q: Complex views, like screens with many elements, sometimes fail to load. What could be causing this?
- A: Similar to the previous problem, check that the PHP memory limit has not been exceeded. Additionally, check the PHP script timeout (`max_execution_time` parameter) and increase if necessary.

- Q: My graphs have gaps.
- A: It's not only graphs—data is missing in the database as well. This problem should be resolved by finding out what causes data loss. Common reasons for this are:
- Network problems—if the network is unreliable, data will be missing.
 - An overloaded monitored device—for example, if you have added a switch with many ports and are monitoring several metrics on each port very often, try increasing the intervals and disabling unneeded items.
 - An overloaded Zabbix server, usually the database. Check the system load on the Zabbix database server, especially `iowait`. Consider practices discussed in Chapter 15, *Taking Care of Zabbix*, to improve performance.
- Q: I had Zabbix installed and running, but suddenly it is showing me the installation screen again.
- A: Check the accessibility of `conf/zabbix.conf.php` file. Usually you can return to the Zabbix interface by clicking **Cancel** button in the installation wizard.
- Q: I am trying to use a large page with many elements, but refresh kicks in before the page even finishes loading. How can I solve this?
- A: Increase the refresh period in your user profile. While the page loading speed won't be improved by that, at least page will get a chance to load completely.
- Q: The clock on my server is correct, but the frontend shows incorrect times.
- A: Check that the time zone is set correctly in the PHP configuration.
- Q: Zabbix server is running, but the frontend claims it is not.
- A: This could be caused by multiple factors.
- Check `conf/zabbix.conf.php` file—the frontend uses the server address and port specified there to query the Zabbix server process
 - Check your PHP configuration—enabling `safe_mode` is known to cause such a problem
- Q: I am having a problem with the frontend that is not listed here.
- A: Check the Apache error log and PHP log—these often offer an insight into the cause.

Locked out of the frontend

A common mistake, performed by both new and seasoned users—is locking oneself out of the frontend. This can be achieved in several ways, but we're more interested here in how to get back in.

- Q: I forgot my password and tried to log in until Zabbix frontend stopped responding.
- A: By default, Zabbix denies access for 30 seconds after five failed login attempts, so just wait for 30 seconds. You can customize these values in `includes/defines.inc.php`:
- `ZBX_LOGIN_ATTEMPTS`—after how many unsuccessful attempts Zabbix denies access
 - `ZBX_LOGIN_BLOCK`—for how long to deny access, in seconds

- Q: I have forgot my `admin` user password, or I have been tasked with managing a Zabbix installation where the `admin` user's password is not known.

- A: You can easily reset admin user password by directly modifying the database:

```
mysql> update zabbix.users set passwd=MD5('somepassword') where alias='Admin';
```

Of course, replace `somepassword` with some other string. Keep in mind that MySQL by default saves console commands in `~/.mysql_history` file, so you might want to set password to some temporary version and update it in the frontend later.

- Q: I changed the authentication method, but it didn't work as planned and now I can't log in anymore.

- A: You can restore Zabbix's internal authentication method by editing the database:

```
mysql> update zabbix.config set authentication_type='0' where configid='1';
```

Authentication type 0 is the internal one. For the record, other types are 1 (LDAP) and 2 (HTTP).

Problems with monitoring

Sometimes monitoring something proceeds without a hitch, sometimes it just won't work.

General monitoring

- Q: I added a host or item, but I don't see it in **Monitoring | Latest data**.
- A: It most likely has not received any data. Hosts without any data do not appear in the monitoring section.
- Q: I can see my host in **Latest data**, and new values are coming in – but it is missing in **Monitoring | Overview**.
- A: **Overview** is probably set to display triggers – verify that the host has triggers configured. Hosts without triggers are not displayed in the trigger mode.

Monitoring with Zabbix agent

- Q: I am trying to monitor a host using Zabbix agent checks, but it doesn't work.
- A: Common reasons why Zabbix agent items won't work include:
- The Zabbix agent daemon is not running. Trivial right? Still, start by checking that it is actually running.
 - The Zabbix daemon is not listening on the correct port or interface. You can check what port and interface Zabbix agent daemon is listening on by running `netstat -ntpl` on the monitored host. The default agent daemon port is 10050.
 - The Server IP address in the agent daemon configuration file is incorrect. Check the configuration file and make sure the `Server` directive specifies the IP that the Zabbix server will be connecting from.
 - Network problems prevent the server from connecting to the agent daemon properly. This includes things like local and network firewalls blocking connections, but also some network devices and setups actually masking the source IP address of the Zabbix server's outgoing connections. Test connectivity by executing `telnet <monitored host IP> 10050` from the Zabbix server. Use another port, if you have customized that. If the connection is not opened, debug it as a network problem. If the connection is immediately closed, the Zabbix agent daemon does not see the connection as coming from the IP address set in the configuration file. Note that in some cases you might have to actually use the IPv6 address, as the Zabbix agent is receiving that as the one of the incoming connection.

Q: I am trying to monitor a host using Zabbix agent active checks, but it does not work.

A: Active items are a bit more tricky. Things to verify:

- Check network connectivity as with normal items – from the monitored machine, execute `telnet <Zabbix server IP> 10051`. Use another port if you have customized that.
- Make sure that the time specified in `refreshActiveChecks` option in the agent daemon configuration file has passed before expecting results from the active items. If you want to force agent to reload list of items from the server, restart the agent. Make sure to wait for Zabbix server to refresh its item cache, otherwise agent won't be getting configuration changes.
- Check whether the host name specified in agent daemon configuration file in the `Hostname` option matches the one configured for the host in the frontend. Note that this is not the IP address or DNS name, only the host name will work.
- Make sure that Zabbix server you want to retrieve/send active checks is listed as the first entry for `Server` option in agent daemon configuration file.

