Zabbix 5 IT Infrastructure Monitoring Cookbook

Explore the new features of Zabbix 5 for designing, building, and maintaining your Zabbix setup

Nathan Liefting | Brian van Baekel

Zabbix 5 IT Infrastructure Monitoring Cookbook

Explore the new features of Zabbix 5 for designing, building, and maintaining your Zabbix setup

Nathan Liefting

Brian van Baekel

BIRMINGHAM—MUMBAI

Packt

Zabbix 5 IT Infrastructure Monitoring Cookbook

Group Product Manager: Wilson D'souza
Publishing Product Manager: Rahul Nair
Senior Editor: Arun Nadar
Content Development Editor: Pratik Andrade
Technical Editor: Nithik Cheruvakodan
Copy Editor: Safis Editing
Project Coordinator: Neil Dmello
Proofreader: Safis Editing
Indexer: Priyanka Dhadke
Production Designer: Roshan Kawale

First published: January 2021
Production reference: 1290121

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

ISBN 978-1-80020-223-8

`www.packt.com`

To my grandparents, for supporting my education, my brother, for always being at the ready, and my mom and stepdad, for cheering me on. To my girlfriend, for always supporting whatever new idea I get into my head. To my colleagues throughout the years, my first mentor, Sander F., for inspiring me, and Brian, for making it all possible.

– Nathan Liefting

Foreword

2020 was probably for most of us not the most typical year. There was the Covid-19 issue that stopped the world as we know it. To make things worse, I had to deal with some personal and family issues last year, so writing a new book was not top of the list of things I wanted to do in 2020. So, when Packt asked me to write a successor to the *Zabbix Cookbook*, I had to decline.

Of course, I didn't want to let the book be written by someone I didn't believe in, and Packt asked me if I knew someone who was willing and able to write the book.

My first thoughts went to Brian, who I have worked closely with over the years and who I think I can call a friend. I focus on the Belgian market, while Brian focuses on the Dutch and the US markets, but we both speak the same language. Being certified Zabbix trainers, speaking the same language helps a lot when we run into unknown issues when we give training or do some work for a client.

Brian became a certified trainer a few years after me but has grown quickly in a short time and is very dedicated to the job. A few years ago, Brian was even bold enough to start his own business, with the main focus on Zabbix. Last year, when most businesses had to downsize or went broke, Brian managed to even hire someone to expand his company. This is where Nathan came into the picture. Nathan already had some experience with Zabbix, of course, but he managed to help with the book and also became a certified Zabbix trainer in probably one of the most economically difficult years in recent history. So, I think I can say that I am certain this book has been worked on by the best people for the job. I know that Brian and Nathan have spent lots of time on this book in the last year, and I know they are dedicated and knowledgeable. I hope you like what they did with this book, and I hope it will help you with your first steps in setting up and running Zabbix.

Good luck, and thank you Brian and Nathan.

Patrik Uytterhoeven

Open Source Consultant / Zabbix trainer at Open-Future, Nossegem Belgium

Contributors

About the authors

Nathan Liefting, also known as Larcorba, is an IT consultant, trainer, and content creator (artist). He has more than 6 years of professional experience in IT. His experience ranges from managing networks running EVPN/VXLAN to Linux environments and programming. Nathan started working with Zabbix in 2016, when it was still at Zabbix 2 and Zabbix 3 was just released.

He now works for Opensource ICT Solutions BV in the Netherlands as a Zabbix trainer and consultant, designing and building professional Zabbix environments and Zabbix components for some of the biggest companies around the world.

Brian van Baekel quickly discovered how powerful Zabbix is during his career as a network engineer. Ever since, he has been working with Zabbix in various (large) environments, leading to his official Zabbix Certified Trainer certification in early 2017.

In 2018, Brian founded Opensource ICT Solutions BV in the Netherlands and Opensource ICT Solutions LLC in the USA. Both companies primarily focus on building Zabbix environments all over the world. Fun fact: even his cat is named "Zabbix."

About the reviewers

James Cook is a seasoned IT engineer from Perth, Western Australia, specializing in systems administration, monitoring, automation, and programming. He currently works for a large-scale managed service provider (Kinetic IT Pty. Ltd.) leading a team of colleagues who specialize in developing monitoring and automation solutions for both on-premise and cloud technologies. His latest work has focused on developing a Zabbix monitoring solution for his employer's clients that is scalable, catering for multiple tenants, along with developing integration that provides automated service restoration and incident management. James is an experienced programmer, competent in several languages, including C, Python, Ruby, and Perl. He uses these to automate and integrate different products while making the pain of manual tasks disappear.

Justin Addams was born and raised in Western Australia. He is a highly skilled IT professional with over 10 years of industry experience. Most of the time you can find him monitoring systems or automating some inane task out of existence to save someone's sanity. He holds both the Zabbix Certified Specialist and Professional certifications. This, in combination with working a wide array of positions, from small business IT services to infrastructure support to enterprise management systems development, has led to a solid skillset in supporting business requirements through monitoring and automation.

I would like to thank the light of my life, my wife.

Table of Contents

Preface

1
Getting Started with Zabbix and User Management

Technical requirements 2
Installing the Zabbix server 2
Getting ready 3
How to do it... 3
How it works... 6
Setting up the Zabbix frontend 6
Getting ready 7
How to do it... 7
How it works... 12
Using the Zabbix frontend 12
Getting ready 13
How to do it... 14
Navigating the frontend 19
Getting ready 20
How to do it... 20
Creating user groups 26
Getting ready 26
How to do it... 28
Creating your first users 31
Getting ready 31
How to do it... 32
Advanced user authentication with SAML 37
Getting ready 37
How to do it... 39
How it works... 41
There's more... 42

2
Setting Up Zabbix Monitoring

Technical requirements 44
Setting up Zabbix agent 2 monitoring 44
Getting ready 44
How to do it 44
How it works... 50
See also 51
Working with SNMP monitoring 51
Getting ready 51

How to do it... 52
How it works... 55

Creating Zabbix simple checks and the Zabbix trapper 56
Getting ready 56
How to do it... 56
How it works... 61

Working with calculated and dependent items 62
Getting ready 62
How to do it... 62
How it works... 67

Creating external checks 68
Getting ready 68
How to do it... 68
How it works... 70

Setting up JMX monitoring 70
Getting ready 70
How to do it... 71
How it works... 72
See also 73

Setting up database monitoring 73
Getting ready 73
How to do it... 73
How it works... 76
There's more... 76

Setting up HTTP agent monitoring 76
Getting ready 76
How to do it... 77
How it works... 78

Using Zabbix preprocessing to alter item values 79
Getting started 79
How to do it... 79
How it works... 84
See also 85

3
Working with Triggers and Alerts

Technical requirements 88
Setting up triggers 88
Getting ready 88
How to do it... 88
How it works... 93
There's more... 96
See also 97

Setting up alerts 97
Getting ready 97
How to do it... 97
How it works... 104
There's more... 105

Keeping alerts effective 105
Getting ready 105
How to do it... 105
How it works... 108
There's more... 109

Customizing alerts 109
Getting ready 110
How to do it... 110
How it works... 111

4

Building Your Own Structured Templates

Technical requirements 114
Creating your Zabbix template 114
Getting ready 114
How to do it… 114
How it works… 116
There's more… 117

Setting up template applications 118
Getting ready 118
How to do it… 118
How it works… 119
See also 121

Creating template items 121
Getting ready 121
How to do it… 121
How it works… 125
See also 125

Creating template triggers 125
Getting ready 125
How to do it… 126
How it works… 126

Setting up different kinds of macros 127
Getting ready 127
How to do it… 127
How it works… 129
There's more… 131

Using LLD on templates 131
Getting ready 131
How to do it… 131
How it works… 137
See also 140

Nesting Zabbix templates 140
Getting ready 140
How to do it… 141
How it works… 143

5

Visualizing Data, Inventory, and Reporting

Technical requirements 146
Creating graphs for accessing visual data 146
Getting ready 146
How to do it… 147
How it works… 153

Creating maps to keep an eye on infrastructure 153
Getting ready 154
How to do it… 154
How it works… 161

Creating dashboards for getting the right overview 162
Getting ready 162
How to do it… 162
How it works… 167

Setting up Zabbix inventory 167
Getting ready 168
How to do it… 168
How it works… 169

Working through Zabbix reporting 169
Getting ready 169
How to do it... 170

6
Using Discovery for Automatic Creation

Technical requirements 176
Setting up Zabbix Agent discovery 176
Getting ready 176
How to do it... 176
How it works... 181
There's more... 181
Setting up Zabbix SNMP discovery 182
Getting ready 182
How to do it... 182
How it works... 186
Working with Active agent autoregistration 187
Getting ready 187
How to do it... 187
How it works... 192
There's more... 193
Using the new Windows performance counter discovery 193
Getting ready 193
How to do it... 194
How it works... 198
Discovering JMX objects 199
Getting ready 199
How to do it... 200
How it works... 202
There's more... 203

7
Setting Up Zabbix Proxies

Technical requirements 206
Setting up a Zabbix proxy 206
Getting ready 206
How to do it... 206
How it works... 208
There's more... 209
Working with passive Zabbix proxies 209
Getting ready 209
How to do it... 210
How it works... 211
Working with active Zabbix proxies 212
Getting ready 212
How to do it... 212
How it works... 214
Monitoring hosts with Zabbix proxy 215
Getting ready 215
How to do it... 215
How it works... 217
There's more... 219
See also 219

Using Discovery with Zabbix proxies 220
Getting ready 220
How to do it... 220
How it works... 222

Monitoring your Zabbix proxies 223
Getting ready 223
How to do it... 223
How it works... 228

8

Integrating Zabbix with External Services

Technical requirements 232
Setting up Slack alerting with Zabbix 232
Getting ready 232
How to do it... 232
How it works... 242
See also 243

Setting up Microsoft Teams alerting with Zabbix 243
Getting ready 243
How to do it... 244
How it works... 250
See also 251

Using Telegram bots with Zabbix 251
Getting ready 251
How to do it... 252
How it works... 261
There's more... 262
See also 262

Integrating Atlassian Opsgenie with Zabbix 262
Getting ready 263
How to do it... 263
How it works... 270
There's more... 271

9

Extending Zabbix Functionality with Custom Scripts and the Zabbix API

Technical requirements 274
Using the Zabbix API for extending functionality 274
Getting ready 274
How to do it 275
How it works 280
See also 282

Building a jumphost using the Zabbix API and Python 282
Getting ready 282
How to do it... 283
How it works 289
See also 291

Creating maintenance periods as a Zabbix User 291
Getting ready 291
How to do it 291
How it works 298
See also 299

Enabling and disabling a host from Zabbix maps 299
Getting ready 299
How to do it 299
How it works 306
See also 306

10 Maintaining Your Zabbix Setup

Technical requirements 308

Setting Zabbix maintenance periods 308
Getting ready 308
How to do it... 308
How it works 311

Backing up your Zabbix setup 311
Getting ready 311
How to do it 312
How it works 315
There's more 315

Upgrading the Zabbix backend from older PHP versions to PHP 7.2 or higher 316
Getting ready 316
How to do it 316
How it works 318

Upgrading a Zabbix database from older MariaDB versions to MariaDB 10.5 318
Getting ready 318
How to do it 319
How it works 321

Upgrading your Zabbix setup 321
Getting ready 322
How to do it 322
How it works 328
See also 329

Maintaining Zabbix performance over time 329
Getting ready 329
How to do it 329
How it works 334
There's more 338

11 Advanced Zabbix Database Management

Technical requirements 340

Setting up MySQL partitioning for your Zabbix database 340
Getting ready 340
How to do it... 340
How it works... 348
See also 350

Using the new PostgreSQL TimescaleDB functionality 350
Getting ready 350
How to do it... 350

How it works... 355
See also 356

Securing your Zabbix MySQL database 356
Getting ready 356
How to do it... 357
How it works... 366

12

Bringing Zabbix to the Cloud with Zabbix Cloud Integration

Technical requirements 370
Setting up AWS monitoring 370
Getting ready 370
How to do it... 371
How it works... 376
There's more... 378

Setting up Microsoft Azure monitoring 378
Getting ready 378
How to do it... 378
How it works... 382
There's more... 383

Building your Zabbix Docker monitoring 383
Getting ready 383
How to do it... 383
How it works... 385
There's more... 385

Other Books You May Enjoy

Index

Preface

Welcome to *Zabbix 5, IT Infrastructure Monitoring Cookbook*. IT infrastructure ranges from Windows and Linux to networking and development, and basically anything that runs on computer hardware. In this book, we will go over various subjects useful to anyone in IT that wants to use Zabbix to monitor their IT infrastructure.

Who this book is for

Monitoring systems are often overlooked within IT organizations, but they can provide an overview that will save you time, money, and headaches. This book is for IT engineers that want to learn something about Zabbix 5 and how to use it to bring their IT environments to the next level.

What this book covers

Chapter 1, *Getting Started with Zabbix and User Management*, covers how to set up Zabbix, work your way through its menus, and create your first users.

Chapter 2, *Setting Up Zabbix Monitoring*, covers how to set up almost any type of monitoring within Zabbix.

Chapter 3, *Working with Triggers and Alerts*, covers how to set up triggers and get alerts from them.

Chapter 4, *Building Your Own Structured Templates*, covers how to build templates that are structured and will work wonders for keeping your Zabbix setup organized.

Chapter 5, *Visualizing Data, Inventory, and Reporting*, teaches how to visualize data in graphs, maps, and dashboards. It also covers how to use the Zabbix inventory and reporting functionality.

Chapter 6, *Using Discovery for Automatic Creation*, covers how to use Zabbix discovery for automatic host creation as well as items, triggers, and more with agents, SNMP, WMI, and JMX.

Chapter 7, *Setting Up Zabbix Proxies*, teaches how to set up Zabbix proxies correctly for use in a production environment.

Chapter 8, *Integrating Zabbix with External Services*, teaches how to integrate Zabbix with external services for alerting.

Chapter 9, *Extending Zabbix Functionality with Custom Scripts and API*, covers how to extend Zabbix functionality by using custom scripts and the Zabbix API.

Chapter 10, *Maintaining Your Zabbix Setup*, covers how to maintain a Zabbix setup and keep its performance up over time.

Chapter 11, *Advanced Zabbix Database Management*, teaches how to manage Zabbix databases for an advanced setup.

Chapter 12, *Bringing Zabbix to the Cloud with Zabbix Cloud Integration*, covers how to use Zabbix in the cloud with services such as AWS, Azure, Docker, and Kubernetes.

To get the most out of this book

You should have a good basis in IT to understand the terminology used in this book. This book is best for people with at least a starting knowledge about monitoring systems, Linux, and network engineering.

Software/hardware covered in the book	OS requirements
Zabbix 5	Linux (any)
Python 3	Linux (any), macOS
MySQL	Linux (any)
PostgreSQL	Linux (any)
Apache	Linux (any)

Make sure you have a virtualization environment ready to create virtual machines for use with the recipes. VirtualBox, VMware, or any type of client/hypervisor will do.

If you are using the digital version of this book, we advise you to type the code yourself or access the code via the GitHub repository (link available in the next section). Doing so will help you avoid any potential errors related to the copying and pasting of code.

Download the example code files

You can download the example code files for this book from GitHub at `https://github.com/PacktPublishing/Zabbix-5-Network-Monitoring-Cookbook`. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: `http://www.packtpub.com/sites/default/files/downloads/9781800202238_ColorImages.pdf`.

Conventions used

There are a number of text conventions used throughout this book.

`Code in text`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "It's important to back up all of our Zabbix configuration data, which is located in `/etc/zabbix/`."

A block of code is set as follows:

```
# MariaDB Server
# To use a different major version of the server, or to pin to
a specific minor version, change URI below.
deb [arch=amd64] http://downloads.mariadb.com/MariaDB/
mariadb-10.5/repo/ubuntu xenial main
```

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

```
systemctl start mariadb
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this: "Then we navigate to **Monitoring | Hosts** and click on **Latest data** for the Zabbix server host."

> **Tips or important notes**
> Appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at `customercare@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/support/errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packt.com`.

Whether you are a real Zabbix guru or if you've just started working with Zabbix, this book will include some recipes for everyone. We are going over most of the Zabbix basics and we are even doing some cool stuff with the Zabbix API in the book.

We decided to write this book because we want to supply you with the Zabbix information available online and in official Zabbix training in a clear and straightforward way. We've all been through the process of bookmarking all these amazing community blog posts, community guides, and even official documentation. Sometimes it can be a bit much, which is where this book will help. See it as a guide with something for everyone without the need to Google until your fingers fall off.

Now, even if you are experienced, or have finished this and maybe other books and you've bookmarked every useful page about Zabbix, you might still not know everything. This is where we come in. Zabbix is a free product built on an amazing open source community, but besides that, there are some real Zabbix gurus out there that have decided to make a living out of it. Our company, Opensource ICT Solutions, comes from these humble beginnings and we are there to provide our customers with everything they need when it comes to Zabbix. As a Premium Zabbix partner, we provide the following services:

- Official Zabbix training
- Official Zabbix support
- Zabbix consultancy
- Helpdesk services

So, if you've enjoyed this book, please do think about us and others in our amazing Zabbix community. Give us a follow on LinkedIn (and other social media) and if you ever need help, give us a call! We will definitely be ready to help you out with any questions you might run into.

Brian van Baekel – Founder of Opensource ICT Solutions

Opensource ICT Solutions

WEB: `https://oicts.com`

EMAIL ADDRESS: `info@oicts.com`

PHONE: +1-929-377-1253 or +31(0)72 743 65 83

1
Getting Started with Zabbix and User Management

There have been quite a few changes to the Zabbix UI since 4.4 and earlier in this Zabbix 5 release. This book has been written completely with Zabbix 5, but while reading, you'll find some information detailing the differences between 4 and 5.

In this chapter, we will explore the Zabbix UI to get you familiar with it. We will go over finding your hosts, triggers, dashboards, and more to make sure you feel confident diving into the deeper material later on in this book. The Zabbix UI has a lot of options to explore, so if you are just getting started, don't get overwhelmed. It's quite structurally built actually and once you get the hang of it, I am confident you will find your way without issues.

We will also work on creating our first user groups, users, and some advanced user authentication as a bonus. This way, we will make sure we have a structured Zabbix setup before continuing on with this book. You will learn all about these subjects in the following recipes:

- Installing the Zabbix server
- Setting up the Zabbix frontend
- Using the Zabbix frontend
- Navigating the Zabbix frontend
- Creating user groups
- Creating your first users
- Advanced user authentication with SAML

Technical requirements

We'll be starting this chapter with a server that has already been set up. There are a lot of guides out there detailing how to install a Zabbix server, so if you're not familiar with doing so, make sure to get started with that first. You can find more information about the installation here: `https://www.zabbix.com/download`.

So, before diving into this chapter, make sure you have the following set up:

- A Zabbix server installed from the precompiled Zabbix packages on a Linux distribution of your choice. I recommend CentOS or Ubuntu, but a distribution such as Debian or any other will suit you just as well. We'll be using CentOS throughout the book.
- MariaDB set up to work with your Zabbix server. PostgreSQL is fine as well.
- NGINX or Apache set up to serve the Zabbix frontend.
- A working Zabbix agent to monitor the Zabbix server.

Installing the Zabbix server

Before doing anything within Zabbix, we need to install it and get ready to start working with it. In this recipe, we are going to discover how to install Zabbix server 5.

Getting ready

Before we actually install the Zabbix server, we are going to need to fulfill some prerequisite requirements. We will be using **MariaDB** mostly throughout this book. MariaDB is popular and a lot of information is available on the use of it with Zabbix.

At this point, you should have a prepared Linux server in front of you running either an RHEL- or Debian-based distribution. I'll be installing CentOS 8 and Ubuntu 20 on my server; let's call them `lar-book-centos` and `lar-book-ubuntu`.

When you have your server ready, we can start the installation process.

How to do it…

1. Let's start by adding the Zabbix 5.0 repo to our system.

   For RHEL-based systems, use the following:

   ```
   rpm -Uvh https://repo.zabbix.com/zabbix/4.5/rhel/8/
   x86_64/zabbix-release-4.5-2.el8.noarch.rpm
   dnf clean all
   ```

   For Debian-based systems, use the following:

   ```
   wget https://repo.zabbix.com/zabbix/4.5/ubuntu/pool/
   main/z/zabbix-release/zabbix-release_4.5-1+focal_all.deb
   dpkg -i zabbix-release_4.5-1+focal_all.deb
   sudo apt update
   ```

2. Now that the repo is added, let's install MariaDB on our server.

   For RHEL-based systems, use the following:

   ```
   dnf install mariadb-server
   systemctl enable mariadb
   systemctl start mariadb
   ```

   For Debian-based systems, use the following:

   ```
   apt-get install mariadb-server
   systemctl enable mariadb
   systemctl start mariadb
   ```

3. After installing MariaDB, make sure to secure your installation with the following command:

```
sudo /usr/bin/mysql_secure_installation
```

4. Run through the secure installation setup and make sure to remember your password.
5. Now, let's install our Zabbix server with MySQL support.

   For RHEL-based systems, use the following:

```
dnf install zabbix-server-mysql
systemctl enable mariadb
systemctl start mariadb
```

   For Debian-based systems, use the following:

```
apt-get install zabbix-server-mysql
systemctl enable mariadb
systemctl start mariadb
```

6. With the Zabbix server installed, we are ready to create our Zabbix database. Log in to MariaDB with the following:

```
sudo mysql -u root -p
```

7. Enter the password you set up during the secure installation and create the Zabbix database with the following:

```
create database zabbix character set utf8 collate utf8_
bin;
grant all privileges on zabbix.* to zabbix@localhost
identified by 'password';
flush privileges;
quit
```

8. Now we need to import our Zabbix database scheme to our newly created Zabbix database:

```
zcat /usr/share/doc/zabbix-server-mysql*/create.sql.gz |
mysql -u zabbix -p zabbix
```

Important note

At this point, it might look like you are stuck and the system is not responding. Do not worry though as it will just take a while to import the SQL scheme.

We are now done with the preparations at our MariaDB side and are ready to move on to the next step, which will be configuring the Zabbix server:

1. The Zabbix server is configured using the Zabbix server config file. This file is located in `/etc/zabbix/`. Let's open this file with our favorite editor; I'll be using Vim throughout the book:

```
vim /etc/zabbix/zabbix_server.conf
```

2. Now, edit the following lines in the file:

```
DBName=zabbix
DBPassword= password
```

Tip

Before starting the Zabbix server on a CentOS 8 machine, you should configure SELinux to allow the use of the Zabbix server. If this is a test machine, you can use a permissive stance for SELinux, but it might not be smart to use this in production.

3. All done; we are now ready to start our Zabbix server:

```
systemctl enable zabbix-server
systemctl start zabbix-server
```

4. Check whether everything is starting up as expected with the following:

```
systemctl status zabbix-server
```

5. Alternatively, monitor the log file, which provides a detailed description of the Zabbix startup process:

```
tail -f /var/log/zabbix/zabbix_server.log
```

How it works...

The Zabbix server is the main process for our Zabbix setup. It is responsible for our monitoring, problem alerting, and a lot of the other tasks described in this book. A complete Zabbix stack consists of at least the following:

- A database (MySQL, PostgreSQL, or Oracle)
- A Zabbix server
- Apache or NGINX running the Zabbix frontend with PHP 7.2+

We can see the components and how they communicate with each other in the following figure:

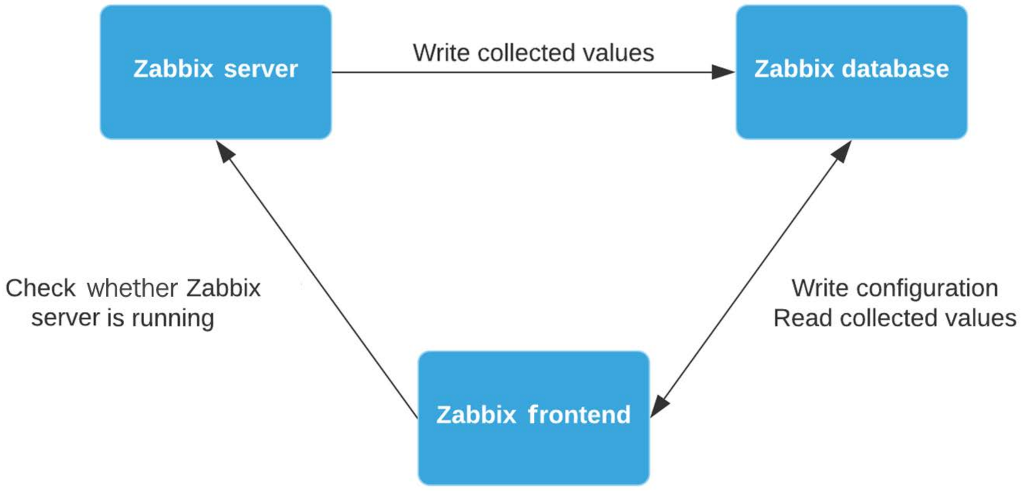


Figure 1.1 – Zabbix setup communications diagram

We've just set up the Zabbix server and database; by running these two, we are basically ready to start monitoring. The Zabbix server communicates with the Zabbix database to write collected values to it.

There is still one problem though: we cannot configure our Zabbix server to do anything. For this, we are going to need our Zabbix frontend, which we'll set up in the next recipe.

Setting up the Zabbix frontend

The Zabbix frontend is the face of our server. It's where we will configure all of our hosts, templates, dashboard, maps, and everything else. Without it, we would be blind to what's going on on the server side, so let's set it up in this recipe.

Getting ready

We are going to set up the Zabbix frontend using Apache. Before starting with this recipe, make sure you are running the Zabbix server on a Linux distribution of your choice. I'll be using the `lar-book-centos` and `lar-book-ubuntu` hosts in these recipes to show the setup process on CentOS 8 and Ubuntu 20.

How to do it...

1. Let's start by installing PHP to our server with the following command.

   For RHEL-based systems, use the following:

   ```
   dnf install php
   ```

   For Debian-based systems, use the following:

   ```
   apt-get install php
   ```

2. We will need to edit our PHP configuration to use the correct time zone; otherwise, the Zabbix frontend will not show us the right time values.

   For RHEL-based systems, use the following:

   ```
   vim /etc/php-fpm.d/zabbix.conf
   ```

   For Debian-based systems, use the following:

   ```
   vim /etc/zabbix/apache.conf
   ```

3. Now, uncomment the following line and set the right time zone for you.

   For RHEL-based systems, use the following:

   ```
   ; php_value[date.timezone] = Europe/Riga
   ```

   For Debian-based systems, use the following:

   ```
   # php_value date.timezone Europe/Riga
   ```

4. Make sure that Apache is installed onto your server with the following.

   For RHEL-based systems, use the following:

   ```
   dnf install httpd
   ```

   For Debian-based systems, use the following:

   ```
   apt-get install apache2
   ```

5. Now that we have fulfilled our requirements, it is time for us to actually install the frontend. Issue the following command to get started.

   For RHEL-based systems, use the following:

   ```
   dnf install zabbix-web-mysql zabbix-apache-conf
   ```

   For Debian-based systems, use the following:

   ```
   apt-get install zabbix-frontend-php zabbix-apache-conf
   ```

Tip

Don't forget to allow ports `80` and `443` in your firewall if you are using one. Without this, you won't be able to connect to the frontend.

6. Restart the Zabbix components and make sure they start up when the server is booted with the following.

   For RHEL-based systems, use the following:

   ```
   systemctl enable httpd php-fpm
   systemctl restart zabbix-server httpd php-fpm
   ```

   For Debian-based systems, use the following:

   ```
   systemctl enable apache2
   systemctl restart zabbix-server apache2
   ```

7. We should now be able to navigate to our Zabbix frontend without any issues and start the final steps to set up the Zabbix frontend.
8. Let's go to our browser and navigate to our server's IP. It should look like this:

   ```
   http://<your_server_ip>/zabbix
   ```

9. We should now see the following web page:

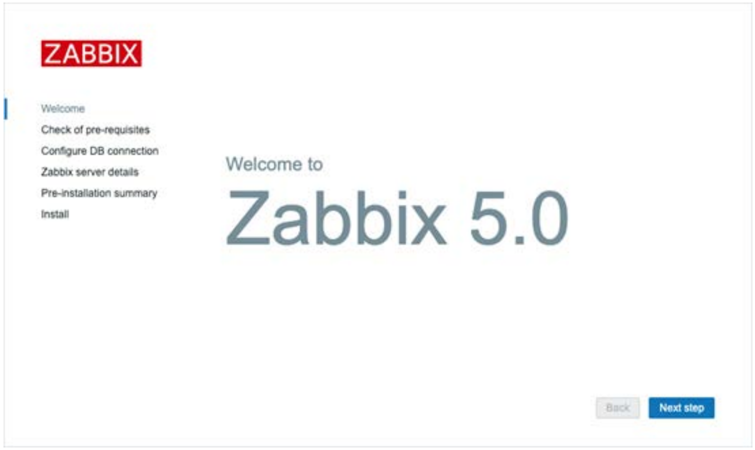
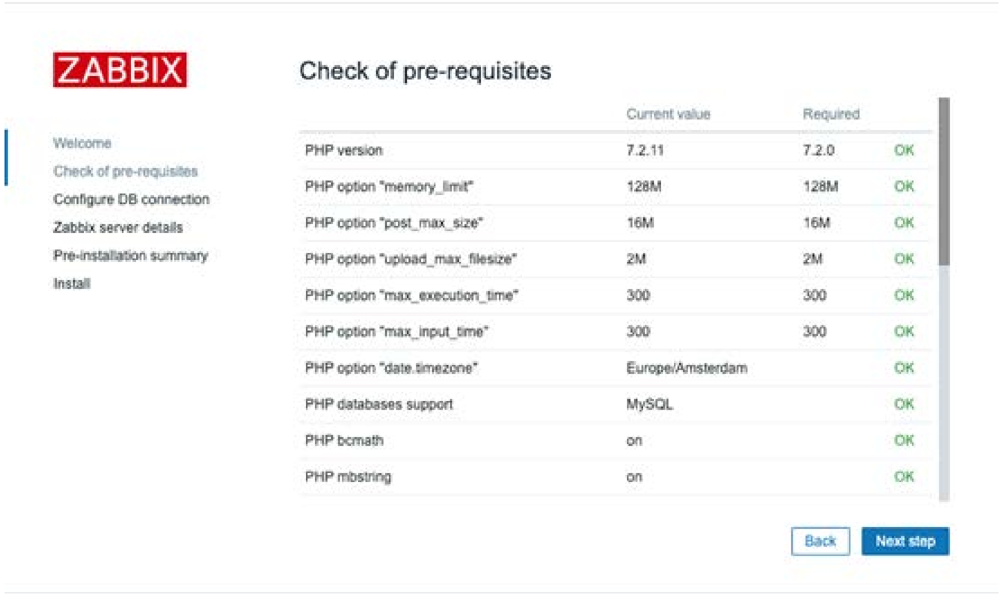


Figure 1.2 – The Zabbix welcome screen

If you don't see this web page, it's possible you have missed some steps in the setup process. Retrace your steps and double-check your configuration files; even the smallest typo could prevent the web page from serving.

10. Let's continue by clicking **Next step** on this page, which will serve you with the next page:

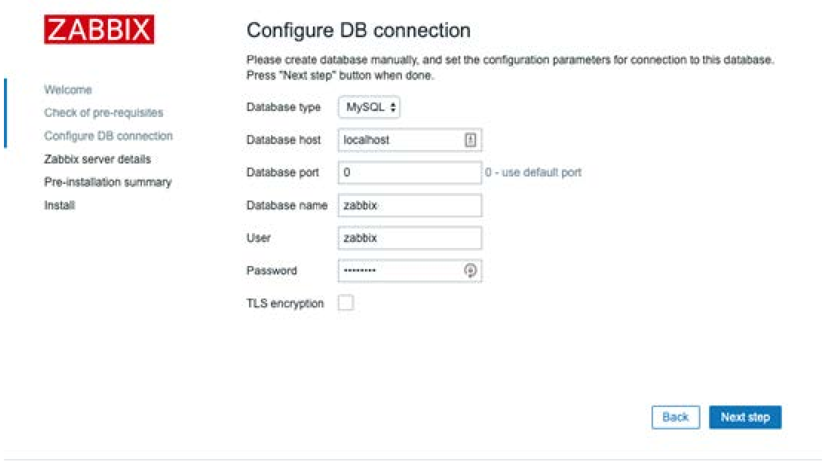


Check of pre-requisites

	Current value	Required	
PHP version	7.2.11	7.2.0	OK
PHP option "memory_limit"	128M	128M	OK
PHP option "post_max_size"	16M	16M	OK
PHP option "upload_max_filesize"	2M	2M	OK
PHP option "max_execution_time"	300	300	OK
PHP option "max_input_time"	300	300	OK
PHP option "date.timezone"	Europe/Amsterdam		OK
PHP databases support	MySQL		OK
PHP bcmath	on		OK
PHP mbstring	on		OK

Figure 1.3 – The Zabbix installation pre-requisites page

11. Every single option here should be showing **OK** now; if not, fix the mistake it's showing you. If everything is **OK**, you may proceed by clicking **Next step** again, which will take you to the next page:



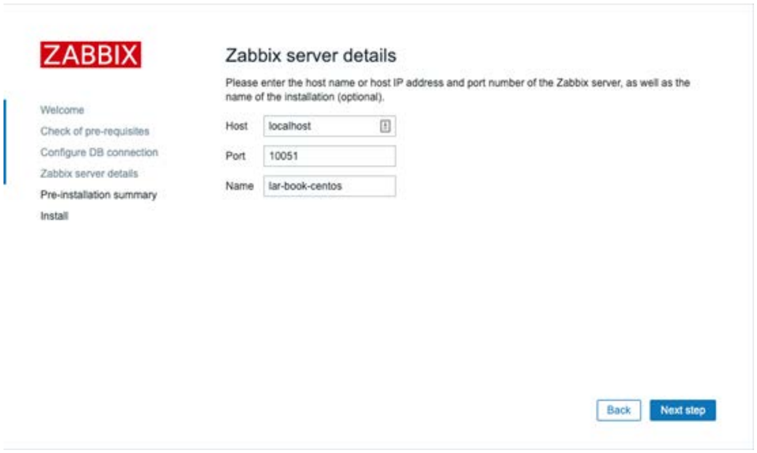
Configure DB connection

Please create database manually, and set the configuration parameters for connection to this database.
Press "Next step" button when done.

Database type	MySQL
Database host	localhost
Database port	0 — 0 - use default port
Database name	zabbix
User	zabbix
Password	••••••••
TLS encryption	☐

Figure 1.4 – The Zabbix installation DB connection page

12. Here, we need to tell our Zabbix frontend where our MySQL database is located. Since we installed it on **localhost**, we just need to change the database name and fill in the password.
13. This should make the Zabbix frontend able to communicate with the database. Let's proceed by clicking **Next step** again:



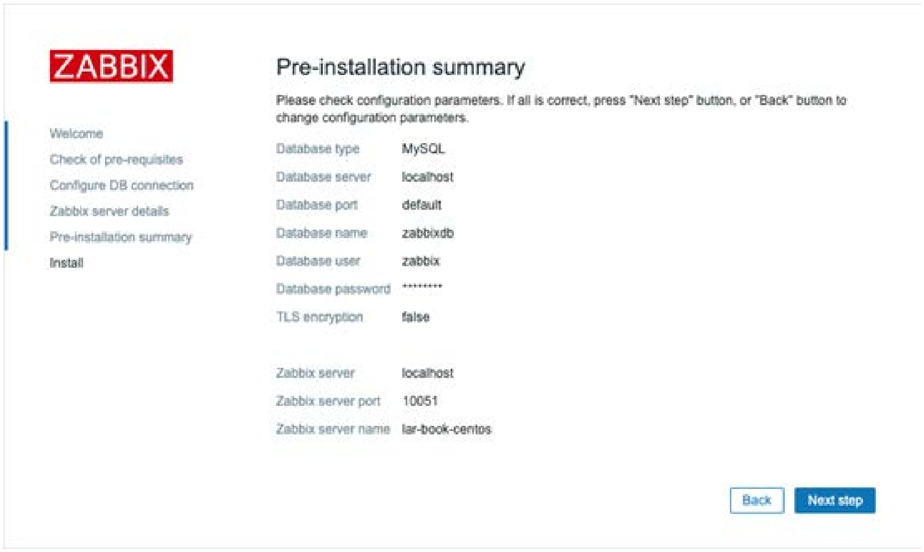
Zabbix server details

Please enter the host name or host IP address and port number of the Zabbix server, as well as the name of the installation (optional).

Host	localhost
Port	10051
Name	lar-book-centos

Figure 1.5 – The Zabbix installation server details page

14. Next up is the Zabbix server configuration. Make sure to name your server something useful or something cool. For example, I've set up a production server called `Meeseeks` because every time we get an alert, Zabbix will say "I'm Mr. Meeseeks look at me."
15. Let's name our server and proceed to the next step:



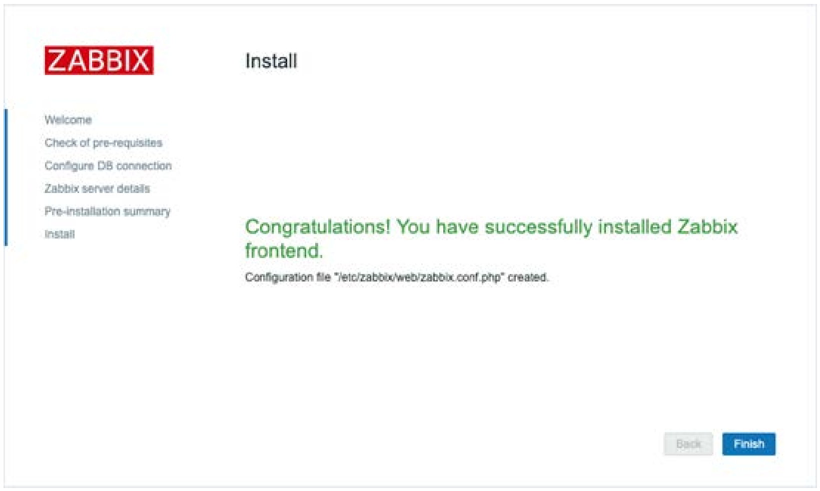
Pre-installation summary

Please check configuration parameters. If all is correct, press "Next step" button, or "Back" button to change configuration parameters.

Database type	MySQL
Database server	localhost
Database port	default
Database name	zabbixdb
Database user	zabbix
Database password	********
TLS encryption	false
Zabbix server	localhost
Zabbix server port	10051
Zabbix server name	lar-book-centos

Figure 1.6 – The Zabbix installation summary page

16. Verify your settings and proceed to click **Next step** one more time.



Install

Congratulations! You have successfully installed Zabbix frontend.

Configuration file "/etc/zabbix/web/zabbix.conf.php" created.

Figure 1.7 – The Zabbix installation finish page

17. You have successfully installed the Zabbix frontend. You may now press the **Finish** button and we can start using the frontend. You'll be served with a login page where you can use the following default credentials:

```
Username: Admin
Password: zabbix
```

How it works...

Now that we've installed our Zabbix frontend, our Zabbix setup is complete and we are ready to start working with it. Our Zabbix frontend will connect to our database to edit the configuration values of our setup, as we can see in the following figure:

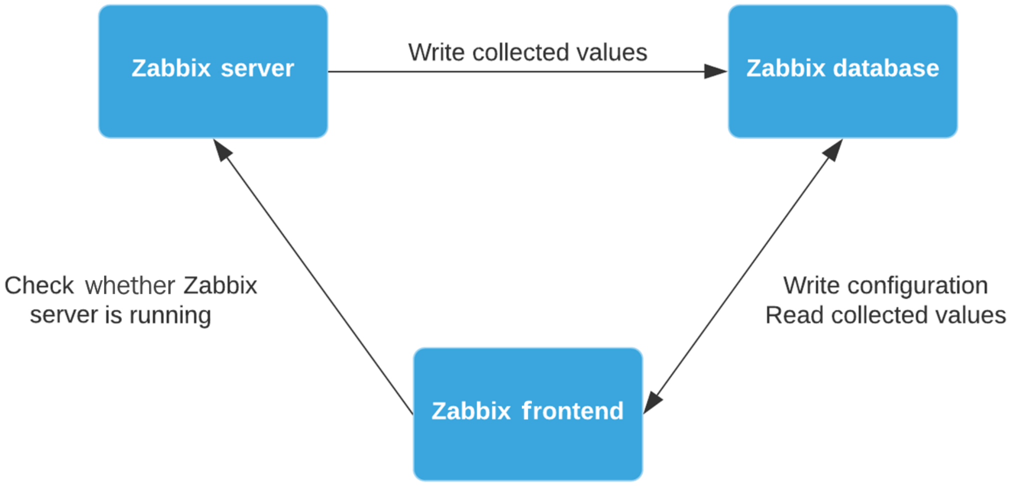


Figure 1.8 – Zabbix setup communications diagram

The Zabbix frontend will also talk to our Zabbix server, but this is just to make sure the Zabbix server is up and running. Now that we know how to set up the Zabbix frontend, we can start using it. Let's check this out in the next recipe.

Using the Zabbix frontend

If this is your first time using Zabbix, congratulations on getting to the UI. If this is your first time using Zabbix 5.0, surprise, this is the new layout! We'll be going over some of the different elements that we can find in the Zabbix frontend so that during this book, you'll feel confident in finding everything you need.

Getting ready

To get started with the Zabbix UI, all we need to do is log in to the frontend. You will be served with the following page at the IP on which your server is running the Zabbix frontend:

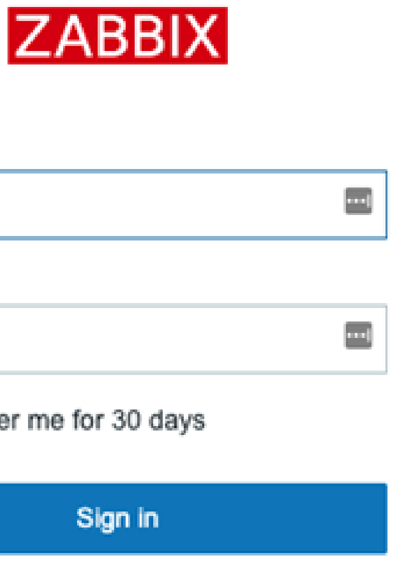


Figure 1.9 – The Zabbix login screen

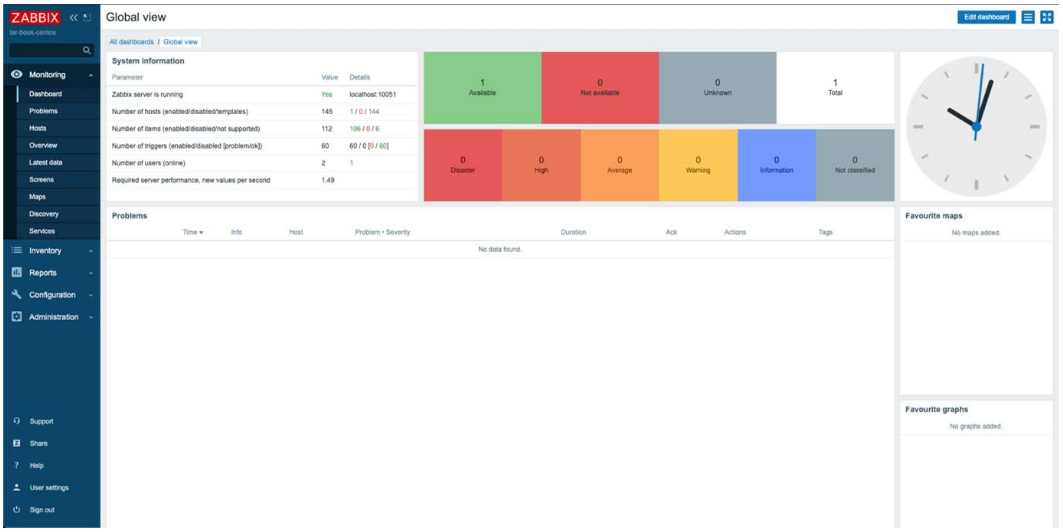
Make sure you log in to the Zabbix frontend with the default credentials:

- **Username**: `Admin`
- **Password**: `zabbix`

> **Tip**
> Just like in Linux, Zabbix is case-sensitive in most places. When entering your username, make sure to include the right cases; otherwise, you won't be able to log in!

How to do it…

After you log in, you'll be served with the default page, which is the default dashboard. This is what Zabbix has called **Global view** and it provides us with a nice overview of what's going on. We can completely customize this and all the other dashboards that Zabbix supplies, but it's a good idea to familiarize yourself with the default setup before building something new:



System information

Parameter	Value	Details
Zabbix server is running	Yes	localhost:10051
Number of hosts (enabled/disabled/templates)	145	1 / 0 / 144
Number of items (enabled/disabled/not supported)	112	106 / 0 / 6
Number of triggers (enabled/disabled [problem/ok])	60	60 / 0 [0 / 60]
Number of users (online)	2	1
Required server performance, new values per second	1.49	

Available	Not available	Unknown	Total
1	0	0	1

Disaster	High	Average	Warning	Information	Not classified
0	0	0	0	0	0

Problems

Time ▾	Info	Host	Problem • Severity	Duration	Ack	Actions	Tags

No data found.

Favourite maps

No maps added.

Favourite graphs

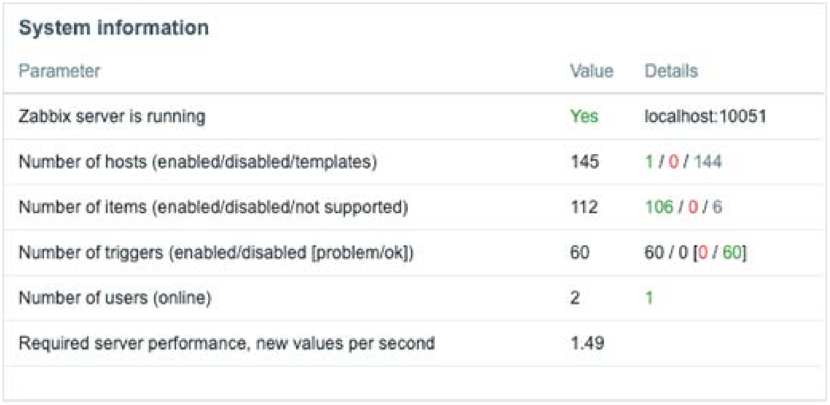
No graphs added.

Figure 1.10 – The Global view dashboard

So, let's get started on getting to know this Zabbix 5.0 frontend by looking at the default dashboard. Please follow along in the frontend by clicking and checking out the content mentioned.

Zabbix uses *dashboards* and they are filled with *widgets* to show you the information. Let's go over the different widgets in the default dashboard and detail their information.

From left to right, let's start with the **System information** widget:



System information

Parameter	Value	Details
Zabbix server is running	Yes	localhost:10051
Number of hosts (enabled/disabled/templates)	145	1 / 0 / 144
Number of items (enabled/disabled/not supported)	112	106 / 0 / 6
Number of triggers (enabled/disabled [problem/ok])	60	60 / 0 [0 / 60]
Number of users (online)	2	1
Required server performance, new values per second	1.49	

Figure 1.11 – The System information widget

This is the **System information** widget, which as you might have guessed details all the system information to you. This way, we can keep an eye on what's going on with our Zabbix server and see whether our Zabbix is even running. Let's go over the parameters:

- **Zabbix server is running**: This details to us whether the Zabbix server backend is reachable, and we can see where it is reachable. In this case, the Zabbix frontend can reach the Zabbix server, and it is running on **localhost:10051**.
- **Number of hosts**: This parameter displays the number of hosts **enabled** (**1**), the number of hosts **disabled** (**0**), and the number of **templates** we have (**144**). It gives us a quick overview of our Zabbix server host information.
- **Number of items**: Here, we can see the details of our Zabbix server's items: in this case, **enabled** (**106**), **disabled** (**0**), and **not supported** (**6**).
- **Number of triggers**: This details the number of triggers to us. We can see how many are **enabled** (**60**) and **disabled** (**0**), as well as how many are in a **problem** state (**0**) and how many are in the **ok** state (**60**).
- **Number of users (online)**: The first value details the total number of users. The second value details the numbers of users currently logged in to the Zabbix frontend.
- **Required server performance, new values per second**: Perhaps I'm introducing you to a completely new concept here, which is **New Values Per Second**, or **NVPS**. A Zabbix server receives or requests values through items and writes this to our MariaDB (or another database). The NVPS detailed here shows the average number of NVPS received by the Zabbix server. Keep a close eye on this as your Zabbix server grows; it's a good indicator to see how fast you should scale up.

Now, that's one of the most important widgets when it comes to your Zabbix server and it's a great one to keep on your main dashboard if you ask me. Let's move on to the next widget, **Host availability**:

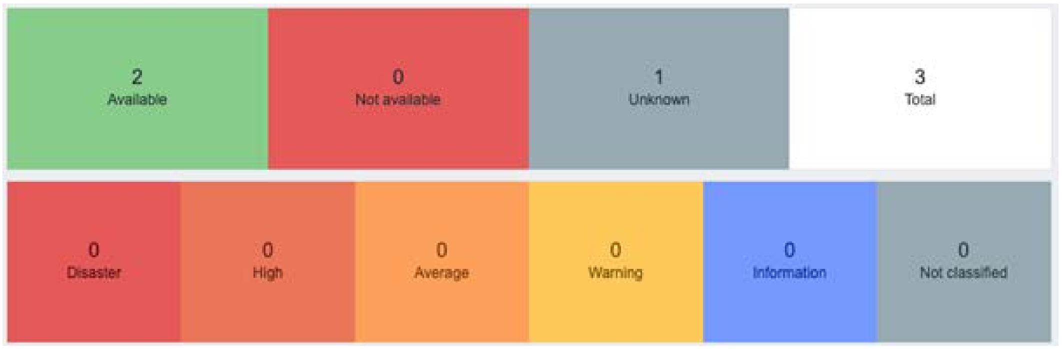


Figure 1.12 – The Host availability widget

The **Host availability** widget is a quick overview widget showing you everything you want to know about your monitored host's availability status. In this widget, it shows whether the host is **Available**, **Not Available**, or **Unknown**. This way, you get a good overview of the availability of all the hosts you could be monitoring with your Zabbix server in a single widget.

On top of that, it also shows you how many hosts currently have a trigger in a certain state. There are several default states in Zabbix:

- **Disaster**
- **High**
- **Average**
- **Warning**
- **Information**
- **Not classified**

We can fully customize the severity levels; for example, what severity levels we want to put on which triggers. So, if you are worried about the severities right now, don't be; we'll get to that later.

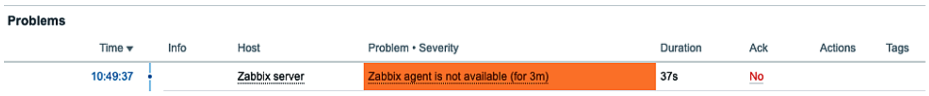
Tip

Customizing the severity levels can be very useful to your organization. We can customize the severity levels to match levels used throughout our company or even to match some of our other monitoring systems used.

The next widget is **Local**:

Figure 1.13 – The Local widget, indicating a time

It's a clock with the local time, need I say more? Let's move on to the **Problems** widget:



Problems

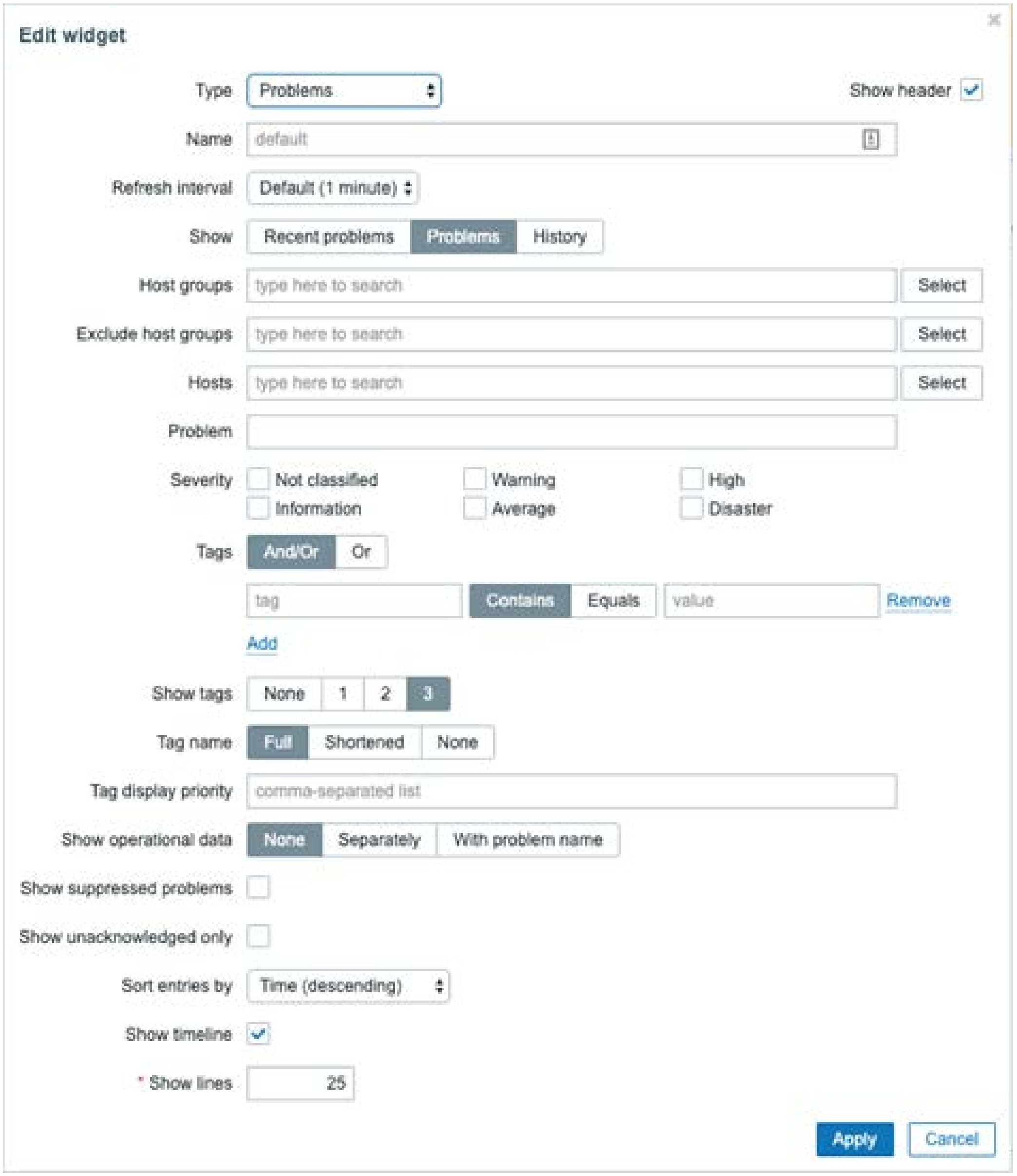
Time ▾	Info	Host	Problem • Severity	Duration	Ack	Actions	Tags
10:49:37		Zabbix server	Zabbix agent is not available (for 3m)	37s	No		

Figure 1.14 – One of the Problems widgets available

Now, this is an interesting widget that I use a lot. We see our current problems on this screen, so if we have our triggers set up correctly, we get valuable information here. A quick overview of how many hosts are having problems is one thing, but the **Problems** screen also gives us more details about the problem:

- **Time**: At what time this problem was noticed by the Zabbix server first
- **Info**: Details information about the problem.
- **Host**: What host this problem occurred on.
- **Problem/Severity**: What the problem is and how severe it is. The severity is shown in a color, in this case orange meaning **Average**.
- **Duration**: How long this has been a problem.
- **Ack**: Whether this problem has been acknowledged, and if you hover over **Yes** or **No**, it will show you the acknowledged details.
- **Actions**: What actions have been taken after this problem occurred, for example, a custom script that executes on problem creation.
- **Tags**: What tags are assigned to this problem.

The **Problems** widget is very useful. We have different types of this widget available and as mentioned before, it is completely customizable, based on how this widget shows our problems to us. Take a quick look at some of the options, which we'll detail further in a later chapter:



Edit widget

Type	Problems
Show header	✓
Name	default
Refresh interval	Default (1 minute)
Show	Recent problems [✓ Problems] History
Host groups	type here to search
Exclude host groups	type here to search
Hosts	type here to search
Problem	
Severity	Not classified, Information, Warning, Average, High, Disaster
Tags	[✓ And/Or] Or
	tag [✓ Contains] Equals value Remove
	Add
Show tags	None 1 2 [✓ 3]
Tag name	[✓ Full] Shortened None
Tag display priority	comma-separated list
Show operational data	[✓ None] Separately With problem name
Show suppressed problems	
Show unacknowledged only	
Sort entries by	Time (descending)
Show timeline	✓
* Show lines	25

Figure 1.15 – The Edit widget screen

Tip

We can hide severity levels from these widgets to make sure we only see important ones. Sometimes, we don't want to see informational severity problems on our dashboards; it can distract you from a more important problem. Keep your dashboards clean by customizing the widget to its full extent.

Now, there are two more widgets that are completely empty on our default dashboard. These are the **Favourite maps** and **Favourite graphs** widgets. These widgets can be filled with a quick link to your favorite maps and graphs, respectively, giving you a fast way to access them without clicking through the menus:

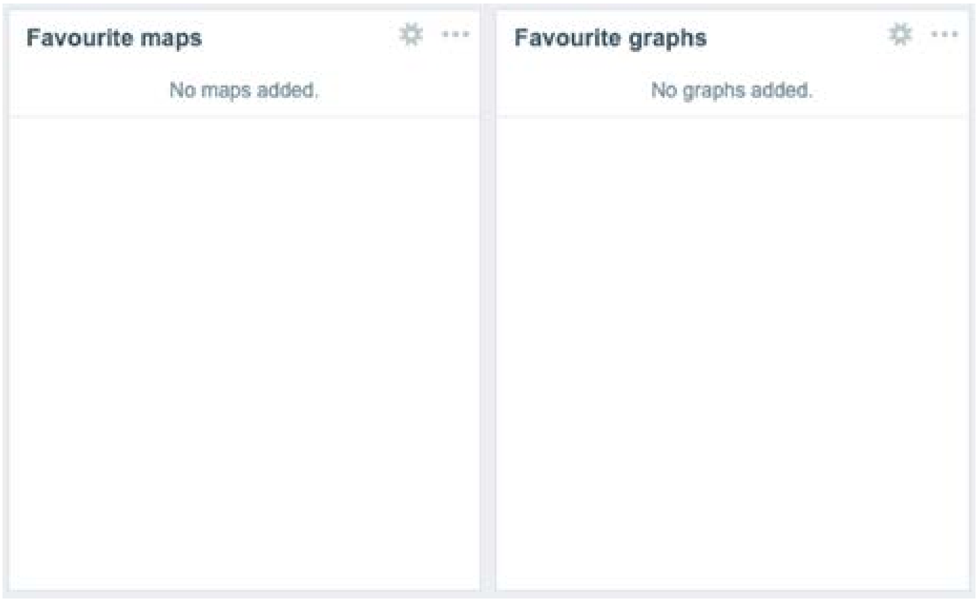


Figure 1.16 – The Favourite widgets

Now we know how to work with the Zabbix frontend and we can continue further on with how to navigate our instance.

Navigating the frontend

Navigating the Zabbix frontend is easier than it looks at first glance, especially with some of the amazing changes made in Zabbix 5.0. Let's explore the Zabbix navigational UI some more in this recipe by looking at the navigation bar and what it has to offer.

Getting ready

Now that we've seen the first page after logging in with the default dashboard, it is time to start navigating through the Zabbix UI and see some of the other pages available. We'll move through the sidebar and explore the pages available in our Zabbix installation so that when we start monitoring our networks and applications, we know where we can find everything.

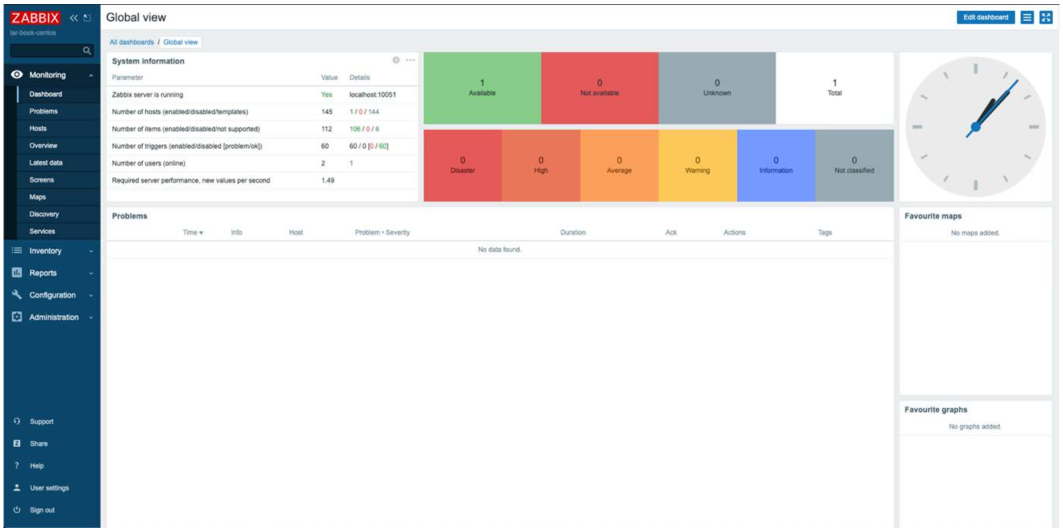
So, before continuing, make sure you have the Zabbix server ready as set up in the previous recipe.

How to do it…

The Zabbix navigation bar is the gateway to all of our powerful tools and configuration settings. Zabbix uses a left-side navigation bar to keep our UI as clean as possible. On top of that, they have made the sidebar disappearing so that we can keep a close look at all of our content, without the sidebar blocking our vision.

> **Tip**
>
> We cannot change the Zabbix navigation menu location, but it is possible to keep it from hiding. If you want the navigation bar to always be on top, simply click the button next to the Zabbix logo.

Let's take a look at the Zabbix sidebar as we see it from our default page and get to know it. Please follow along in the frontend by clicking and checking out the content mentioned:



System information

Parameter	Value	Details
Zabbix server is running	Yes	localhost:10051
Number of hosts (enabled/disabled/templates)	145	1 / 0 / 144
Number of items (enabled/disabled/not supported)	112	106 / 0 / 6
Number of triggers (enabled/disabled [problem/ok])	60	60 / 0 [0 / 60]
Number of users (online)	2	1
Required server performance, new values per second	1.49	

Problems

Time ▾	Info	Host	Problem • Severity	Duration	Ack	Actions	Tags
No data found.							

Figure 1.17 – The default Zabbix page

We've got some categories here to choose from, and one level below the categories, we've got our different pages. First, let's start by detailing the categories:

- **Monitoring**: The **Monitoring** category is where we can find all of our information about our configured hosts. It's basically the category you want to use when you're working with Zabbix to read any collected information you've worked hard to acquire.
- **Inventory**: The **Inventory** category is a cool extra feature in Zabbix that we can use to look at our host-related inventory information. You can add stuff such as software versions or serial numbers to hosts and look at them here.
- **Reports**: The **Reports** category contains a variety of predefined and user-customizable reports focused on displaying an overview of parameters such as system information, triggers, and gathered data.
- **Configuration**: The **Configuration** category is where we build everything we want to see in monitoring, inventory, and reporting. We can edit our settings to suit our every need so that Zabbix can show us that data in a useful way.
- **Administration**: This **Administration** category is where we administer the Zabbix server. You'll find all your settings from the server here to enable you and your colleagues to have a good working Zabbix experience.

You'll go over all of these quite a lot while using this book, so remember them well. Let's dive a little deeper into the categories by looking at them one by one. Let's start with the **Monitoring** category:

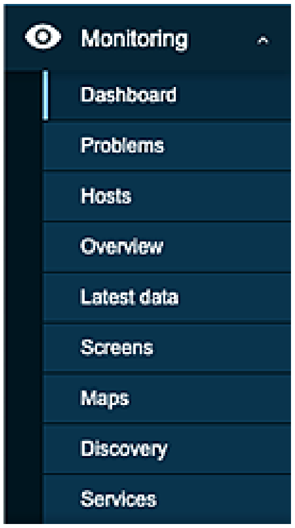


Figure 1.18 – The Monitoring section of the sidebar

The **Monitoring** tab contains the following pages:

- **Dashboard**: This is where you will find the default dashboard we showed in *Figure 1.10*. It is also where we can add many more dashboards for everything we can think of.
- **Problems**: We can look at our current problem in detail here. We are provided with a bunch of filter options to narrow down our problem search if needed.
- **Hosts**: The **Hosts** tab is our spot for seeing what's going on with a specific host and seeing what monitoring options are available to work with for that host.
- **Overview**: The **Overview** page is where we can get a general overview of data and triggers for all of our hosts.
- **Latest data**: Here is a page we're going to use quite a lot throughout our professional Zabbix lifetime. The **Latest data** page is where we can find collected values for every single host, which we can of course filter on.
- **Screens**: Screens are the predecessor of dashboards and I don't recommend configuring them anymore as they will be merged into dashboards in a later version of Zabbix. Use dashboards instead as it will make a possible migration a lot easier.
- **Maps**: Maps are a very helpful tool in Zabbix to get an overview of your infrastructure. We can use them for network overviews and such.
- **Discovery**: This page provides us with an overview of discovered devices. We'll work more on this later.
- **Services**: You'll find your configured services with **service level agreement** (**SLA**) information here.

Next, we have the **Inventory** category:

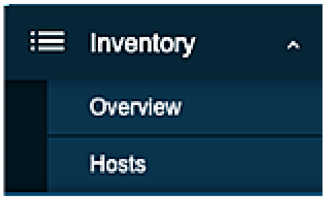


Figure 1.19 – The Inventory section of the sidebar

The **Inventory** tab contains the following pages:

- **Overview**: A quick overview page for your inventory information.
- **Hosts**: A more detailed look into inventory values on a per-host base.

Next, we have the **Reports** category:

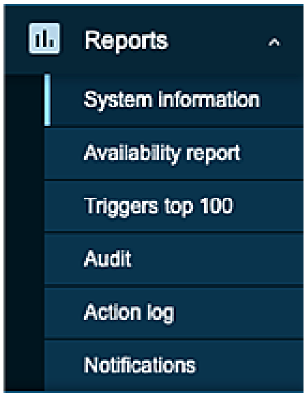


Figure 1.20 – The Reports section of the sidebar

The **Reports** tab contains the following pages:

- **System information**: You can look at the system information here; it contains the same information as the **System information** widget.
- **Availability report**: On this page, we can see the percentage of time a trigger has been in a **problem** state compared to the **ok** state. This is a helpful way of seeing for how long certain items are actually healthy.
- **Triggers top 100**: The top 100 triggers are shown here.
- **Audit**: We can see who changed what on our Zabbix server here. This is a great way to see which colleague locked you out by accident or whether it was on purpose.
- **Action log**: We can see a list of actions that have been taken due to triggers going to a **problem** or **ok** state.
- **Notifications**: On this page, we can see a list of notifications sent to our users.

Next, we have the **Configuration** category:

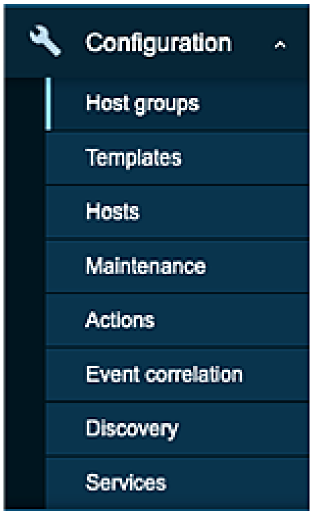


Figure 1.21 – The Configuration section of the sidebar

The **Configuration** tab contains the following pages:

- **Host groups**: We configure our host groups here; for instance, a group for all *Linux servers*.
- **Templates**: This is where we configure our templates that we can use to monitor hosts from the Zabbix server.
- **Hosts**: Another **Hosts** tab, but this time it is not for checking the data. This is where we add and configure host settings.
- **Maintenance**: In Zabbix, we have the availability to set maintenance periods; this way, triggers won't disturb you while you take something offline for maintenance, for example.
- **Actions**: Remember how I mentioned we can configure actions for when a trigger changes state? This is where we configure those actions.
- **Event correlation**: We can correlate problems here to reduce noise or prevent event storms. This is achieved by closing new or old problems when they correlate to other problems.
- **Discovery**: This is where we configure Zabbix discovery for automatic host creation.
- **Services**: You'll configure your services with SLA information here.

Finally, we have the **Administration** category:

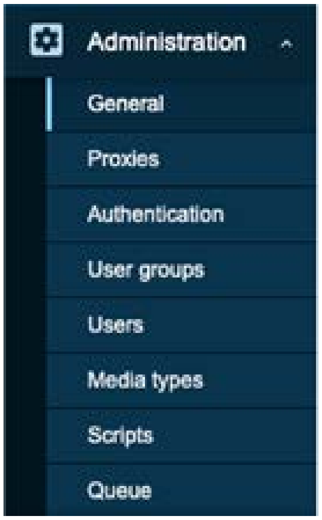


Figure 1.22 – The Administration section of the sidebar

The **Administration** tab contains the following pages:

- **General**: The general page contains our Zabbix server configuration. Settings ranging from housekeeper to frontend theme are found here. Don't forget to turn on the dark theme to save your eyes from strain! It's important to keep your eyes protected when working with computer screens a lot. Do note that we set defaults for all users but most settings can also be customized per user.
- **Proxies**: This is where we configure proxies that should be connected to this Zabbix server.
- **Authentication**: We can find our authentication settings here, such as LDAP, SAML, and HTTP.
- **User groups**: This is where we configure user groups and the permissions for these user groups.
- **Users**: Add users on this page.
- **Media types**: There are several media types pre-configured in Zabbix, which you'll find here already. We can also add custom media types.
- **Scripts**: This is where we can add custom scripts, for extending Zabbix features.
- **Queue**: View your Zabbix server queue here. Items might be stuck in a queue due to data collection issues.

Tip

When using Zabbix authentication such as HTTP, LDAP, or SAML, we still need to create our users internally. Configure your users to match your authentication method's username in Zabbix and use the authentication method for password management.

Creating user groups

To log in to the Zabbix frontend, we are going to need users. Right now, we are logged in with the default user, which is logical because we need a user to create users. This isn't a safe setup though, because we don't want to keep using `zabbix` as a password. So, we are going to learn how to create new users and group them accordingly.

It's important to choose how you want to manage users in Zabbix before setting up user accounts. If you want to use something such as LDAP or SAML, it's a smart idea to make the choice to use one of those authentication methods right away, so you won't have any migration trouble.

Getting ready

Now that we know how the Zabbix UI is structured and we know how to navigate it, we can start doing some actual configuration. We'll start out by creating some user groups to get familiar with the process and start using them. This way, our Zabbix setup gets not only more structured but also more secure.

To get started with this, we'll need the server from the last recipes and the knowledge we've acquired there to navigate to the correct configuration sections.

Looking at the following figure, we can see how our example company, **Cloud Hoster**, is set up. We will create the users seen in the diagram to create a structured and solid user setup:

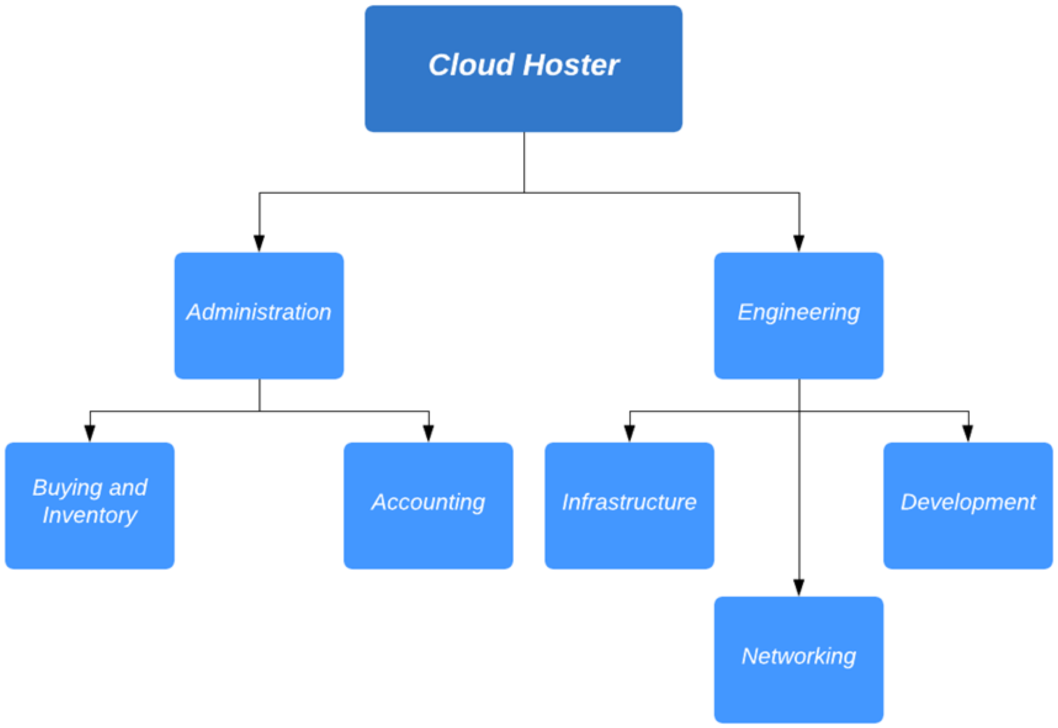


Figure 1.23 – Cloud Hoster department diagram

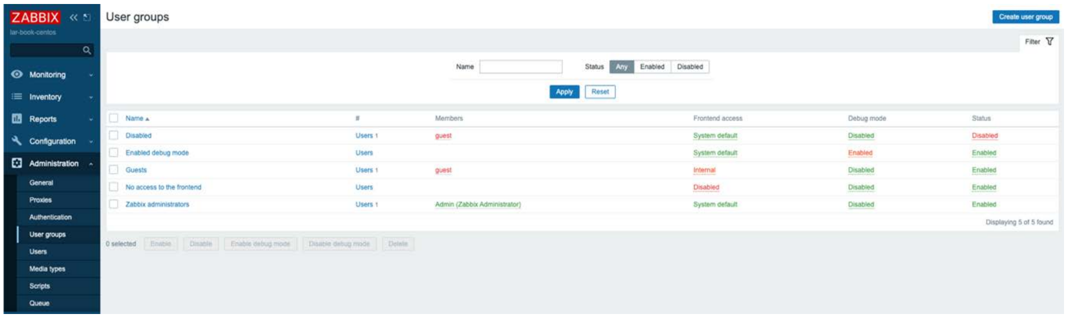
So, **Cloud Hoster** has some departments that need access to the Zabbix frontend and others who don't need it at all. Let's say we want to give the following departments access to the Zabbix frontend:

- **Networking**: To configure and monitor their network devices
- **Infrastructure**: To configure and monitor their Linux servers
- **Buying and Inventory**: To look at inventory information and compare it to other internal tools

How to do it...

Let's get started on creating these three groups in our Zabbix UI:

1. To do this, navigate to **Administration** | **User groups**, which will show you the following page:

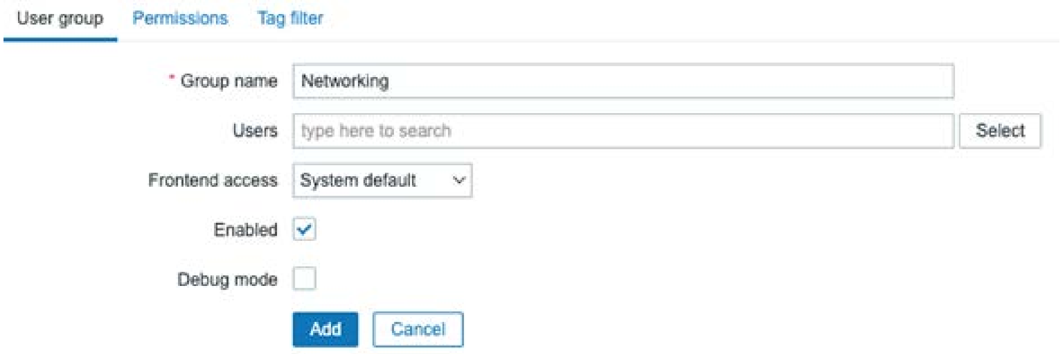


Name ▲	#	Members	Frontend access	Debug mode	Status
Disabled	Users 1	guest	System default	Disabled	Disabled
Enabled debug mode	Users		System default	Enabled	Enabled
Guests	Users 1	guest	Internal	Disabled	Enabled
No access to the frontend	Users		Disabled	Disabled	Enabled
Zabbix administrators	Users 1	Admin (Zabbix Administrator)	System default	Disabled	Enabled

Displaying 5 of 5 found

Figure 1.24 – The Zabbix User groups window

2. Now, let's start by creating the **Networking** group by clicking **Create user group** on the top-right corner. This will bring you to the following screen:

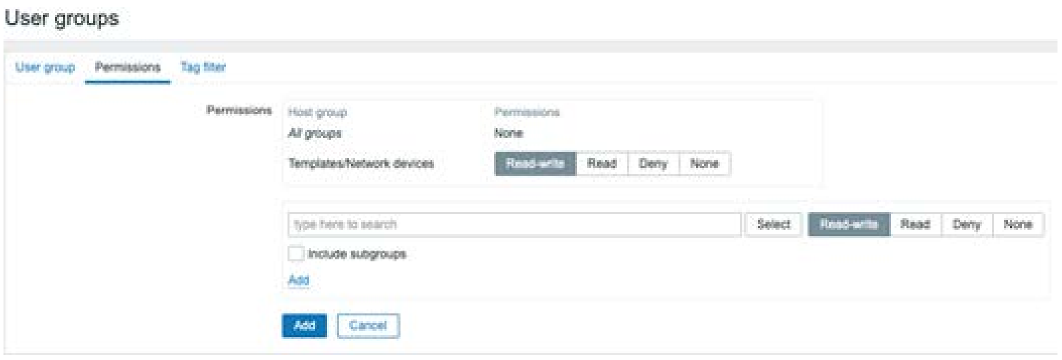


* Group name	Networking
Users	type here to search
Frontend access	System default
Enabled	✓
Debug mode	

Figure 1.25 – The Zabbix User groups configuration window

We will need to fill in the information, starting with **Group name**, which of course will be `Networking`. There are no users for this group yet, so we'll skip that one. **Frontend access** is the option to provide us with authentication; if you select **LDAP** here, LDAP authentication will be used for authenticating. We will keep it as **System default**.

Now, let's move on to the next page, named **Permissions**:


User groups

User group | Permissions | Tag filter

Permissions	Host group	Permissions
	All groups	None
	Templates/Network devices	Read-write

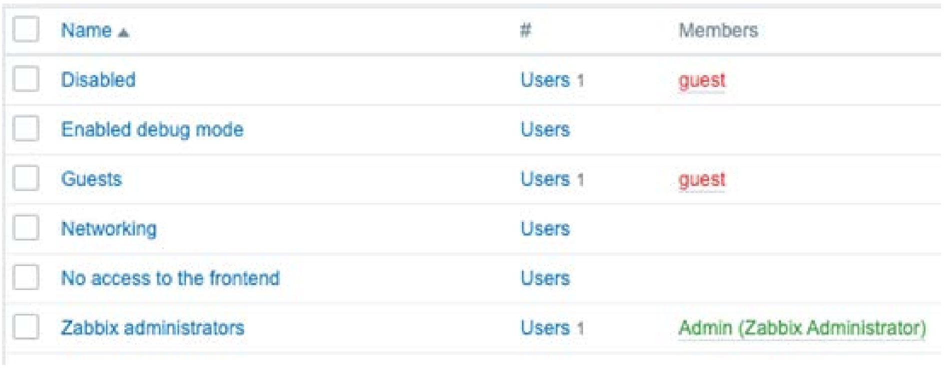
Figure 1.26 – The Zabbix User groups Permissions configuration window

Here, we can specify what host groups our group will have access to. There's a default host group for **Networking** already, which we will use in this example. Press **Select** to take you to a pop-up window with host groups available. Select **Templates/ Network devices** here and it'll take you back to the previous window, with the group filled in. Select **Read-write** and press the small-text **Add** button to add these permissions.

We won't be adding anything else, so press the bigger blue **Add** button to finish creating this host group.

> **Tip**
>
> Zabbix has a lot of screens with two **Add** buttons. Always make sure to press the smaller text one to add your options and the bigger one to complete your adding of an item – in this case, a host group. Be prepared to press the wrong button a few times before learning the way Zabbix has implemented this.

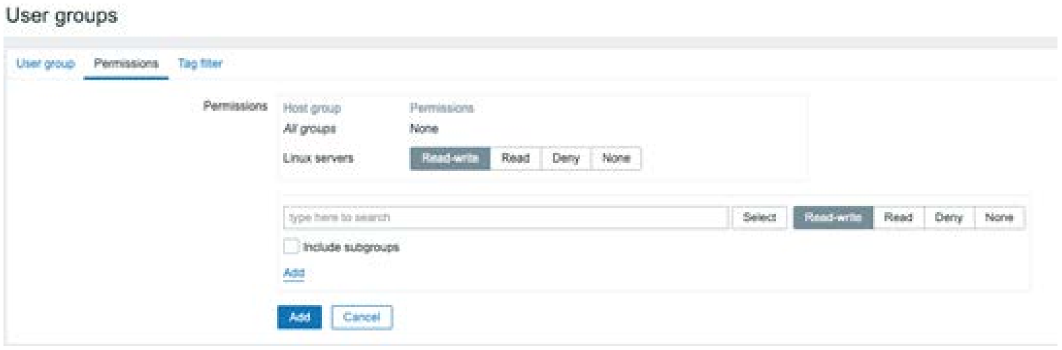
Now we will have a new host group called **Networking** that is only allowed to read and write to the **Templates/Network devices** host group:



Name ▲	#	Members
Disabled	Users 1	guest
Enabled debug mode	Users	
Guests	Users 1	guest
Networking	Users	
No access to the frontend	Users	
Zabbix administrators	Users 1	Admin (Zabbix Administrator)

Figure 1.27 – The Zabbix User groups window

3. Let's repeat this process for the **Infrastructure** host group, except instead of adding the **Templates/Network devices** host group, we'll add the **Linux servers** host group, like this:



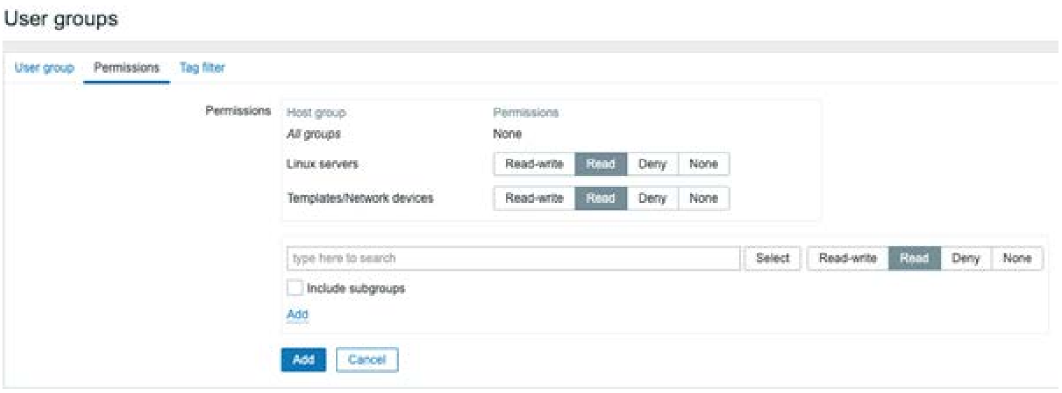
User groups

Permissions

Host group	Permissions
All groups	None
Linux servers	Read-write

Figure 1.28 – The Zabbix User groups Permissions configuration window with one host group

4. Then, to add **Buying and Inventory**, we'll do something differently. We'll repeat the process we've just done except for the part with the permissions. We want **Buying and Inventory** to be able to read our inventory data, but we don't want them to actually change our host configuration. Add both **Templates/Network devices** and **Linux servers** to the group, but with only **Read** permissions like this:


User groups

User group | Permissions | Tag filter

Permissions

Host group	Permissions
All groups	None
Linux servers	Read-write [✓ Read] Deny None
Templates/Network devices	Read-write [✓ Read] Deny None

type here to search | Select | Read-write [✓ Read] Deny None

Include subgroups

Add

Add | Cancel

Figure 1.29 – The Zabbix User groups Permissions configuration window with two host groups

Congratulations! Finishing this means you've ended up with three different host groups and we can continue on to create our first new users! Let's get to it.

Creating your first users

With our newly created user groups, we've taken our first step toward a more structured and secure Zabbix setup. The next step is to actually assign some users to the newly created user groups to make sure they are assigned our new user permissions from the group.

Getting ready

To get started, we'll need the server and the newly created user groups from the last recipe. So, let's start the configuration.

Now we know there are three departments in the company called **Cloud Hoster** that are going to use our Zabbix installation. We've created host groups for them but there are also users in those departments that actually want to use our installation. Let's meet them:

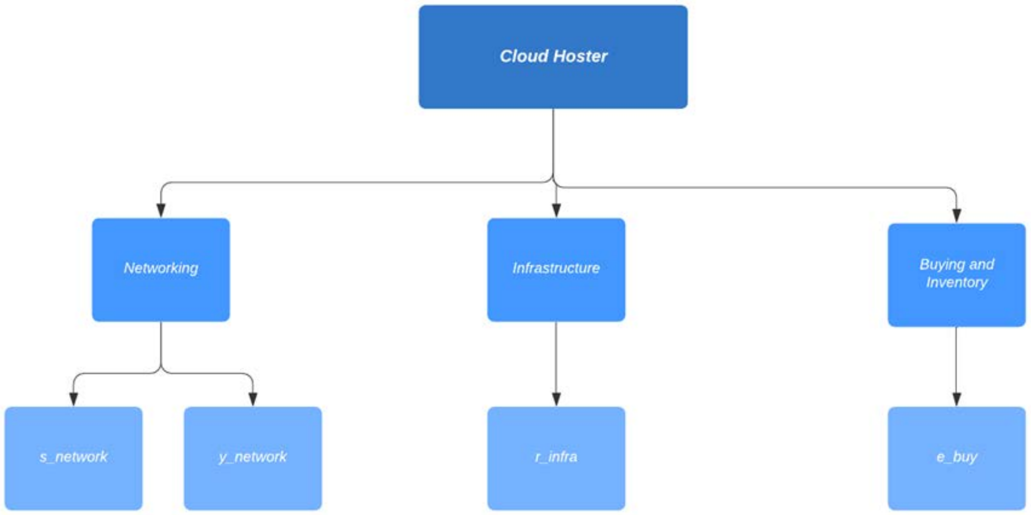


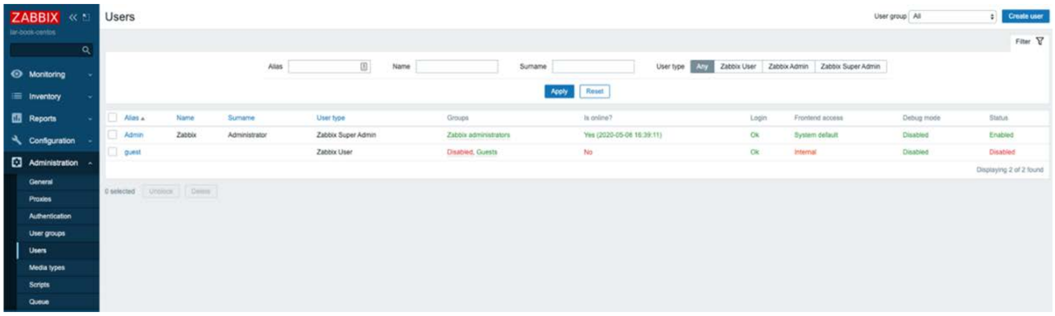
Figure 1.30 – Cloud Hoster users diagram

These are the users we need to configure for **Cloud Hoster** to use.

How to do it...

Let's start creating the users. We will start with our **Networking** department:

1. Navigate to **Administration** | **Users**, which will bring you to this page:

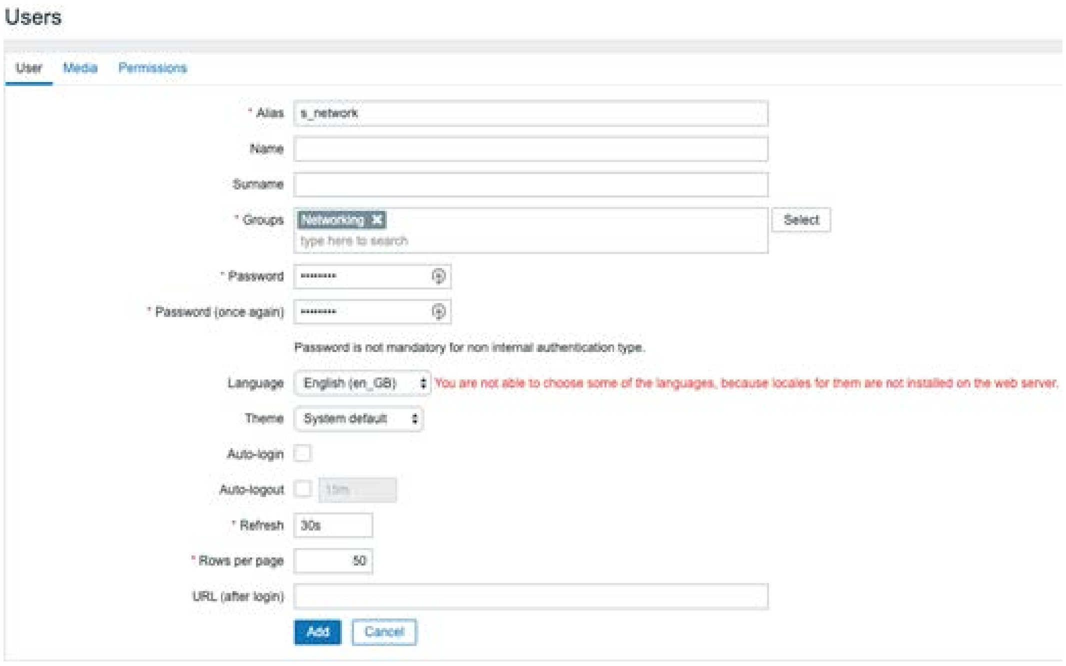


Alias ▲	Name	Surname	User type	Groups	Is online?	Login	Frontend access	Debug mode	Status
Admin	Zabbix	Administrator	Zabbix Super Admin	Zabbix administrators	Yes (2020-05-06 16:39:11)	Ok	System default	Disabled	Enabled
guest			Zabbix User	Disabled, Guests	No	Ok	Internal	Disabled	Disabled

Displaying 2 of 2 found

Figure 1.31 – The Zabbix Users window

This is where all the user creation magic is happening, as we will be managing all of our users from this page. To create our first **Networking** department user named **s_network**, click the **Create user** button on the top-right corner, bringing us to the following screen:



Users

User | Media | Permissions

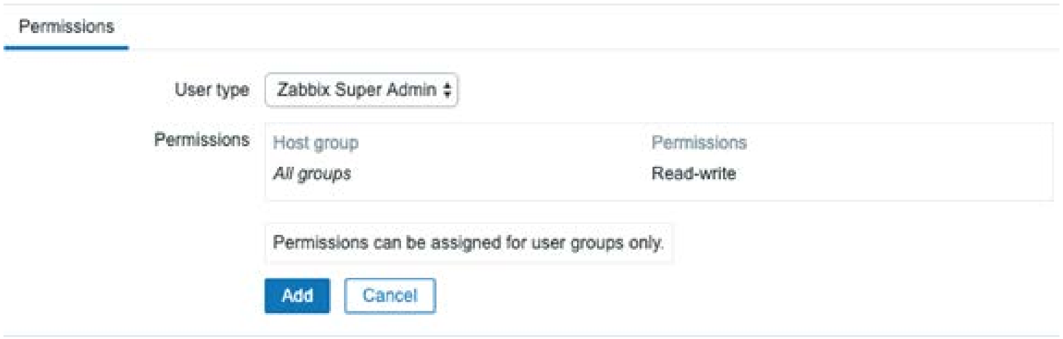
* Alias	s_network
Name	
Surname	
* Groups	Networking ✕ type here to search — Select
* Password	••••••••
* Password (once again)	••••••••
	Password is not mandatory for non internal authentication type.
Language	English (en_GB) — You are not able to choose some of the languages, because locales for them are not installed on the web server.
Theme	System default
Auto-login	☐
Auto-logout	☐ 15m
* Refresh	30s
* Rows per page	50
URL (after login)	

Add Cancel

Figure 1.32 – The Zabbix Users configuration window

Fill out **Alias** to provide us with the username this user will be using, which will be **s_network**. Also, it's important to add this user to the group we have just created to give our user the right permissions. Click **Select** and pick our group called **Networking**.

Last but not least, set a secure password in the **Password** fields; don't forget it because we will be using it later. After this, move on to the **Permissions** tab as we won't be configuring **Media** just yet:



User type: Zabbix Super Admin

Permissions:

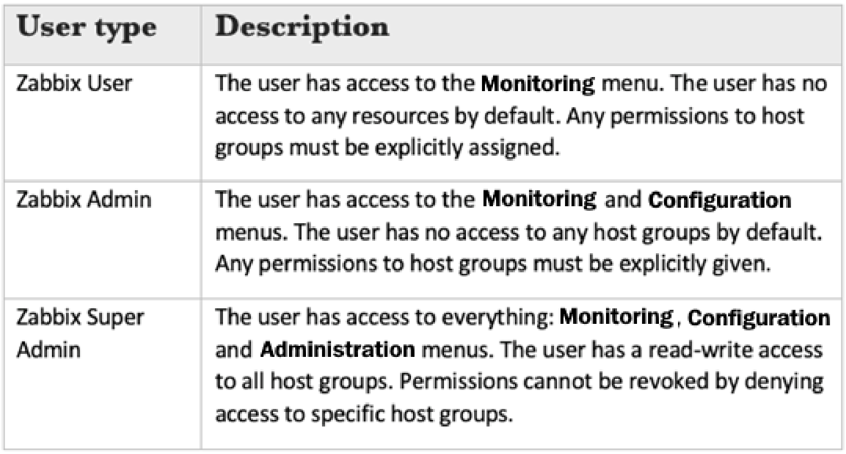
Host group	Permissions
All groups	Read-write

Permissions can be assigned for user groups only.

Figure 1.33 – The Zabbix Users Permissions configuration window

Select the **User type** option named **Zabbix Super Admin** here. This will enable our user to access all tabs and see information about all host groups in the Zabbix server.

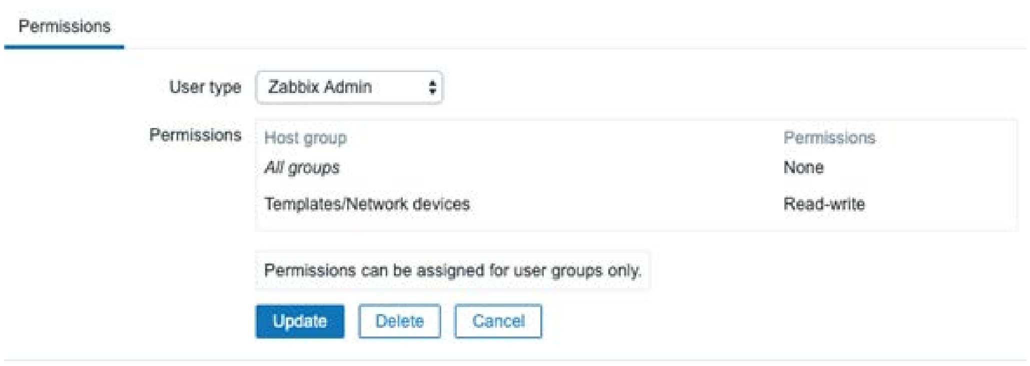
The following **User type** options are available in Zabbix:



User type	Description
Zabbix User	The user has access to the **Monitoring** menu. The user has no access to any resources by default. Any permissions to host groups must be explicitly assigned.
Zabbix Admin	The user has access to the **Monitoring** and **Configuration** menus. The user has no access to any host groups by default. Any permissions to host groups must be explicitly given.
Zabbix Super Admin	The user has access to everything: **Monitoring**, **Configuration** and **Administration** menus. The user has a read-write access to all host groups. Permissions cannot be revoked by denying access to specific host groups.

Figure 1.34 – A table detailing the different Zabbix user types

2. Let's repeat the previous steps for the user named **y_network** but in the **Permissions** tab, select the **Zabbix Admin** option like this:



User type: Zabbix Admin

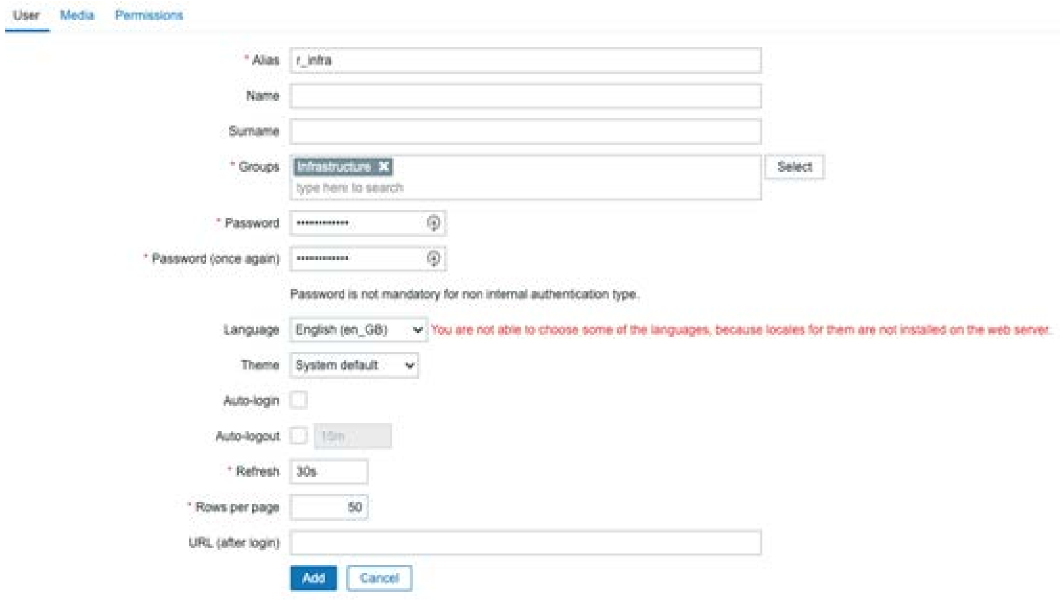
Permissions:

Host group	Permissions
All groups	None
Templates/Network devices	Read-write

Permissions can be assigned for user groups only.

Figure 1.35 – The Zabbix Users Permissions configuration window

After creating these two users, let's move on to create the infrastructure user, **r_infra**. Repeat the steps we took for **s_network**, changing the alias, of course. Then, add this user to the group and give our user the right permissions. Click **Select** and pick our group called **Infrastructure**. It will look like this:

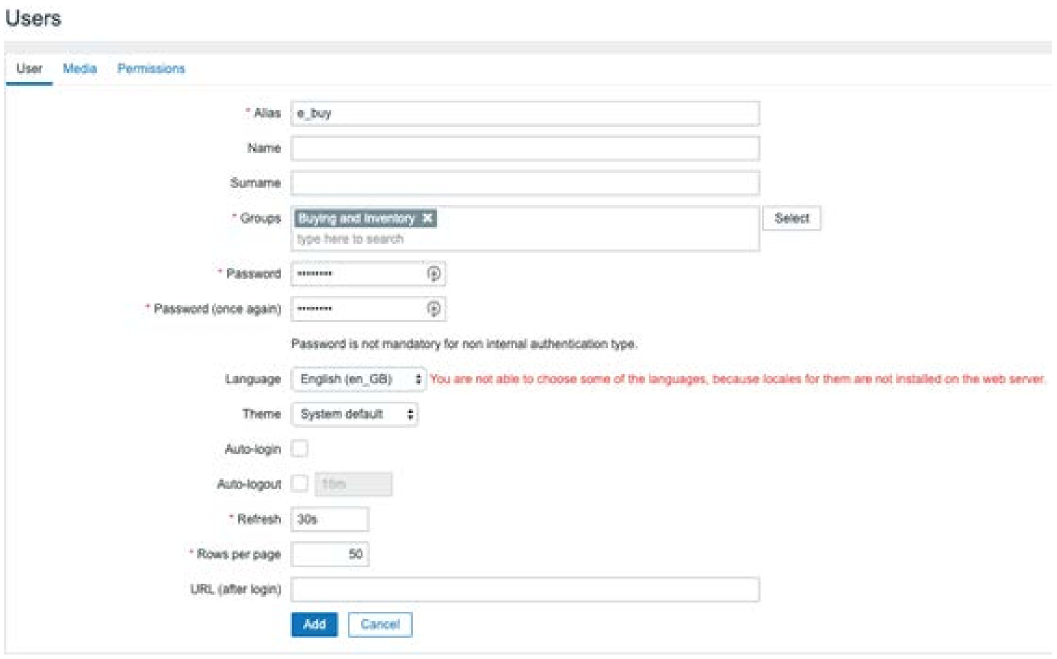


* Alias	r_infra
Name	
Surname	
* Groups	Infrastructure ✕ (type here to search) Select
* Password	•••••••••••••
* Password (once again)	•••••••••••••
	Password is not mandatory for non internal authentication type.
Language	English (en_GB) You are not able to choose some of the languages, because locales for them are not installed on the web server.
Theme	System default
Auto-login	☐
Auto-logout	☐ 15m
* Refresh	30s
* Rows per page	50
URL (after login)	

Figure 1.36 – The Zabbix User configuration window for r_infra

Lastly, make this user another **Zabbix Super Admin**.

3. Now, for our last user, let's repeat our steps again, changing the alias and the group in the **User** tab like this:



Users

User | Media | Permissions

* Alias	e_buy
Name	
Surname	
* Groups	Buying and Inventory ✕ (type here to search) [Select]
* Password	••••••••
* Password (once again)	••••••••
	Password is not mandatory for non internal authentication type.
Language	English (en_GB) — You are not able to choose some of the languages, because locales for them are not installed on the web server.
Theme	System default
Auto-login	☐
Auto-logout	☐ 15m
* Refresh	30s
* Rows per page	50
URL (after login)	

[Add] [Cancel]

Figure 1.37 – The Zabbix User configuration window for e_buy

You don't need to change anything in the **Permissions** tab for this user as the default setting is **Zabbix User**. This is what we want for this user so that they can only access data and not change anything.

When you're done, you'll end up with the following:

- **s_network**: A user with access to the **Networking** user group permissions and that is **Zabbix Super Admin**.
- **y_network**: A user with access to the **Networking** user group permissions and that is **Zabbix Admin**.
- **r_infra**: A user with access to the **Infrastructure** user group permissions and that is **Zabbix Super Admin**.
- **e_buy**: A user with access to the **Buying and Inventory** user group permissions and that is **Zabbix User**.

Advanced user authentication with SAML

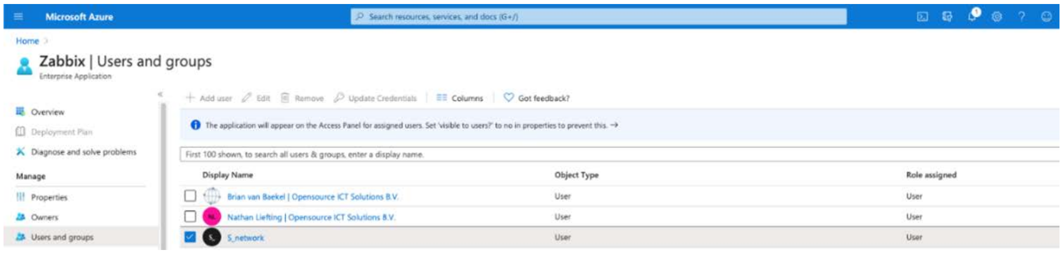
New to Zabbix 5.0 is SAML authentication, a widely used form of authentication in the IT world. We'll be using this as a form of managing passwords for our Zabbix users. Please note that if you've worked with Zabbix before and you've configured LDAP, SAML, like LDAP, allows user authentication with passwords. You still have to create users with their permissions.

Getting ready

To get started with SAML authentication, we will need our configured Zabbix server from the previous recipe. It's important that we have all the configured users from the previous recipe. We will also need something to authenticate with SAML. We will be using Azure **Active Directory** (**AD**) SAML.

Make sure to set up users in your (Azure) AD before continuing with this recipe. You can use your existing AD users for authentication, so you can use this recipe with your existing AD setup.

We will be using the **s_network** user as an example:



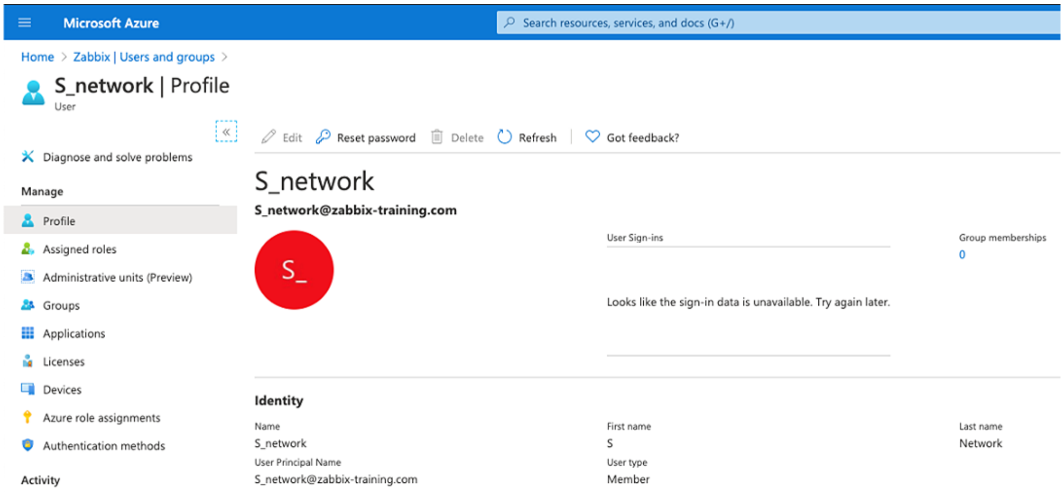
The application will appear on the Access Panel for assigned users. Set 'visible to users?' to no in properties to prevent this. →

First 100 shown, to search all users & groups, enter a display name.

	Display Name	Object Type	Role assigned
☐	Brian van Baekel | Opensource ICT Solutions B.V.	User	User
☐	Nathan Liefting | Opensource ICT Solutions B.V.	User	User
✓	S_network	User	User

Figure 1.38 – The Azure Users and groups window

These are our user details:



Microsoft Azure

Search resources, services, and docs (G+/)

Home > Zabbix | Users and groups >

S_network | Profile

User

Diagnose and solve problems

Manage

Profile

Assigned roles

Administrative units (Preview)

Groups

Applications

Licenses

Devices

Azure role assignments

Authentication methods

Activity

Edit Reset password Delete Refresh Got feedback?

S_network

S_network@zabbix-training.com

S_

User Sign-ins

Looks like the sign-in data is unavailable. Try again later.

Group memberships

0

Identity

Name	First name	Last name
S_network	S	Network
User Principal Name	**User type**	
S_network@zabbix-training.com	Member	

Figure 1.39 – The Azure user details window

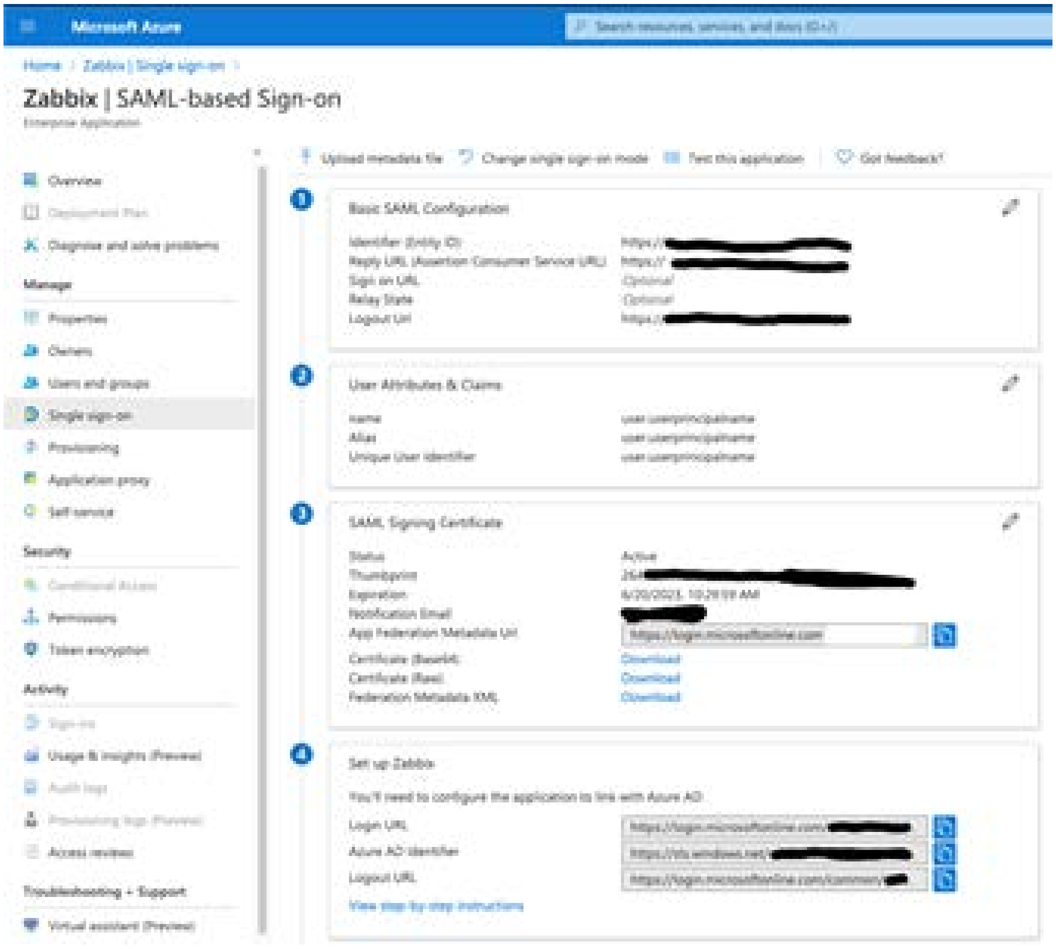
To set up SAML, retrieve your SAML settings from your AD or another SAML provider. To work with Zabbix, we will need the following:

- IdP entity ID
- SSO service URL
- SLO service URL
- Username attribute
- SP entity ID
- SP name ID format

How to do it...

Now that we have our Azure AD ready, let's see how we can configure SAML using our setup:

1. In our Azure AD setup, we can find this info under **Active Directory | Single sign-on**:



Zabbix | SAML-based Sign-on
Enterprise Application

1 Basic SAML Configuration

Identifier (Entity ID)	https:// [redacted]
Reply URL (Assertion Consumer Service URL)	https:// [redacted]
Sign on URL	Optional
Relay State	Optional
Logout Url	https:// [redacted]

2 User Attributes & Claims

name	user.userprincipalname
Alias	user.userprincipalname
Unique User Identifier	user.userprincipalname

3 SAML Signing Certificate

Status	Active
Thumbprint	264 [redacted]
Expiration	6/20/2023, 10:29:59 AM
Notification Email	[redacted]
App Federation Metadata Url	https://login.microsoftonline.com
Certificate (Base64)	Download
Certificate (Raw)	Download
Federation Metadata XML	Download

4 Set up Zabbix

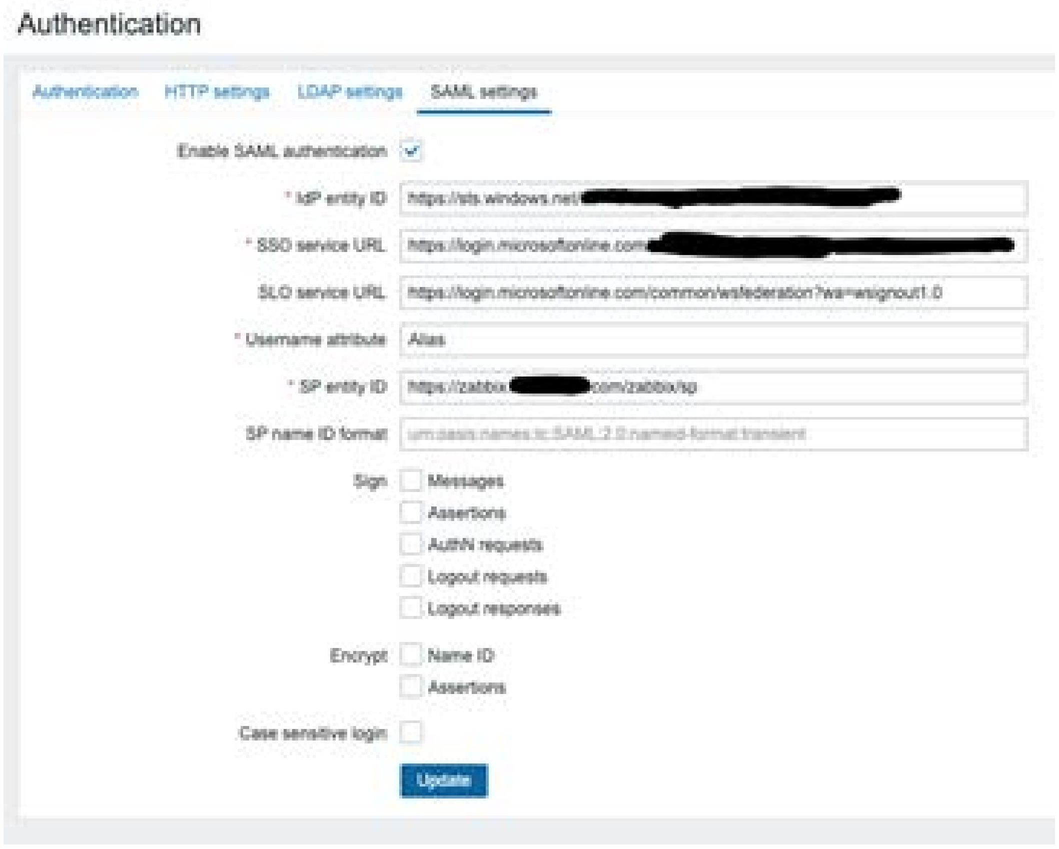
You'll need to configure the application to link with Azure AD.

Login URL	https://login.microsoftonline.com/ [redacted]
Azure AD Identifier	https://sts.windows.net/ [redacted]
Logout URL	https://login.microsoftonline.com/common/ [redacted]

View step-by-step instructions

Figure 1.40 – The Azure SAML settings window

2. Fill this information into your Zabbix instance as follows:


Authentication

Authentication | HTTP settings | LDAP settings | SAML settings

Enable SAML authentication	✓
* IdP entity ID	https://sts.windows.net/[redacted]
* SSO service URL	https://login.microsoftonline.com[redacted]
SLO service URL	https://login.microsoftonline.com/common/wsfederation?wa=wsignout1.0
* Username attribute	Alias
* SP entity ID	https://zabbix[redacted].com/zabbix/sp
SP name ID format	urn:oasis:names:tc:SAML:2.0:nameid-format:transient
Sign	☐ Messages ☐ Assertions ☐ AuthN requests ☐ Logout requests ☐ Logout responses
Encrypt	☐ Name ID ☐ Assertions
Case sensitive login	☐

Figure 1.41 – The Zabbix SAML settings

3. After following these steps, it should now be possible to log in with your user configured in Zabbix and use the password set in Azure AD for this:

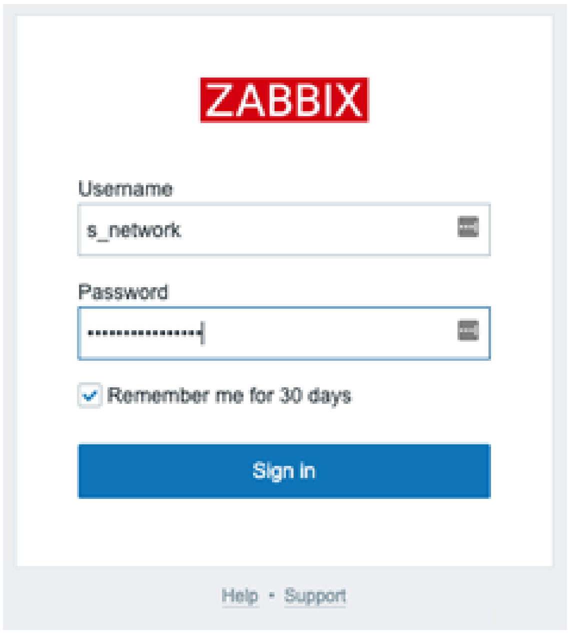


Figure 1.42 – The Zabbix login window

How it works...

Zabbix advanced user authentication is used to centralize password management. While we are not able to actually assign user groups and permissions to users via this setup, we can use it for password management.

This way, we can make sure it is easier for users to keep their passwords centralized in medium to big office environments:

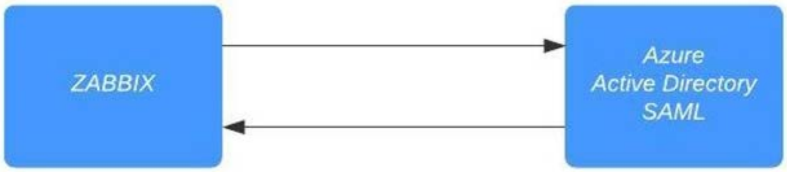


Figure 1.43 – Zabbix SAML authentication diagram

Zabbix communicates with our Azure AD SAML component when we press the login button. The user is then authenticated against your Azure AD user and a confirmation is sent back to the Zabbix server. Congratulations, you are now logged in to your Zabbix server!

There's more...

We can do this kind of authentication not only with SAML but also with HTTP and LDAP. This way, you can choose the right form of advanced authentication for your organization.

Check out the Zabbix documentation for more information on the different forms of authentication: `https://www.zabbix.com/documentation/current/manual/web_interface/frontend_sections/administration/authentication`.

It's also possible to work with an identity provider such as Okta, so your options aren't limited to Azure AD. As long as it supports SAML, you can use it to authenticate against your Zabbix server.

2
Setting Up Zabbix Monitoring

Zabbix is built to be flexible and should be able to monitor just about anything you could ever require. In this chapter, we will learn more about working with Zabbix to build a lot of different options for monitoring. We'll go over them recipe by recipe, so you'll end up with a solid understanding of how they work. We'll cover the following recipes on the different monitoring types:

- Setting up Zabbix agent monitoring
- Working with SNMP monitoring
- Creating Zabbix simple checks and Zabbix trapper
- Working with calculated and dependent items
- Creating external checks
- Setting up JMX monitoring
- Setting up database monitoring
- Setting up HTTP agent monitoring
- Using Zabbix preprocessing to alter item values

Technical requirements

We will need a Zabbix server capable of performing monitoring, with the following requirements:

- A server with Zabbix server installed on a Linux distribution of your choice, such as CentOS or Ubuntu, but a distribution such as Debian or anything else will suit you just as well
- MariaDB set up to work with your Zabbix server
- NGINX or Apache set up to serve the Zabbix frontend

I'll be using the same server as we used in the previous chapter, but any Zabbix server should do.

Setting up Zabbix agent 2 monitoring

With the release of Zabbix 5, Zabbix also officially started support for the new Zabbix agent 2. Zabbix agent 2 brings some major improvements and is even written in another coding language, which is Golang instead of C. In this recipe, we will be exploring how to work with Zabbix agent 2 and explore some of the new features introduced by it.

You'll also need a Linux distribution of your choice running Zabbix agent 2.

Getting ready

To get started with Zabbix agent 2, all we need to do is install it to a (Linux) host that we want to monitor. Make sure you have an empty Linux (CentOS 8) host ready to monitor.

How to do it

Let's see how to install Zabbix agent 2 and then move on to actually working with it.

Installing Zabbix agent 2

Let's start by installing Zabbix agent 2 on the Linux host we want to monitor. I'll be using a CentOS 8 machine:

1. Issue the following command to add the repository:

   For RHEL-based systems:

   ```
   rpm -Uvh https://repo.zabbix.com/zabbix/5.0/rhel/8/
   x86_64/zabbix-release-5.0-1.el8.noarch.rpm
   ```

For Debian-based systems:

```
wget https://repo.zabbix.com/zabbix/5.0/debian/pool/
main/z/zabbix-release/zabbix-release_5.0-1+buster_all.deb

dpkg -i zabbix-release_5.0-1+buster_all.deb
```

2. Then issue the following command to install Zabbix Agent 2:

   For RHEL-based systems:

   ```
   dnf -y install zabbix-agent2
   ```

   For Debian-based systems:

   ```
   apt install zabbix-agent2
   ```

Congratulations, Zabbix agent 2 is now installed and ready to use.

Using a Zabbix agent in Passive mode

Let's start by building a Zabbix agent with passive checks:

1. After installing Zabbix agent 2, let's open the Zabbix agent configuration file for editing:

   ```
   vim /etc/zabbix/zabbix_agent2.conf
   ```

   In this file, we edit all the Zabbix agent configuration values we could need from the server side.

2. Let's start by editing the following values:

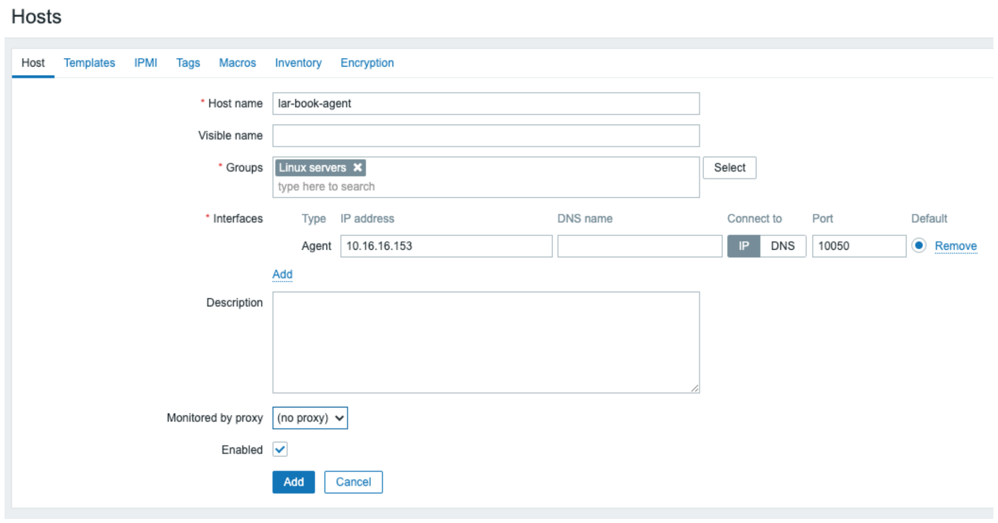
   ```
   Server=127.0.0.1
   Hostname=Zabbix server
   ```

3. Change `Server` to the IP of the Zabbix server that will monitor this passive agent. Change `Hostname` to the hostname of the monitored server.
4. Now restart the Zabbix agent 2 process:

   ```
   systemctl restart zabbix-agent2.service
   ```

5. Now move to the frontend of your Zabbix server and add this host for monitoring.
6. Go to **Configuration | Hosts** in your Zabbix frontend and click **Create host** in the top-right corner.

7. To create this host in our Zabbix server, we need to fill in the values seen in the following screenshot:


Hosts

Host | Templates | IPMI | Tags | Macros | Inventory | Encryption

* Host name: lar-book-agent

Visible name:

* Groups: Linux servers ✖ (type here to search) Select

* Interfaces:

Type	IP address	DNS name	Connect to	Port	Default
Agent	10.16.16.153		IP	10050	● Remove

Add

Description:

Monitored by proxy: (no proxy)

Enabled: ✓

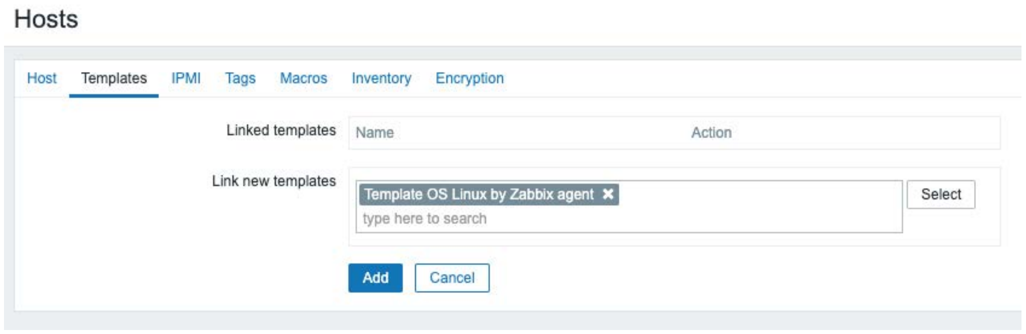
Add | Cancel

Figure 2.1 – The Zabbix host creation page for host lar-book-agent

It's important to add the following:

- **Host name**: To identify this host.
- **Groups**: To logically group hosts.
- **Interfaces**: To monitor this host on a specific interface. No interface means no communication.

8. It is also important to add a template to this host. As this is a Linux server monitored by a Zabbix agent, let's add the correct out-of-the-box template as shown in the following screenshot:


Hosts

Host | Templates | IPMI | Tags | Macros | Inventory | Encryption

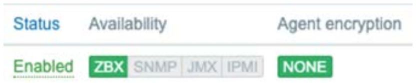
Linked templates: Name | Action

Link new templates: Template OS Linux by Zabbix agent ✖ (type here to search) Select

Add | Cancel

Figure 2.2 – The Zabbix host template page for host lar-book-agent

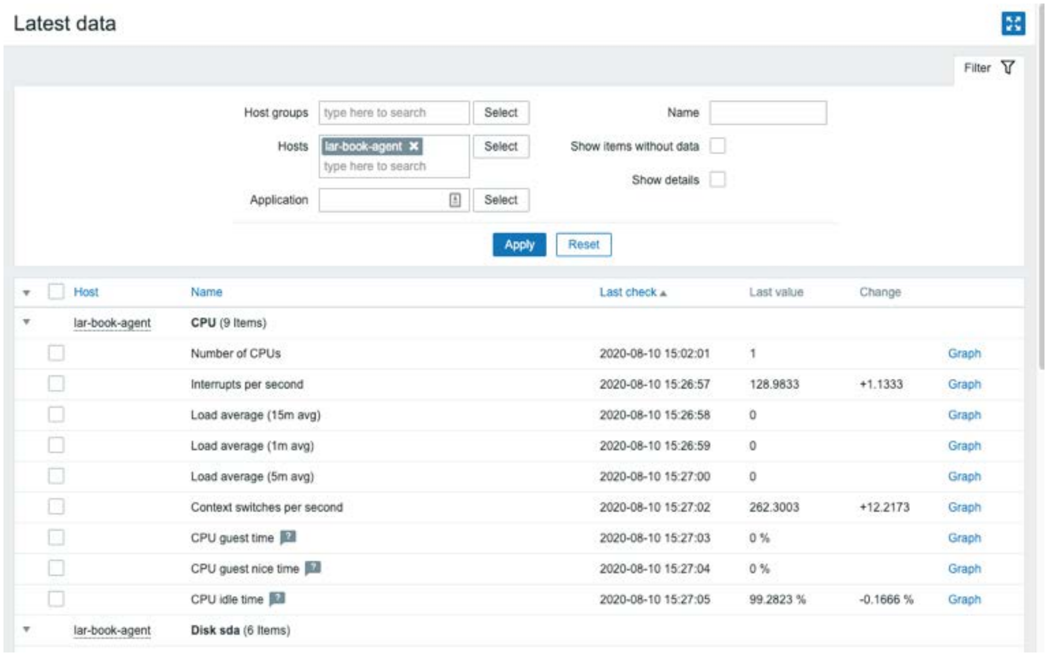
9. Click the **Add** button and you're done creating this agent host. Now that you've got this host, make sure the **ZBX** icon turns green, indicating that this host is up and being monitored with the passive Zabbix agent:



Status	Availability	Agent encryption
Enabled	ZBX SNMP JMX IPMI	NONE

Figure 2.3 – The Zabbix configuration hosts page, host lar-book-agent

10. You can now see the values received on this host by going to **Monitoring | Hosts** and checking the **Latest data** button. Please note that the values could take a while to show up:



Latest data

Host groups	type here to search	Name	
Hosts	lar-book-agent	Show items without data	☐
		Show details	☐
Application			

Host	Name	Last check ▲	Last value	Change	
lar-book-agent	CPU (9 Items)				
	Number of CPUs	2020-08-10 15:02:01	1		Graph
	Interrupts per second	2020-08-10 15:26:57	128.9833	+1.1333	Graph
	Load average (15m avg)	2020-08-10 15:26:58	0		Graph
	Load average (1m avg)	2020-08-10 15:26:59	0		Graph
	Load average (5m avg)	2020-08-10 15:27:00	0		Graph
	Context switches per second	2020-08-10 15:27:02	262.3003	+12.2173	Graph
	CPU guest time	2020-08-10 15:27:03	0 %		Graph
	CPU guest nice time	2020-08-10 15:27:04	0 %		Graph
	CPU idle time	2020-08-10 15:27:05	99.2823 %	-0.1666 %	Graph
lar-book-agent	Disk sda (6 Items)				

Figure 2.4 – The Zabbix latest data page for host lar-book-agent

Using a Zabbix agent in Active mode

Now let's check out how to configure the Zabbix agent with active checks. We need to change some values on the monitored Linux server host side:

1. Start by executing the following command:

```
vim /etc/zabbix/zabbix_agent2.conf
```

2. Now let's edit the following value to change this host to an active agent:

```
ServerActive=127.0.0.1
```

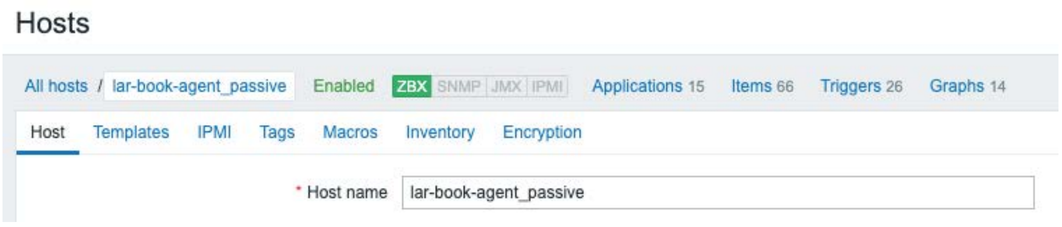
3. Change `ServerActive` to the IP of the Zabbix server that will monitor this passive agent and also change `Hostname` to `lar-book-agent`:

```
Hostname=lar-book-agent
```

4. Now restart the Zabbix agent 2 process:

```
systemctl restart zabbix-agent2.service
```

5. Now move to the frontend of your Zabbix server and let's add another host with a template to do active checks instead of passive ones.
6. First, let's rename our passive host. To do that, go to **Configuration | Hosts** in your Zabbix frontend and click the host we just created. Change **Host name** as follows:


Hosts
All hosts / lar-book-agent_passive
Enabled
ZBX SNMP JMX IPMI
Applications 15
Items 66
Triggers 26
Graphs 14
Host Templates IPMI Tags Macros Inventory Encryption

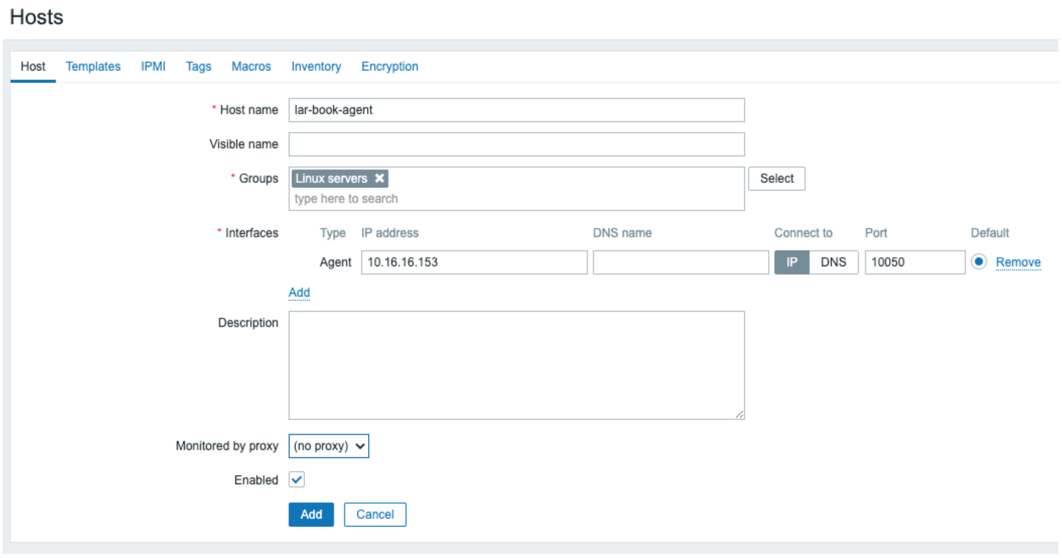
* Host name: lar-book-agent_passive

Figure 2.5 – The Zabbix host configuration page for host lar-book-agent_passive

We are doing this because for an active Zabbix agent, the hostname in the file needs to match our Zabbix server.

7. Go to **Configuration | Hosts** in your Zabbix frontend and click **Create host** in the top-right corner.

8. Now let's create the host as follows:

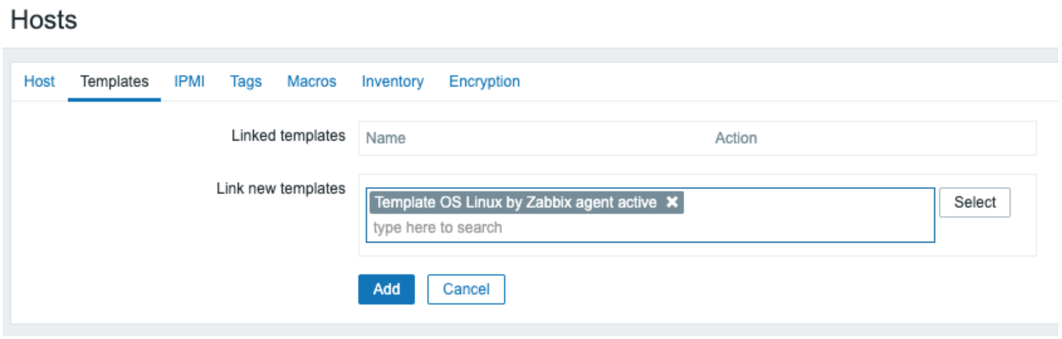

Hosts

Host | Templates | IPMI | Tags | Macros | Inventory | Encryption

* Host name	lar-book-agent
Visible name	
* Groups	Linux servers ✖
* Interfaces	Type: Agent; IP address: 10.16.16.153; DNS name: ; Connect to: IP; Port: 10050; Default: ●
Description	
Monitored by proxy	(no proxy)
Enabled	✓

Figure 2.6 – The Zabbix host configuration page for host lar-book-agent

9. Also, make sure to add the correct template, named `Template OS Linux by Zabbix agent active`:


Hosts

Host | Templates | IPMI | Tags | Macros | Inventory | Encryption

Linked templates	Name / Action
Link new templates	Template OS Linux by Zabbix agent active ✖

Figure 2.7 – The Zabbix host template page for host lar-book-agent

Please note that the **ZBX** icon won't turn green for an active agent. But when we navigate to **Monitoring | Hosts** and check **Latest data**, we can see our active data coming in.

Tip

As you might have noticed just now, a Zabbix agent can run in both passive and active mode. Keep this in mind when creating your Zabbix agent templates, as you might want to combine the check types.

How it works...

Now that we have configured our Zabbix agents and know how they should be set up, let's see how the different modes work.

Passive agent

The **passive agent** works by collecting data from our host with the Zabbix agent. Every time an item on our host reaches its *interval*, the Zabbix server asks the Zabbix agent what the value is now:

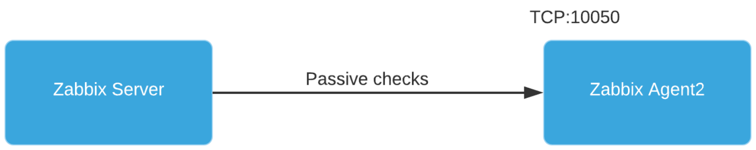


Figure 2.8 – Communication diagram between server and passive agent

Active agent

The **active agent** works by sending data from the Zabbix agent to Zabbix server. Every time an item on our agent reaches its *interval*, the agent will send the value to our server. We can also use this to send a notification to our server faster when something goes wrong:

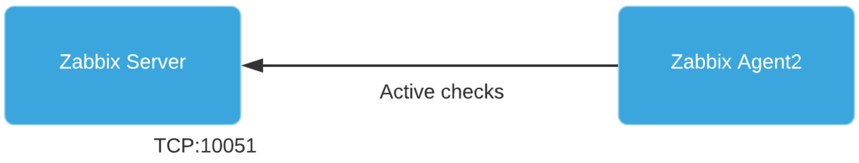


Figure 2.9 – Communication diagram between server and active agent

As mentioned, we can use both types of checks at the same time, giving us the freedom to configure every type of check we could possibly need. Our setup would then look like this:

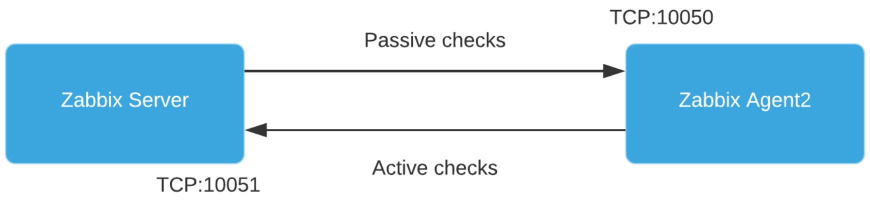


Figure 2.10 – Communication diagram between server and both agent types

See also

There's a lot of new stuff going on under the hood of Zabbix agent 2; if you're interested in learning more about the core of Zabbix agent 2, check out this cool blog post by Alexey Petrov: `https://blog.zabbix.com/magic-of-new-zabbix-agent/8460/`.

Working with SNMP monitoring

Now let's do something I enjoy most when working with Zabbix: build SNMP monitoring. My professional roots lie in network engineering, and I have worked a lot with SNMP monitoring to monitor all these different network devices.

Getting ready

To get started, we need the two Linux hosts we used before in the previous recipes:

- Our Zabbix server host
- The host we used in the previous recipe to monitor via the Zabbix active agent

How to do it…

Monitoring via SNMP polling is easy and very powerful. We will start by configuring SNMPv3 on our monitored Linux host:

1. Let's start by issuing the following commands to install SNMP on our host:

   For RHEL-based systems:

   ```
   dnf install net-snmp net-snmp-utils
   systemctl stop snmpd
   ```

   For Debian-based systems:

   ```
   apt-get install net-snmp net-snmp-utils
   systemctl stop snmpd
   ```

2. Now let's create the new SNMPv3 user we will use to monitor our host. Please note that we'll be using insecure passwords, but make sure to use secure passwords for your production environments. Issue the following command:

   ```
   net-snmp-create-v3-user -ro -A my_authpass -X my_privpass
   -a SHA -x AES snmpv3user
   ```

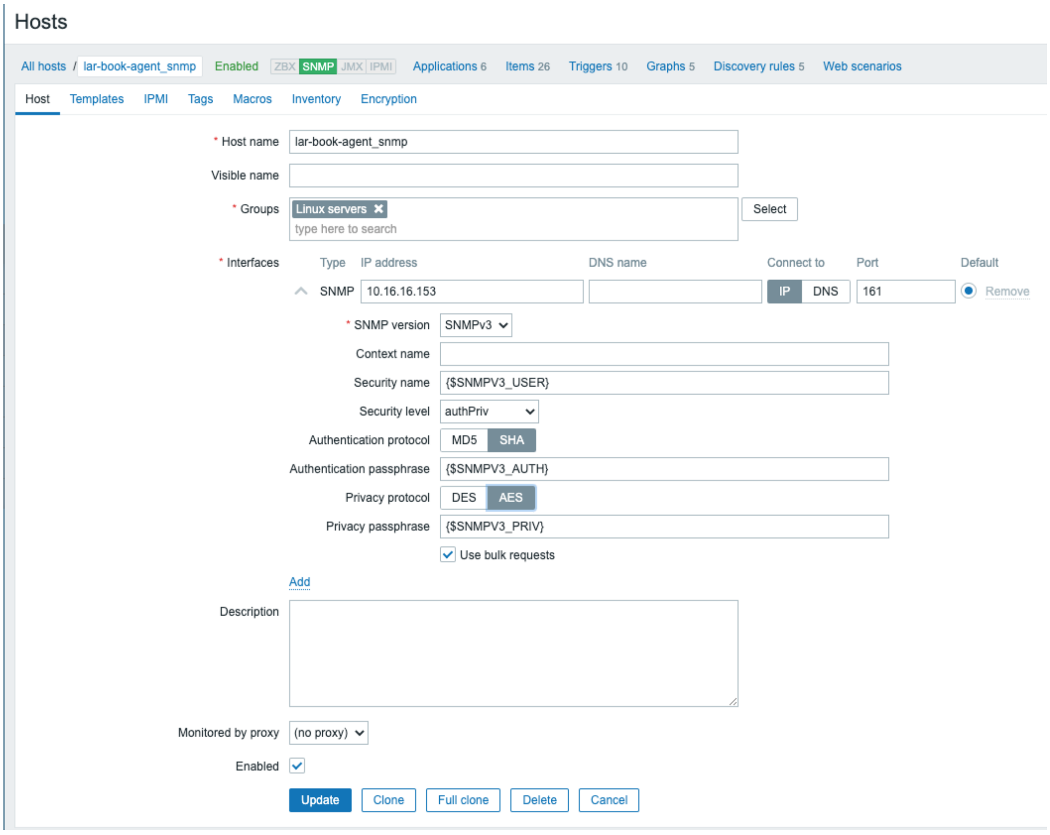
   This will create an SNMPv3 user with the username `snmpv3user`, the authentication password `my_authpass`, and the privilege password `my_privpass`.

3. Now restart the `snmpd` daemon and enable it at boot:

   ```
   systemctl restart snmpd.service
   systemctl enable snmpd.service
   ```

   This is all we need to do on the Linux host side; we can now go to the Zabbix frontend to configure our host. Go to **Configuration | Hosts** in your Zabbix frontend and click **Create host** in the top-right corner.

4. Now fill in the host configuration page:



Hosts

All hosts / lar-book-agent_snmp | Enabled | ZBX SNMP JMX IPMI | Applications 6 | Items 26 | Triggers 10 | Graphs 5 | Discovery rules 5 | Web scenarios

Host | Templates | IPMI | Tags | Macros | Inventory | Encryption

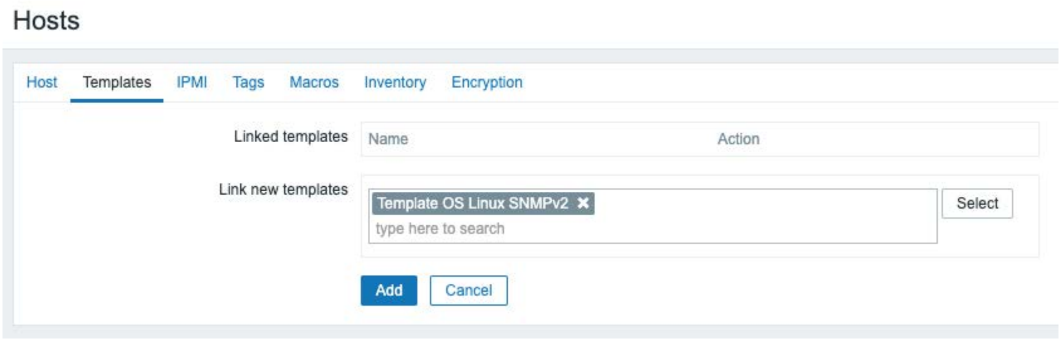
* Host name	lar-book-agent_snmp
Visible name	
* Groups	Linux servers
* Interfaces	

Type	IP address	DNS name	Connect to	Port	Default
SNMP	10.16.16.153		IP	161	●

* SNMP version	SNMPv3
Context name	
Security name	{$SNMPV3_USER}
Security level	authPriv
Authentication protocol	MD5 [✓ SHA]
Authentication passphrase	{$SNMPV3_AUTH}
Privacy protocol	DES [✓ AES]
Privacy passphrase	{$SNMPV3_PRIV}
	✓ Use bulk requests
Description	
Monitored by proxy	(no proxy)
Enabled	✓

Figure 2.11 – Zabbix host configuration page for host lar-book-agent_snmp

5. Make sure to add the right out-of-the-box template as shown in the following screenshot:



Hosts

Host | Templates | IPMI | Tags | Macros | Inventory | Encryption

Linked templates	Name / Action
Link new templates	Template OS Linux SNMPv2

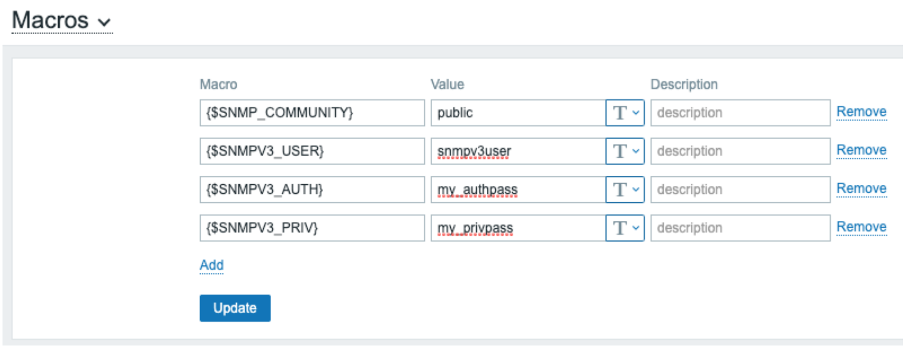
Figure 2.12 – Zabbix host template page for host lar-book-agent_snmp

Tip

While upgrading from an earlier Zabbix version to Zabbix 5, you won't get all the new out-of-the-box templates. If you feel like you are missing some templates, you can download them at the Zabbix Git repository: `https://git.zabbix.com/projects/ZBX/repos/zabbix/browse/templates`.

6. We are using some macros in our configuration here for the username and password. We can use these macros to actually add a bunch of hosts with the same credentials. This is very useful, for instance, if you have a bunch of switches with the same SNMPv3 credentials.

   Let's fill in the macros under **Administration** | **General** and use the dropdown to select **Macros**. Fill in the macros like this:



Macros

Macro	Value	Description	
{$SNMP_COMMUNITY}	public	description	Remove
{$SNMPV3_USER}	snmpv3user	description	Remove
{$SNMPV3_AUTH}	my_authpass	description	Remove
{$SNMPV3_PRIV}	my_privpass	description	Remove

Figure 2.13 – Zabbix global macro page with SNMP macros

A cool new feature in Zabbix 5 is the ability to hide macros in the frontend; do keep in mind that these values are still unencrypted in the Zabbix database.

7. Use the dropdown to change **{$SNMPv3_AUTH}** and **{$SNMPv3_PRIV}** to **Secret text**:

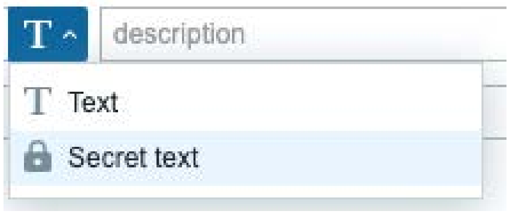
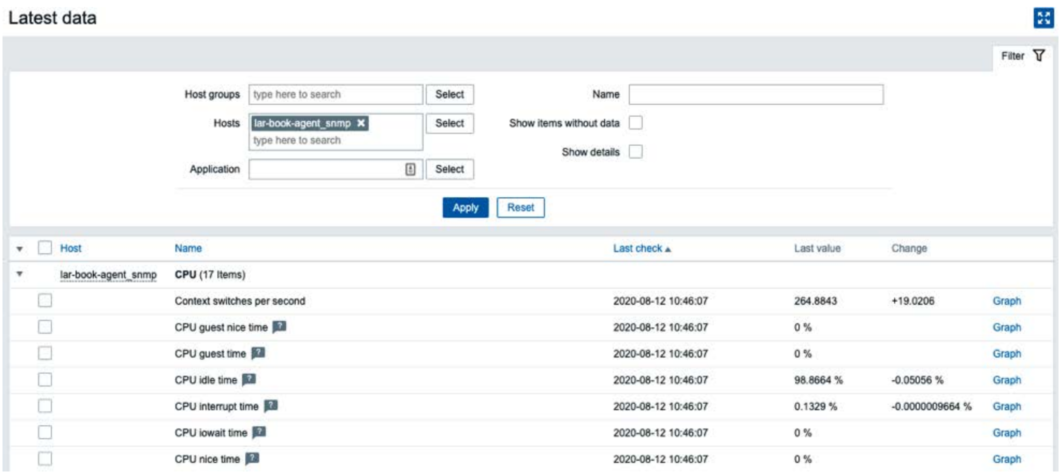


Figure 2.14 – Zabbix secure text dropdown for SNMP auth and priv macros

8. Now after applying these changes, we should be able to monitor our Linux server via SNMPv3. Let's go to **Monitoring | Hosts** and check the **Latest data** page for our new host:



Latest data

Filter

Host groups: type here to search [Select]
Hosts: lar-book-agent_snmp ✕ type here to search [Select]
Application: [Select]
Name:
Show items without data ☐
Show details ☐
[Apply] [Reset]

	Host	Name	Last check ▲	Last value	Change	
▾	lar-book-agent_snmp	**CPU** (17 Items)				
☐		Context switches per second	2020-08-12 10:46:07	264.8843	+19.0206	Graph
☐		CPU guest nice time	2020-08-12 10:46:07	0 %		Graph
☐		CPU guest time	2020-08-12 10:46:07	0 %		Graph
☐		CPU idle time	2020-08-12 10:46:07	98.8664 %	-0.05056 %	Graph
☐		CPU interrupt time	2020-08-12 10:46:07	0.1329 %	-0.0000009664 %	Graph
☐		CPU iowait time	2020-08-12 10:46:07	0 %		Graph
☐		CPU nice time	2020-08-12 10:46:07	0 %		Graph

Figure 2.15 – SNMP latest data for host lar-book-agent_snmp

How it works...

When we create a host as we did in *step 4*, Zabbix polls the host using SNMP. Polling SNMP like this works with SNMP OIDs. For instance, when we poll the item called **Free memory**, we ask the SNMP agent running on our Linux host to show us the value for **1.3.6.1.4.1.2021.4.6.0**. That value is then returned to us on the Zabbix server:

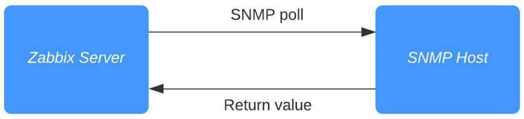


Figure 2.16 – Diagram showing communication between Zabbix server and SNMP host

Of course, SNMPv3 adds authentication and encryption to this process.

SNMP OIDs work in a tree structure, meaning every number behind the dot can contain another value. For example, see this for our host:

```
1.3.6.1.4.1.2021.4 = UCD-SNMP-MIB::memory
```

If we poll that Define acronym, we get several OIDs back:

```
.1.3.6.1.4.1.2021.4.1.0 = INTEGER: 0
.1.3.6.1.4.1.2021.4.2.0 = STRING: swap
.1.3.6.1.4.1.2021.4.3.0 = INTEGER: 1679356 kB
.1.3.6.1.4.1.2021.4.4.0 = INTEGER: 1674464 kB
.1.3.6.1.4.1.2021.4.5.0 = INTEGER: 1872872 kB
.1.3.6.1.4.1.2021.4.6.0 = INTEGER: 184068 kB
```

That includes our `1.3.6.1.4.1.2021.4.6.0` OID with the value that contains our free memory. This is how SNMP is built, like a tree.

Creating Zabbix simple checks and the Zabbix trapper

In this recipe, we will go over two checks that can help you built some more customized setups. The Zabbix simple checks provide you with an easy way to monitor some specific data. The Zabbix trapper combines with Zabbix sender to get data from your hosts into the server, allowing for some scripting options. Let's get started.

Getting ready

To create these checks, we will need a Zabbix server and a Linux host to monitor. We can use the host with a Zabbix agent and SNMP monitoring from the previous recipes.

Do note for these checks that we do not actually need the Zabbix agent.

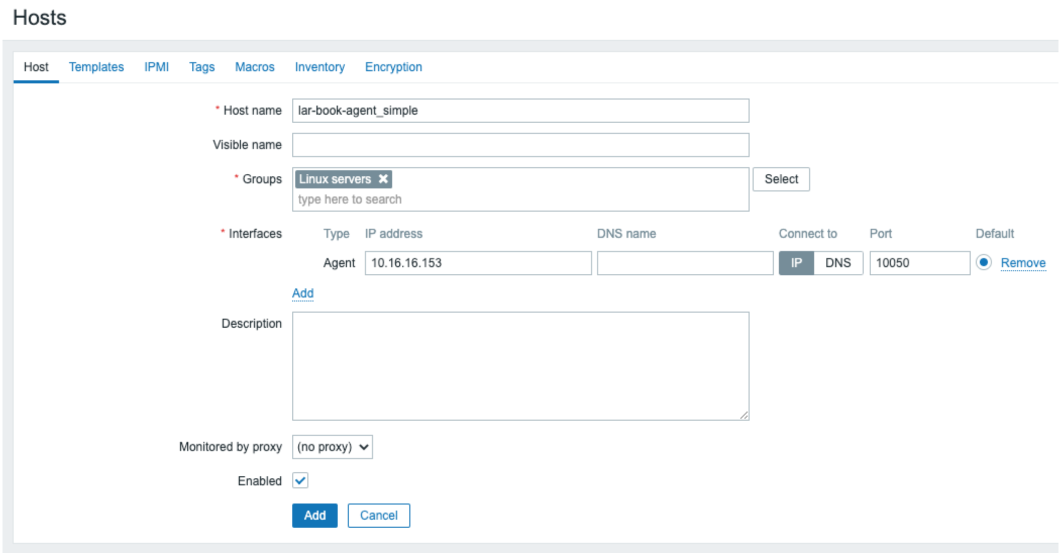
How to do it…

Working with simple checks is quite simple, as the name suggests, so let's start.

Creating simple checks

We will create a simple check to monitor whether a service is running and accepting TCP connections on a certain port:

1. To get this done, we will need to create a new host in Zabbix frontend. Go to **Configuration** | **Hosts** in your Zabbix frontend and click **Create host** in the top-right corner.
2. Create a host with the following settings:

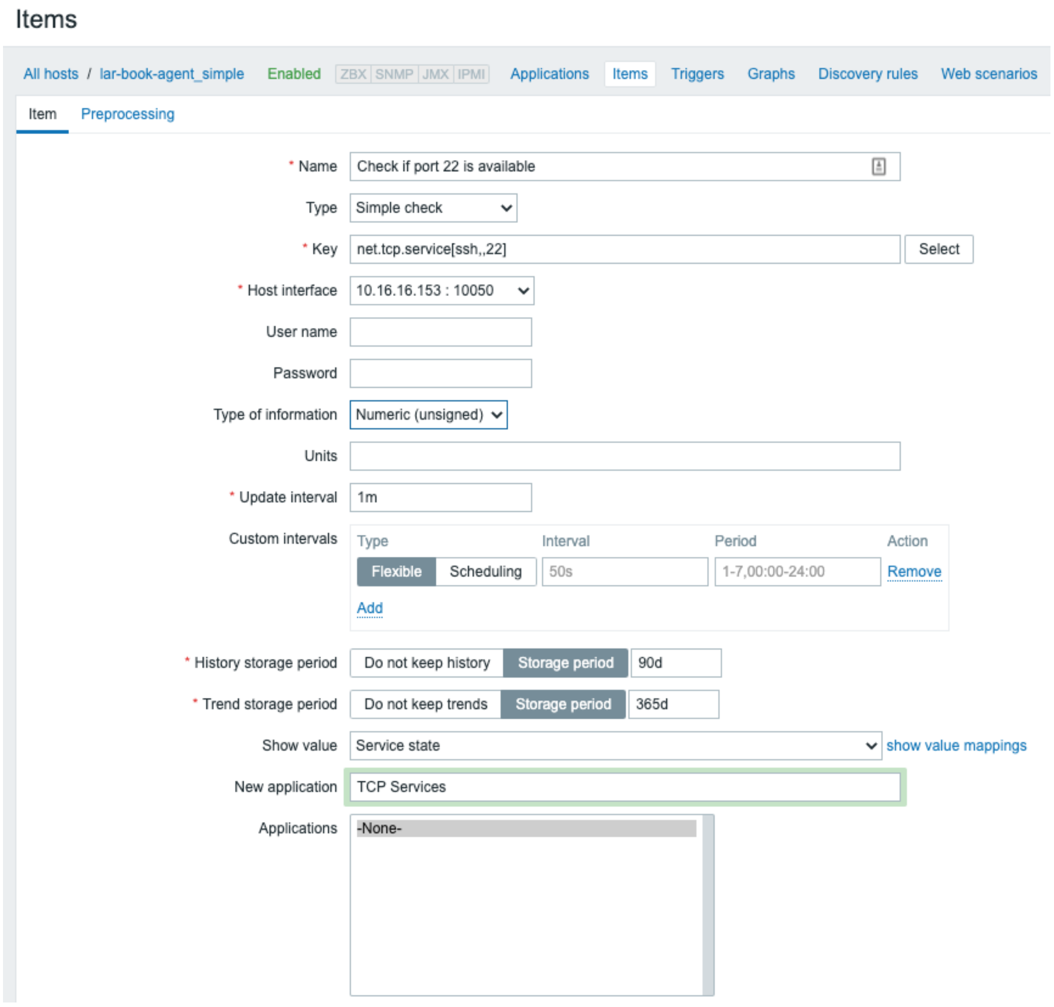


* Host name	lar-book-agent_simple
Visible name	
* Groups	Linux servers
* Interfaces	Type: Agent; IP address: 10.16.16.153; DNS name: ; Connect to: IP; Port: 10050; Default: ●
Description	
Monitored by proxy	(no proxy)
Enabled	✓

Figure 2.17 – Zabbix host configuration page for host lar-book-agent_simple

3. Now go to **Configuration** | **Hosts**, click the newly created host, and go to **Items**. We want to create a new item here by clicking the **Create item** button.

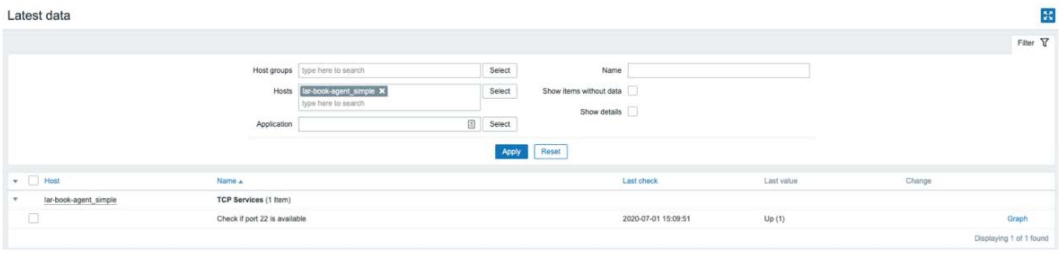
We will create a new item with the following values, and after doing, so we will click the **Add** button at the bottom of the page:



* Name	Check if port 22 is available
Type	Simple check
* Key	net.tcp.service[ssh,,22]
* Host interface	10.16.16.153 : 10050
User name	
Password	
Type of information	Numeric (unsigned)
Units	
* Update interval	1m
Custom intervals	Type: Flexible / Scheduling; Interval: 50s; Period: 1-7,00:00-24:00; Action: Remove; Add
* History storage period	Do not keep history [✓ Storage period] 90d
* Trend storage period	Do not keep trends [✓ Storage period] 365d
Show value	Service state (show value mappings)
New application	TCP Services
Applications	-None-

Figure 2.18 – Zabbix item port 22 check configuration page for host lar-book-agent_simple

4. Now we should be able to see whether our server is accepting SSH connections on port `22` on our **Latest data** screen. Navigate to **Monitoring | Hosts** and check the **Latest data** screen for our new value:



Host	Name ▲	Last check	Last value	Change	
lar-book-agent_simple	**TCP Services** (1 Item)				
	Check if port 22 is available	2020-07-01 15:09:51	Up (1)		Graph

Displaying 1 of 1 found

Figure 2.19 – Zabbix Latest data page for host lar-book-agent_simple, item port 22 check

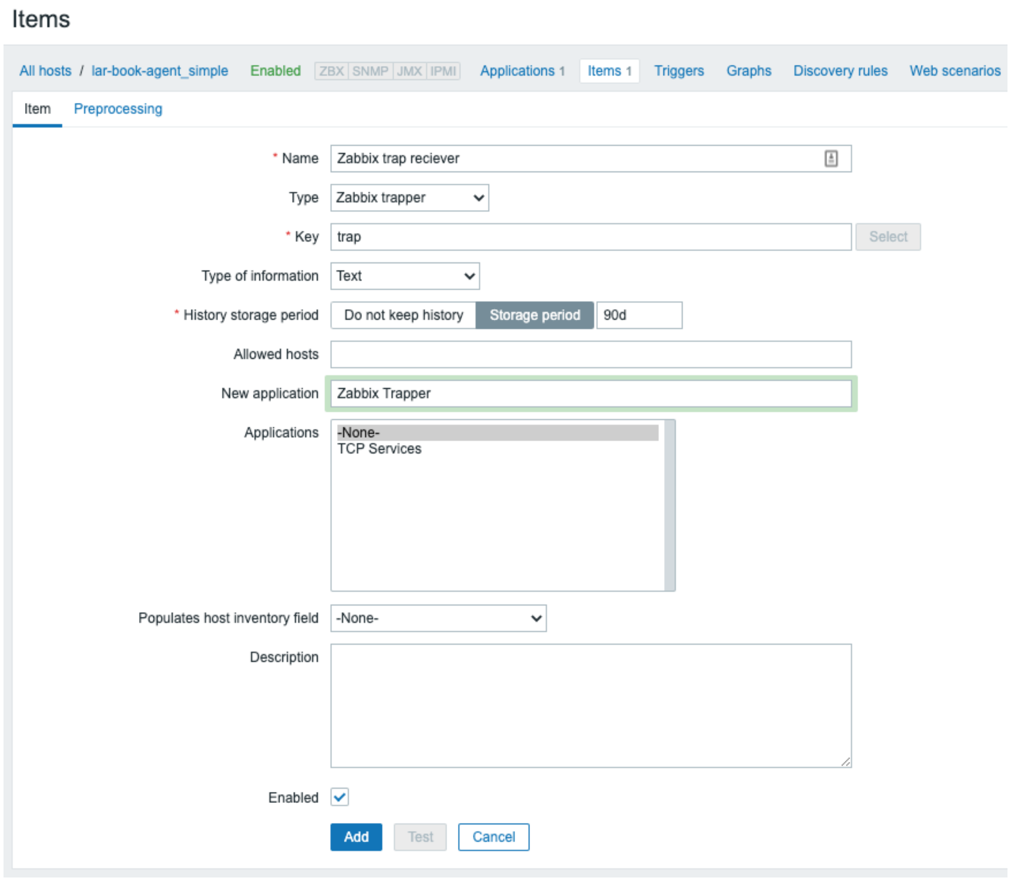
That's all there is to creating your simple checks in Zabbix. Now let's move on to the Zabbix trapper item.

Creating a trapper

We can do some cool stuff with Zabbix trapper items once we get more advanced setups. But for now, let's create an item on our **lar-book-agent_simple** host:

1. Go to **Configuration | Hosts** and click the host, then go to **Items**. We want to create a new item here again by clicking the **Create item** button.

Now let's create the following item and click the **Add** button:


Items

All hosts / lar-book-agent_simple | Enabled | ZBX SNMP JMX IPMI | Applications 1 | Items 1 | Triggers | Graphs | Discovery rules | Web scenarios

Item | Preprocessing

* Name	Zabbix trap reciever
Type	Zabbix trapper
* Key	trap
Type of information	Text
* History storage period	Do not keep history / Storage period: 90d
Allowed hosts	
New application	Zabbix Trapper
Applications	-None- TCP Services
Populates host inventory field	-None-
Description	
Enabled	✓

Figure 2.20 – Zabbix item trap receiver configuration screen for host lar-book-agent_simple

2. If we go to the CLI of our monitored server, we can now execute the following to install Zabbix sender:

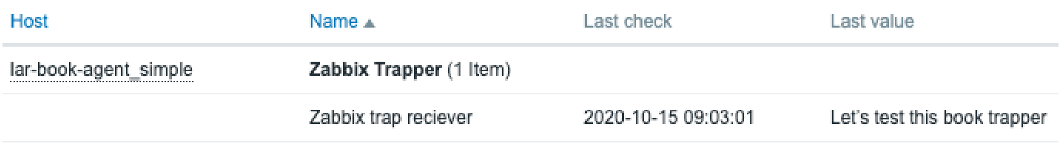
```
sudo dnf -y install zabbix-sender
```

3. After installation, we can use Zabbix sender to send some information to our server:

```
zabbix_sender -z 10.16.16.152 -p 10051 -s "lar-book-
agent_simple" -k trap -o "Let's test this book trapper"
```

Now we should be able to see whether our monitored host has sent out the Zabbix trap and the Zabbix server has received this trap for processing.

4. Navigate to **Monitoring** | **Hosts** and check the **Latest data** screen for our new value:



Host	Name ▲	Last check	Last value
lar-book-agent_simple	**Zabbix Trapper** (1 Item)		
	Zabbix trap reciever	2020-10-15 09:03:01	Let's test this book trapper

Figure 2.21 – Zabbix Latest data page for host lar-book-agent_simple, item trap receiver

There it is, our Zabbix trap in our Zabbix frontend.

How it works...

Now that we have built our new items, let's see how they work by diving into the theoretical side of Zabbix simple checks and trappers.

Simple checks

Zabbix simple checks are basically a list of built-in checks made for monitoring certain values. There is a list and description available on the Zabbix documentation wiki: `https://www.zabbix.com/documentation/current/manual/config/items/itemtypes/simple_checks`.

All of these checks are performed by the Zabbix server to collect data from a monitored host. For example, when we do the Zabbix simple check to check whether a port is open, our Zabbix server requests simply checks whether it can reach that port.

This means that if your monitored host's firewall is blocking port `22` from Zabbix server, we'll get a service is down value. However, this does not mean that SSH isn't running on the server itself:

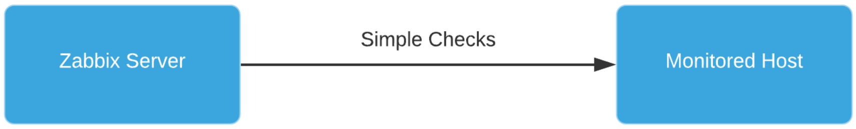


Figure 2.22 – Zabbix server-to-host communication diagram

Tip

Keep in mind that working with simple checks is dependent on external factors such as the firewall settings on the monitored host. When you build a custom simple check, make sure to check these factors as well.

Trappers

When working with Zabbix sender, we are doing exactly the opposite of most checks. We are building an item on our Zabbix server, which allows us to capture trap items. This allows us to build some custom checks to send data to our Zabbix server from a monitored host:

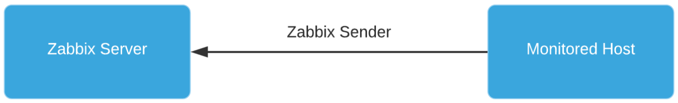


Figure 2.23 – Zabbix server trap receiver diagram

Let's say, for instance, that you want to build a custom Python script that, at the end of running the scripts, sends output to Zabbix server. We could ask Python to send this data with Zabbix sender, and suddenly you'd have this data available for processing on Zabbix server.

We can really extend our options with Zabbix trapper and customize our Zabbix server even further.

Working with calculated and dependent items

Calculated and dependent items are used in Zabbix to produce additional values from existing values. Sometimes, we have already collected a value and we need to do more with the values created by that item. We can do exactly that by using calculated and dependent items.

Getting ready

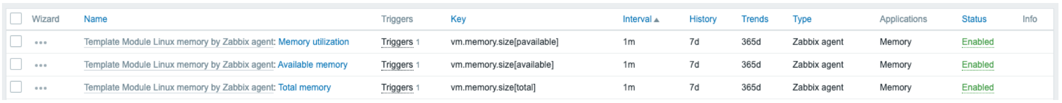
To work with calculated items and dependent items, we are going to need the Zabbix server and monitored hosts from the previous recipes. We will add the items on the **lar-book-agent_passive** host and our Zabbix server host, so we already have some items available to calculate and make dependent.

How to do it...

Let's see how we can extend our items by getting started with the calculated items.

Working with calculated items

1. Let's navigate to our host configuration by going to **Configuration | Hosts** and clicking on our **lar-book-agent_passive** host's items. In the filter field named **Name**, enter `memory` and you will get the following output:

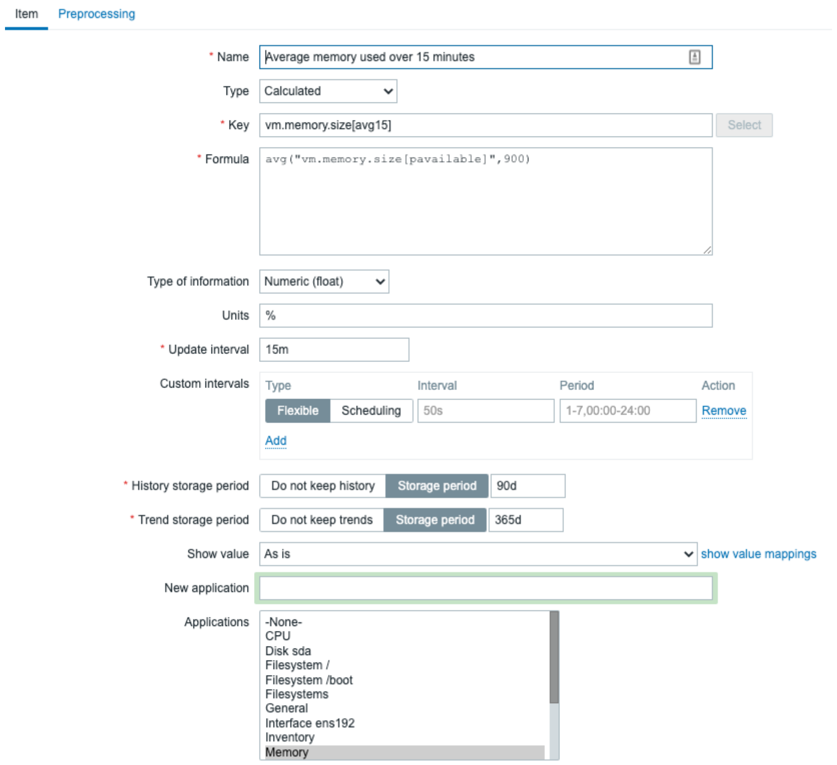


	Wizard	Name	Triggers	Key	Interval ▲	History	Trends	Type	Applications	Status	Info
☐	•••	Template Module Linux memory by Zabbix agent: Memory utilization	Triggers 1	vm.memory.size[pavailable]	1m	7d	365d	Zabbix agent	Memory	Enabled	
☐	•••	Template Module Linux memory by Zabbix agent: Available memory	Triggers 1	vm.memory.size[available]	1m	7d	365d	Zabbix agent	Memory	Enabled	
☐	•••	Template Module Linux memory by Zabbix agent: Total memory	Triggers 1	vm.memory.size[total]	1m	7d	365d	Zabbix agent	Memory	Enabled	

Figure 2.24 – Zabbix item page for host lar-book-agent_passive

2. What we are going to do now is create a calculated item that is going to show us the average memory utilization over a period of 15 minutes. We can use this value to determine how busy our host was during that period, without having to look at the graphs.
3. Let's click the **Create item** button and start creating our new calculated item.

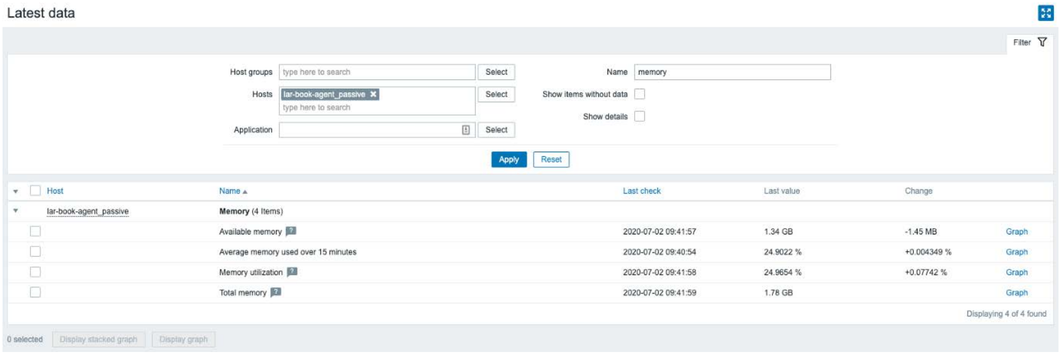
   We want our item to have the following values:



* Name	Average memory used over 15 minutes
Type	Calculated
* Key	vm.memory.size[avg15]
* Formula	avg("vm.memory.size[pavailable]",900)
Type of information	Numeric (float)
Units	%
* Update interval	15m
Custom intervals	Type: Flexible / Scheduling; Interval: 50s; Period: 1-7,00:00-24:00; Action: Remove; Add
* History storage period	Do not keep history / Storage period: 90d
* Trend storage period	Do not keep trends / Storage period: 365d
Show value	As is (show value mappings)
New application	
Applications	-None-, CPU, Disk sda, Filesystem /, Filesystem /boot, Filesystems, General, Interface ens192, Inventory, Memory

Figure 2.25 – Zabbix item configuration page, average memory used

4. Now if we go to check our **Monitoring | Hosts** page and select **Latest data**, we can check out our value. Make sure to filter in the **Name** field for `memory`, so we see the correct values:



Host	Name ▲	Last check	Last value	Change	
lar-book-agent_passive	**Memory** (4 Items)				
	Available memory	2020-07-02 09:41:57	1.34 GB	-1.45 MB	Graph
	Average memory used over 15 minutes	2020-07-02 09:40:54	24.9022 %	+0.004349 %	Graph
	Memory utilization	2020-07-02 09:41:58	24.9654 %	+0.07742 %	Graph
	Total memory	2020-07-02 09:41:59	1.78 GB		Graph

Displaying 4 of 4 found

0 selected Display stacked graph Display graph