Q: I am verifying that I can get the value on the monitored host, but Zabbix agent says it is not supported or gives me wrong data.

A: There are several possible cases:

- You are checking things like the process count or using `zabbix_agentd -t` syntax as root, but various permission limitations, including grsecurity and SELinux, can prevent access for Zabbix agent. This includes the Zabbix agent showing the amount of running particular processes as 0 even when with root access you can see actual amount.
- Another case when local process count differs from what the Zabbix agent returns – various interpreted processes like Python or Perl ones can appear to the agent as interpreter processes, only with user processes as a parameter. Processes known to suffer from this problem include `amavisd` and `xend`. In those situations you can use a different approach, example item key – `proc.num[python,,,xend]`. This will look for `python` processes, having the `xend` string in their parameters.
- The monitored instance is missing. For example, if you are asking for metric with key `net.if.in[eth0,bytes]` and Zabbix agent claims it is not supported, verify that interface `eth0` actually exists. In some cases, Linux can move an existing interface representation to `eth1` or another device.

User parameters

Q: My user parameter does not work.

A: Common causes that break user parameters:

- A missing environment is one of the biggest stumbling blocks when setting up user parameters. Zabbix agent does not preserve environment details like the `HOME` variable or other information. This can break various configurations that are set for the Zabbix user on the monitored host. Make sure to set the environment as required either by setting variables in the user parameter directly, or in a wrapper script.
- Again, restricted permissions for Zabbix user will be confusing to debug if you will run commands for testing as root, so always test user parameters as Zabbix user. If you need root access for some check, configure access via sudo.
- Returning unclean data also can easily break the process. When retrieving data with user parameters, make sure it does not contain characters, making it unsuited for storage (like returning `26.6 C` for a float data type item), or having other weird characters (like having CR/LF linefeed at the end of the data string).

Problems with SNMP devices

Q: My SNMP items do not work.

A: Double-check that the SNMP version and community string are set correctly. Specifying incorrect SNMP version will often cause timeouts, making it harder to debug. Of course, check general connectivity and permissions by using the `snmpwalk` and `snmpget` commands from the Zabbix server. Additionally, make sure you are not overloading the monitored device by querying lots of values too frequently.

Q: My SNMP items work, but some OIDs on a specific device do not, even though data appears in `snmpwalk` output.

A: Try `snmpget` with those OIDs. Some UPSes are known to have buggy firmware that prevents these metrics from working with `SNMPGET` requests, but do work with the `SNMPGETNEXT` requests that `snmpwalk` uses. If that is the case, upgrade the firmware on the device.

Q: I listed all SNMP MIBs I want to use in `/etc/snmp/snmp.conf`, but Net-SNMP utilities do not use them all properly.

A: Some Net-SNMP versions silently trim lines in this file to 1024 symbols, including the newline character. Try splitting options on multiple lines so that a single line does not exceed 1023 printable characters.

Q: I can retrieve data with `snmpget`, but Zabbix times out.

A: If you are trying to use a fully qualified SNMP OID with Net-SNMP 5.3.0 or 5.3.0.1, a bug in Net-SNMP will prevent it from working. Either patch Net-SNMP, or use numeric or short OIDs.

Problems with IPMI monitoring

Q: I can't get the IPMI item to work.

A: There are several things to verify when IPMI items do not work:

- Make sure that the Zabbix server is configured with IPMI support. Trivial, but easy to miss.
- Check whether the `StartIPMIPollers` option in the server's configuration file is set to the default value, 0. If so, set it to 1 and restart Zabbix server.
- Make sure that the sensor names are correct. You can get the sensor names with IPMItool, and you have to use the name as it appears in the IPMItool output with spaces, and without quoting it.
- Discrete IPMI items are not supported.
- There is a bug in older OpenIPMI versions. Retrieving data with IPMItool will work fine, but Zabbix will be unable to get any values. OpenIPMI 2.0.7 is known to have this problem, while OpenIPMI 2.0.14 is known to work.

Problems with ICMP checks

Q: All of my ICMP checks are failing.

A: If you are using the `SourceIP` setting in the Zabbix server configuration file, it requires `fping` with `-s` flag support. This support is not available in official `fping` packages, so you will have to either use a patched one (many distributions provide already patched `fping` packages), or remove the `SourceIP` option.

General issues

Q: I am monitoring an item of type counter, but it doesn't work and Zabbix complains that the received data is not suitable for value type.

A: Note that the received data might be transformed; this is especially common with counter items as they usually are stored as delta (speed per second). In this case, the resulting value is fractional, and the item data type should be set to **Numeric (float)**.

Triggers

- Q: My trigger does not work, or Zabbix refuses to add my trigger.
- A: Check the trigger's syntax, paying close attention to parenthesis—is the correct type used, are they all properly closed? The same goes for quotes. Do not use spaces in a trigger expression, and try splitting up complex expressions to pinpoint the error. Also keep in mind that you can't omit the leading zero in decimals—writing "0.9" as ".9" won't work.

Actions

- Q: My actions do not work.
- A: Make sure the user you want to send notifications to has read permission to at least one of the hosts that participated in generating the event. Since version 1.6.3, Zabbix does not perform actions for users who do not have sufficient permissions.
- Q: My e-mail notifications are not sent, and I can see error messages like [127.0.0.1] did not issue MAIL/EXPN/VRFY/ETRN during connection to MTA in e-mail server log files.
- A: These messages are most likely caused by Zabbix monitoring the SMTP service, not by notification attempts. Check the permissions as mentioned above and check the action log in **Administration | Audit** to find out why notifications are failing.
- Q: I misconfigured an action by enabling escalations or recovery message without adding a condition for trigger to be in a problem state (explained in *Chapter 6, Acting upon monitored conditions*), and now I am receiving notifications constantly, even after adding the correct condition. Can I solve that?
- A: There is one harsh method to stop these escalations, which would never stop on their own—you can delete all the currently active escalations. Beware, this will also remove correct escalations. In the correct database, execute:

```
mysql> delete * from escalations;
```

Since Zabbix version 1.8.2, unwanted escalations can be resolved by adding the proper action condition and making the affected triggers change their state.



Being Part of the Community

Zabbix isn't just another product, it's an open source project. As such it has an easy way to closely follow the development, and to get community support, and it can benefit from new contributors. But each open source project is different in how it is run, and what guidelines it has, so let's look at what one can expect to find from this aspect of Zabbix:

- Community support can be a great way to solve some problem, either by discussing it on the official forum, looking at the wiki, chatting on the IRC channel, or using the open bug tracker
- Following the development more closely by getting latest source code can allow one to try out fixes for problems as soon as possible, provide early feedback, and get more familiar with internals of Zabbix
- For specific development, support contracts or other services commercial support might be handy

Community and support

There's a vibrant community of Zabbix users who communicate and share using different means. You are free to choose the communication and information exchange method you prefer, but it is good to know how things are arranged.

You are welcome to ask new questions and help others by answering theirs, but it is suggested to obey some basic rules, which will help you to get answers:

- Be polite. Remember that nobody is required to respond to you on the forum, IRC, or elsewhere.
- If you get no response, maybe nobody knows the answer right now, be patient. Remember that people live in different time zones, so what is the middle of the working day for you might be the middle of the night for somebody else.

- Use English, unless communicating in a dedicated native language section. Avoid the use of single letter substitutions for words. Keep in mind that for many participants English is a second or third language, so pointing out mistakes should be polite. Perception of language also varies a lot – what is considered offensive in one region might be completely fine in another.
- When asking for help, provide as much relevant information as possible. This usually includes your Zabbix version and other things, depending on the problem you are having, including the database used and a detailed problem description. It is very helpful to note what steps you have already taken while trying to resolve the problem. Don't make others guess details as if they have to ask for more information, it will delay the solution.

Using the Zabbix forum

The Zabbix forum is located at <http://www.zabbix.com/forum>. You can read it without registering, but for posting you will need authenticated access, so register for a user account. Historically, the forum has always been the busiest place, so it offers both a large collection of already solved problems and a high chance that you will receive assistance with new problems.

While we looked at the general suggestions for efficient and satisfactory communication, there are some forum specific suggestions as well.

- Choose the appropriate forum for your question. If your problem is with the development version of Zabbix, it should not be asked in the forum concerning the Zabbix website.
- Choose wisely when to create a new thread and when to comment on an existing one. It is highly discouraged to ask different question on an existing thread. On the other hand, it's better to search the forum before creating a duplicate thread about an existing problem.
- Enable new message notifications so that you can respond in a timely fashion if additional information is requested. That will help to resolve the problem sooner.

Editing the wiki

In addition to forum, Zabbix also has a wiki. Available at <http://www.zabbix.com/wiki>, it has a collection of user-contributed content, including instructions for installation on specific platforms, Zabbix templates to monitor various devices and applications, as well as various bits and pieces of documentation, contribution related advice, and other information.

Being a wiki, everybody is welcome to participate in adding to and improving the content. It is worth remembering that as you are able to edit and improve all content, everybody else will be able to improve articles you are adding. If you are not confident enough to edit a page, a comment added to that page would be appreciated.



Use your Zabbix forum username and password to get write access to the wiki, as well as the ability to add comments.



There are community updates posted on the wiki – those being news on things that mostly interest people more involved with Zabbix. In addition to various events, a regular news item is the monthly development update, which lays out things which happened in the Zabbix development version during the previous month.

Chatting on IRC

IRC, or Internet Relay Chat, is a fairly old communication method which is especially popular within open source projects communities. Zabbix users also like to gather for Zabbix related discussions on a separate channel. Located on the Freenode network, the channel **#zabbix** is where you can expect to get help and communicate with fellow Zabbix users. You may connect to any Freenode IRC server with a dedicated program called an IRC client (or use one of the many web-IRC gateways like <http://webchat.freenode.net/>), then join this channel. There are many different options available for different operating systems, and you are free to choose any one, it won't impact your ability to communicate with people using a different one. In addition to general communication guidelines, there are some IRC specific ones as well:

- Reiterating the basic suggestion – be patient. Too often people come in, ask their question, and leave a few minutes later. Other members of the channel might be sleeping, eating, or otherwise away from their computer. So ask your question and stay around for a while. If it happens to be a weekend, a while might even be several days.
- Do not ask to ask. If your question is about Zabbix, and it is well thought out then just go ahead and ask. Starting with "Hey, can I ask a question about Zabbix?" will require somebody to confirm that "Yes, you can", then you typing the question, and only then can the helping process start. In the end it can take much longer.

- Do not repeat your question too often, it will only annoy others. While it might be tempting to ask again and again when new people join, they are unlikely to be the experts you are waiting for, so again, be patient. On the other hand, it usually is fine to repeat the question if no answer has appeared for a longer time—a day, for example.
- Do not type the names of people present, hoping that would net you help. That will needlessly distract them. Wait for somebody to respond instead.

Filing issues on the tracker

What if you have discovered a bug, or you have a genial, bright idea for how to improve Zabbix? Zabbix uses an issue tracker to record such things and track the resolution process.

To access the Zabbix issue tracker, navigate to <https://support.zabbix.com>. Here you can register and log in to search existing reports, as well as enter new ones.



Note the `https://` protocol—it is mandatory.



When reporting a new issue, choose the correct project—project ZBX is used for bug reporting, and project ZBXNEXT for new feature requests. It is strongly suggested to search the tracker before filing a new report—maybe the problem has already been reported, and there is no need to create duplicate reports.

What if you have resolved the problem yourself and have a patch fixing some bug or implementing a feature? Just attach it to the corresponding report. You should discuss your approach with Zabbix developers before getting to coding for all but trivial cases—maybe they are already working on it, or maybe your approach will conflict with some other feature in development.

Following the development

So, you have seen some interesting new feature in the monthly development update, and you want to try it out. Maybe you want to check exactly how a particular change was implemented. Or maybe you would like to produce a patch which depends on some changes being made in the development version.

Zabbix stores all its source code in a code repository, namely a Subversion one. As an open source project, it also provides access to this repository to everybody, so it is possible to take a look at how a development version is shaping up. In general, there are many possibilities—not unique to Zabbix, but possible because of the open source approach that open this way. Some might be:

- Trying out new a development version out of curiosity
- Testing the development version to provide feedback sooner in the development cycle
- Using the development version to create patches
- Providing feedback and improving documentation of new features

Providing early feedback is also an influential method to impact future of Zabbix—if a particular feature is important to you, testing it early and letting developers know about any problems you find with it and what improvements are possible is more likely to result in those changes happening.

The first thing we have to solve is actually getting the source.

Getting the source

When looking for the Zabbix development version, there are two ways to get it, each with its strengths and weaknesses.

Daily snapshots

On the Zabbix development download page, <http://www.zabbix.com/developers.php>, there are daily snapshots of development versions provided. These usually have the same setup procedures as the released versions. Benefits of using daily snapshots include:

- Getting them is a simple download
- The package is already generated for you

drawbacks include:

- No way to update only those parts of the development version that have actually changed
- No way to see what actually has changed
- No way to get some arbitrary older version

It is suggested to use daily snapshots if you want a simple, one time peek at how Zabbix's development is progressing.

Accessing the version control system

If you plan to follow Zabbix development for a longer time period, or you want to see how exactly a particular change was implemented, daily snapshots would soon become cumbersome to use. Luckily, we can directly access the system Zabbix developers themselves use, a popular version control system Subversion or SVN.

To access SVN repositories, some specific software, a client, is needed. There are many different SVN clients for various platforms, and you can choose whichever seems more convenient to you. Here we will use the official command line client. As this client is available in almost all Linux distributions, you may want to use it on our Zabbix test server. But before we start playing with it, we must know that the Zabbix source code repository resides at `svn://svn.zabbix.com`. In SVN, development is usually split into a *trunk* and *branches*. While the trunk represents the most recent development work, branches usually are used for stable version maintenance. Zabbix uses the same schema, and there are branches for stable version maintenance like 1.6 while 1.8, and big changes happen in the development section, trunk.

So let's say we are interested in latest new features and want to retrieve trunk. To do this, create some directory to hold the data, called checkout. Enter it and run:

```
$ svn checkout svn://svn.zabbix.com/trunk
```

This will proceed to retrieve all the files in the trunk. As of this writing, Zabbix trunk checkout uses approximately 114 MB on disk, but the amount transferred over the network will be smaller than that. Once the process completes, you might be tempted to proceed but that won't be easy to do, as there is no `configure` script at all. According to Zabbix scripts, we should now execute some commands to create this script:

```
$ aclocal -I m4  
$ autoconf  
$ autoheader  
$ automake -a  
$ automake
```



Since February, 2010, you can simply launch script named `bootstrap.sh` instead of the commands above.

After these are completed, we should have the configuration script. Now we can compile this development version of Zabbix, right? Not quite yet. Development repositories hold only generic database schema, so we would not be able to create the database. We will have to generate the actual schema files ourselves. It is also suggested to create a package, one just like those downloadable from Zabbix site,

so let's do that. Before we can generate the database schema and package, we have to use `configure` script, though. But we can make it slightly faster and requiring less dependencies by omitting any features that are not required. This also allows the creation of a Zabbix package on another machine that would not have all dependencies for the required functionality like SNMP or IPMI monitoring installed. In the simplest approach, run:

```
$ ./configure
```

This will produce the files required for database schema and package generation. Now we can proceed with schema generation:

```
$ make dbschema
```

With database schema files generated, we are ready to create a package:

```
$ make dist
```

After this command completes, the source directory should have a new package named `zabbix-<version>.tar.gz`. Here, the version will be whatever name the development part has received. As we discussed before, odd numbers are used for development versions, so it could be named `1.9`, for example. From now on we are back to the known path, as this package pretty much the same you can download from released version area or from the daily snapshots area.

But that was a lot of work to get the same thing as we could have downloaded right away – why do it at all? Indeed, if you only want to grab development version once, daily snapshots should be your choice. But SVN checkout presents other benefits. Let's see what they are.

Looking at the changesets

A collection of changes to repository is called **changeset**. A changeset that has been placed in a repository is said to be committed. We can list changesets being committed. For example, if we would like to know what the last two changesets that were committed to this part of the repository, we would issue:

```
$ svn log -r PREV:HEAD
```

The Subversion switch `-r` allows us to specify revisions – numeric representations of each change. `PREV` and `HEAD` are special references, being the previous change and latest version respectively. Sometimes we might be instructed to test or use a specific version, called revision. In that case, it is possible to retrieve it by issuing:

```
$ svn up -r 1234
```

Replace 1234 with the revision number you are told to use. This will update the whole checkout to that revision, and you should now run again the commands discussed above, repeating the same process as after just having checked out. But sometimes we might need to update only one or a few files to a specific revision – that can be done by specifying path, for example:

```
$ svn up -r 1234 frontends/php/discovery.php
```

You can specify both directories and files and get different revisions to test behavior changes or find the specific change that introduced a problem for you.

So you have tried out a development version, maybe several revisions. Some time later, you decide to find out what changes have been made to the trunk. First, the current revision should be figured out, while in the checkout directory, run the following command:

```
$ svn info
```

Look for the line that looks like this:

```
Revision: 8000
```

With that number on hand, it is now time to update local copy to the latest and greatest. From the local copy directory, run:

```
$ svn up
```

This will proceed to update everything that has changed, compared to whatever copy you have. As only changes are pulled, this will result in much less data downloaded, compared to downloading daily snapshots over and over again. Now you can just proceed with building Zabbix as discussed before, or you can choose to view what exact changes developers have committed:

```
$ svn log -r 11000:HEAD
```

This command will display the exact changes pushed to the code repository, along with any comments that the developers decided to add. This can be used to determine what exactly was changed. But all this about forward looking development version, trunk – what if you want to see a particular bug fix for some problem in the stable version, applied to that particular branch? Just as we grabbed the trunk from the code repository, we can also grab the branch. Create a separate directory to hold this checkout, and issue:

```
$ svn checkout svn://svn.zabbix.com/branches/1.8
```

As we can see, instead of the trunk we are now specifying the subsection branches. After that comes the specific branch, which can be any valid branch.



Available branches can be listed with `svn ls svn://svn.zabbix.com/branches`.



While installing a branch version is pretty much the same as installing the trunk, there's one more use case with branches. If a particular bug is fixed in the branch and you want to benefit from that before the next stable version is out, it is possible to apply this single change to the installed copy. To do that, though, the change first has to be extracted in a format that is easy to reuse. Here, another command comes to help. Remember `svn log` we used to look at the changesets before? It showed the revision number for each changeset. If we now have this number, we can take a look at what files a particular commit modified.

```
$ svn log -v -c 8001
```

Here we use the switches `-c` to specify single changeset and `-v` to increase verbosity level. In the `Changed paths` section one or more files will be listed, for example:

```
M /trunk/ChangeLog
M /trunk/src/zabbix_server/operations.c
```

When creating a patch, we might want to omit files that do not affect actual software behavior, `ChangeLog` in this case. Creating a patch thus would be done like this:

```
$ svn diff -c 8001 src/zabbix_server/operations.c > /tmp/zabbix.patch
```

Notice how we used Subversion's `diff` subcommand, specified a single file, and redirected output to a file. Now the patch should be applied to our Zabbix installation. To do this, transfer the patch file to Zabbix source installation directory and execute:

```
$ patch -p 0 -i zabbix.patch
```

The `patch` utility is instructed use input file `zabbix.patch` and use full path information as specified to apply the changes. After patching we should evaluate areas the patch applies to—if it's the server, we should recompile and reinstall our server binary, the same with the agent daemon. If changes were performed on the frontend only, we'll usually want to apply the patch to the installed frontend directly, by placing the patch in the frontend directory and applying it as root with:

```
# patch -p 2 -i zabbix.patch.
```

Note that in this case we are instructing the `patch` utility to strip the first two directories from the path. When we are patching the frontend, no recompilation is necessary, and all changes will be visible immediately. What if we applied a patch, but it only made things worse? Thankfully, that is easy to undo by applying the same patch in reverse:

```
# patch -R -p 2 -i zabbix.patch
```

If using the above command line for the frontend, again, no further action is required. If it affects binaries, we would have to recompile them.



See the SVN documentation for more detailed instructions, or ask in the Zabbix IRC channel for Zabbix specific Subversion repository questions.



Commercial support options

Community support is great. It is often speedy, correct, and friendly. However, even if it is like that always, there might be cases when you might need some more formal approach. Common cases when a formal agreement is pursued include:

- A company policy requires a support agreement for all systems put in production
- You want qualified help when implementing Zabbix
- The Zabbix installation will be managed by people who are not deeply involved and don't have great expertise in IT
- You need some feature developed or improved

Approaching commercial support is often the best solution in such cases, and it is possible to obtain such a support from the company behind Zabbix software. Visit Zabbix website, <http://www.zabbix.com/support.php> to obtain more information.

In some cases company procurement requirements might mandate that you use a local company for your support agreement – partner and reseller list at <http://www.zabbix.com/partners.php> should help to choose one that is geographically close.

Summary

Finalizing our trip to a successful and long lasting Zabbix deployment we looked at available options when you might need a helping hand, learned how to check out the latest Zabbix developments, and talked about what to do if you need some contractual help. For Zabbix, available communication channels include IRC, a forum, and a wiki, where community support is concentrated.

Your bug reports are awaited on the issue tracker – hopefully you will have to use it very rarely. Well thought out feature suggestions are also appreciated on the same tracker.

While the community support of Zabbix is of a high quality, it won't offer guaranteed help and new development. For company policy compliance, specific features or just peace of mind when pushing for Zabbix in production – there's the possibility of commercial support or development contracts, fulfilled by the same company that developed Zabbix.

The Zabbix community welcomes you to the club, of those who know what happened in their environment yesterday, see how it is faring today, and have insight on what will happen tomorrow.

Index

Symbols

--add-drop-table flag 375
--add-locks flag 375
--extended-insert flag 375
--quick flag 375
--single-transaction flag 375
-d flag 339
<host> block 349
<status> block 349
<templates> block 349

A

actions
additional conditions 182
and dependencies 182
conditions, limiting 180-182
conditions 62
configuring 62, 63
issue management systems, integrating with 196
main configuration 62
notifications, macros used 185-187
notifications, sending out 184
operations 62
per media limits 183
remote commands 199-201
scripts, using as media 197, 198
things, escalating 187-196
active items, Zabbix agent
creating 82-92
DisablePassive parameter 85
main filter 90
subfilter 91
aggregate items

about 279
configuration 280-282
data querying, not supported by Zabbix agent 288, 289
detail monitoring levels 291, 292
environment trap 293-295
external checks 282-287
flexible user parameters 289-291
func parameter 279, 280
functions 280
group parameter 279
grpfunc 280
grpmax function 282
grpmin function 282
grpsum function 282
key parameter 279
param parameter 280
user parameters 287
working 287
wrapper scripts 296
authentication methods, user
HTTP 149
Internal 149
LDAP 149

B

backing up, Zabbix
backup, restoring 376, 377
configuration, separating 377, 378
database backup 374, 375
requirements 374-377
bar reports
about 265, 266
multiple item values, distributing 269-272

multiple period values, comparing 273-276
multiple period values, distributing 266-269

breadcrumbs 48

bzip2 376

C

categories, item configuration

- availability 67
- efficiency 68
- monitoring 67
- performance 67
- security 67
- system management 68

change level upgrading, Zabbix

- upgrading**
- frontend files, replacing 356
- indexes, adding 355, 356

changeset 397, 400

checks, performing

- ICMP checks, performing 95-97
- steps 93, 94

clock field 342

community

- activity 391
- forum discussion 392
- Internet Relay Chat 393
- issues, tracking 394
- rules 391, 392
- wiki 393

community support

- cases 400
- features 400

compatibility

- rules 361, 362

compound elements

- about 250
- dynamic screens 253-255
- screens 250-252
- slide shows 256

ConfigFrequency option 333

configuration, XML export/import used

- initial configuration, using 348
- modified configuration, modifying 351, 352
- modifying 348, 349
- simple shell scripts, using 350

XML roundtrip, using 350

custom graphs

- custom y-axis scale 231-233
- pie graphs 235-237
- sort order 229-231
- stacked graphs 234
- trigger line 224-227
- two y-axis 227-229
- working time 224-227

D

data, splitting

- built-in database functionality, using 371
- separate servers, using 371
- tables separation 371

database

- existing data, modifying 346
- host, converting to template 345, 346
- users, managing 342

database buffering

- buffer pool size 370
- log file size 370
- temporary tables 371

database patch 354

data displaying, challenges

- about 257
- central display 258
- information overload 258
- non-interactive display 257
- recent changes, flashing 259

data gathering, Zabbix

- action 62
- custom agents, using 300
- data, sending in 297, 298, 299
- e-mail parameters, configuring 60-62
- graphs 54
- host, creating 50, 51
- item 48
- item, configuring 49, 50
- item, creating 52, 53
- methods 297, 300
- trigger, creating 58, 59
- triggers 48

DataSenderFrequency option 333

data visualization
about 221
compound elements 250
need for 221
single elements 222

Dell Remote Access Controller. *See DRAC development version*

changeset 397
source, daily snapshots 395
source, obtaining 395
version control system, accessing 396-398

devices, monitoring
general issues 389
general monitoring 386
ICMP checks problems 389
IPMI monitoring problems 389
SNMP device problems 388
user parameters, not working 388
with Zabbix agents 386, 387

DRAC 136

E

existing data, modifying
change, finding out 347, 348
change, performing 347, 348
information flow, determining 347
problem, examining 346

external checks 282-286

F

fping 95

frontend, Zabbix
administration category 47
Auto-login field 48
Auto-logout field 48
breadcrumbs 48
configuration category 47
exploring 45-48
inventory category 47
monitoring category 47
reports category 47

G

general policy
version upgrades 354

Zabbix versions 353

graphs, single elements
custom graphs 223
defining 222
simple graphs 222, 223

graphs, Zabbix
about 54, 55, 56
clicking 57
data entries 56
dragging 57
Latest data screen 54
scrollbar 56
Zoom option 55

gzip 376

H

HeartbeatFrequency option 333

host

about 50, 143
groups 143
monitoring, proxy used 327-329

host groups
about 143
creating 145-147
host list 148
Linux servers 144
SNMP Devices 144
snmptraps host 148
state, changing 147
templates 144
Windows servers 144
Zabbix Servers 144

Hostname option 387

hysteresis 179

I

id field 342

information flow, Zabbix source
host 64
item 64
operations 64
trigger 64

innodb_buffer_pool_size parameter 370

innodb_file_per_table option 371

innodb_log_file_size parameter 370

installation troubleshooting
compilation 381, 382
frontend 383
services, initiating 383

Intelligent Platform Management Interface. *See IPMI*

internal items
about 364
monitoring 366-368
`zabbix[boottime]` 365
`zabbix[history]` 364
`zabbix[history_str]` 365
`zabbix[items]` 365
`zabbix[items_unsupported]` 365
`zabbix[log]` 365
`zabbix[queue]` 365
`zabbix[rcache]` 366
`zabbix[requiredperformance]` 366
`zabbix[trends]` 365
`zabbix[triggers]` 365
`zabbix[uptime]` 365
`zabbix[wcache]` 366

Internet Relay Chat. *See IRC*

IPMI
about 136
data, gathering 136
DRAC 136
DRAC IPMI access, configuring 137, 138
IPMI about 69
items, setting up 138

IPMI items, setting up
card attached, to an already monitored host 139
card attached, to different host 139, 140
creating 140, 142

IRC 393

issue management systems integration, actions
bugzilla 196
Computer Associate Unicenter Service Desk 197
steps 196

item
about 48
copying 103-105

item configuration
categories, monitoring 67
item types 69
monitoring, ways 70, 71

itemid field 342

K

key parameter, aggregate items 279

key value, proxy
entering 331

L

load
creating 64, 65

logeventid field 342

M

major level upgrade, Zabbix upgrading
data, gathering 360
database, patching 358, 359
frontend configuration file 361
steps 357

Management Information Base. *See MIB*

maps, single elements
about 237
available map elements 249
creating 238-241
customizing 245-249
elements, highlighting 246
elements, linking 241-244
global map options 247, 249
links, labeling 245
macros in labels 245

mass update
using 98-100

MIB 111

minor level upgrade, Zabbix upgrading.
See major level upgrade, Zabbix upgrading

multiple templates
using 214, 215

N

nested templates
about 217
advantages 219

creating 217, 219
example 219

Net-SNMP
SNMPv3, using with 112
using 108-112

netstat command 288

network monitoring
about 7
alerting 8
data gathering 8
data storage 8
features 8
open source monitoring 8
visualization 8
Zabbix 9

O

Object Identifier. *See* **OID**
OID 111

P

passive items, Zabbix agent
creating 76-80
items, closing 81
modifications 82

patch utility 400

performance, considering
data, splitting 371
database buffering, tuning 370
hardware performance, increasing 372
query count, reducing 369, 370

performance counters, querying
about 314
aliases, using 318
numeric references, using 315-317

positional parameter
using 97, 98

proxy
alternatives 324
benefits 329-331
configuration, tweaking 333
key value, entering 331
limitation 330
reliability 331, 332
setting up 325

uses 323, 324
using, to monitor host 327-329
working 324

proxy, setting up
compiling 325
requirements 325
steps 325, 326

ProxyLocalBuffer option 333

ProxyOfflineBuffer option 333

Q

query count, performance
reducing 369,370

R

raw data
about 335
data, using in remote site 340-342
database, querying 337-340
extracting, from frontend 335, 336

S

severity field 342

Simple Network Management Protocol. *See* **SNMP**

simple reports
about 261
Availability report 263, 264
Most busy triggers top 100 264, 265
status of Zabbix 261-263

single elements
about 222
graphs 222
maps 237

Slackware
about 27
script, creating 27-30
service state, verifying 31

SNMP
about 107
Net-SNMP, using 108-112
new MIBs, adding 113, 114
SNMPv3 108
working with, in Zabbix 115-118

snmpget command 111
SNMP items, working with in Zabbix
about 115, 118
creating 116
dynamic indexes 119-123
OIDs, translating 119
trap handling schemes 129
traps, receiving 123-129
snmpstatus command 109
SNMP Trap Translator. *See* **SNMPTT**
SNMPTT 136
SNMPv3
using, with Net-SNMP 112
source field 342
SourceIP option 389
StartIPMIPollers option 389
StartIPMIPollers parameter
steps 326
SUSE Linux Enterprise Server
probe parameter 26
startup scripts 18-26

T

template, linking to host
configuration, changing 211
creating 204-206
in Zabbix 203, 204
linking to host 206
macro, using 212, 213
macro usage, effects 213
multiple templates, using 214
particular item, modifying 210
steps 206-211
unlinking 216, 217
timestamp field 342
trap handling schemes
about 129
custom mapping 129
database lookups 130-136
SNMPTT 136
trigger expressions
constructing 174-176
human-readable constants 177
timeout triggers 177

triggers
about 48
dependencies 168-173
event, generating 178, 179
event details 177, 178
expressions 166, 168
expressions, creating 173
hysteresis 179
name 165
severity 167
states 178
troubleshooting
actions 390
devices, monitoring 386
frontend 383, 384
frontend, locking out 385
installation 381
triggers 390

U

user
account options 150-153
authentication methods 149
creating 150-153
groups, creating 156
new group, creating 156-163
testing 154, 155
user management, database
about 342, 344
alias field 343
attempt_clock field 343
attempt_failed field 343
attempt_ip field 343
autologout field 343
lang field 343
name field 343
passwd field 343
refresh field 343
rows_per_page field 343
surname field 343
theme field 343
url field 343
userid field 343
user parameters 287

V

value field 342

value mapping
using 100-102

W

web frontend

environment, setting up 32
installing 36-41
prerequisites 32

web frontend installation

choices summary 38
configuration file, writing 39
database, accessing 37
logging in 41-43
installation wizard 33, 34
license 34
PHP prerequisites 35, 36
steps 36-41
wizard, finishing 40
Zabbix server details 37

web pages, monitoring 303

scenario, creating 303-310

Windows-specific, monitoring

performance counters, querying 314, 315
services, monitoring 318-320
Zabbix agent, installing 310-313

Windows service, monitoring

automatic service, checking 320
steps 318-320

X

xz 376

xzcat 376

Z

Zabbix

about 9, 391
action log 374
architecture 9
backing up 374

bar reports 265

community 391

concepts 45

configuration, changes 372, 373

configuration, reviewing 369

data gathering 48

data visualization 221

development version 394

features 9

forum discussion 392

frontend, exploring 45

graphs 54

host 144

host groups 143

installing 10

internal items 363

Internet Relay Chat 393

net.tcp.service key, supported services 77

performance, considering 369

proxy 323

simple reports 261

SNMP items, working with 115-118

source, obtaining 14

SUSE Linux Enterprise Server 18

templates 203

upgrading 354

web frontend 32

web pages, monitoring 303

wiki 392, 393

Windows-specific, monitoring 310

Zabbix, architecture

about 10

proxy 10

Zabbix database 10

Zabbix server 10

Zabbix agent

active items 82

checks, performing 93

host, adding 74, 75

installing 73, 74

passive items 75

starting up 74

supported items 93

using 74

Zabbix installation

- about 11
 - distribution packages, benefits 12
 - hardware, requirements 13
 - options 12
 - prerequisites 13
 - software, requirements 13
 - web frontend screen 11
- Zabbix installation, setting up**
- common problems, troubleshooting 381
- Zabbix source**
- compilation 14, 15
 - database, creating 16

database, populating 17

initial configuration 15

user, creating 17

information flow 63

Zabbix, upgrading

- change level upgrading 354, 355
- compatibility 361
- general policy 353
- major level upgrade 357
- minor level upgrade 357

zcat 376



Thank you for buying Zabbix 1.8 Network Monitoring

Packt Open Source Project Royalties

When we sell a book written on an Open Source project, we pay a royalty directly to that project. Therefore by purchasing Zabbix 1.8 Network Monitoring, Packt will have given some of the money received to the Zabbix project.

In the long term, we see ourselves and you—customers and readers of our books—as part of the Open Source ecosystem, providing sustainable revenue for the projects we publish on. Our aim at Packt is to establish publishing royalties as an essential part of the service and support a business model that sustains Open Source.

If you're working with an Open Source project that you would like us to publish on, and subsequently pay royalties to, please get in touch with us.

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.

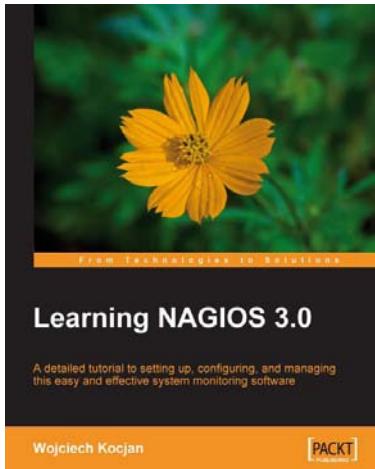
We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

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.



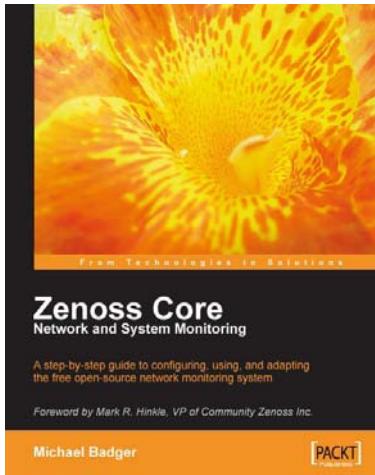
Learning Nagios 3.0

ISBN: 978-1-847195-18-0

Paperback: 316 pages

A comprehensive configuration guide to monitor and maintain your network and systems

1. Secure and monitor your network system with open-source Nagios version 3
2. Set up, configure, and manage the latest version of Nagios
3. In-depth coverage for both beginners and advanced users



Zenoss Core Network and System Monitoring

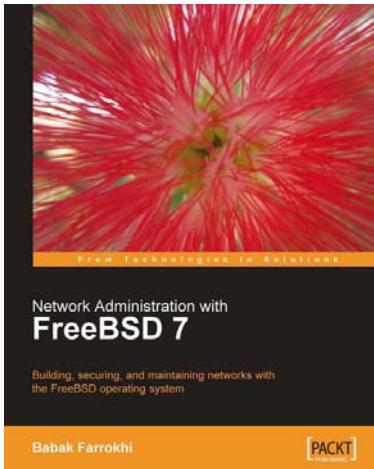
ISBN: 978-1-847194-28-2

Paperback: 280 pages

A step-by-step guide to configuring, using, and adapting this free Open Source network monitoring system - with a Foreword by Mark R. Hinkle, VP of Community Zenoss Inc.

1. Discover, manage, and monitor IT resources
2. Build custom event processing and alerting rules
3. Configure Zenoss Core via an easy to use web interface
4. Drag and drop dashboard portlets with Google Maps integration

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

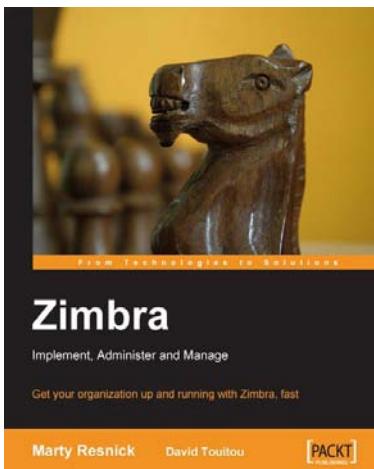


Network Administration with FreeBSD 7

ISBN: 978-1-847192-64-6 Paperback: 280 pages

Building, securing, and maintaining networks with the FreeBSD operating system

1. Set up and manage networking on FreeBSD
2. Virtualization with FreeBSD Jails, IPFW and PF
3. Configure interfaces, protocols, and routing



Zimbra

ISBN: 978-1-847192-08-0 Paperback: 220 pages

Get your organization up and running with Zimbra, fast

1. Get your organization up and running with Zimbra, fast
2. Administer the Zimbra server and work with the Zimbra web client
3. Protect your Zimbra installation from hackers, spammers, and viruses
4. Access Zimbra from Microsoft Outlook

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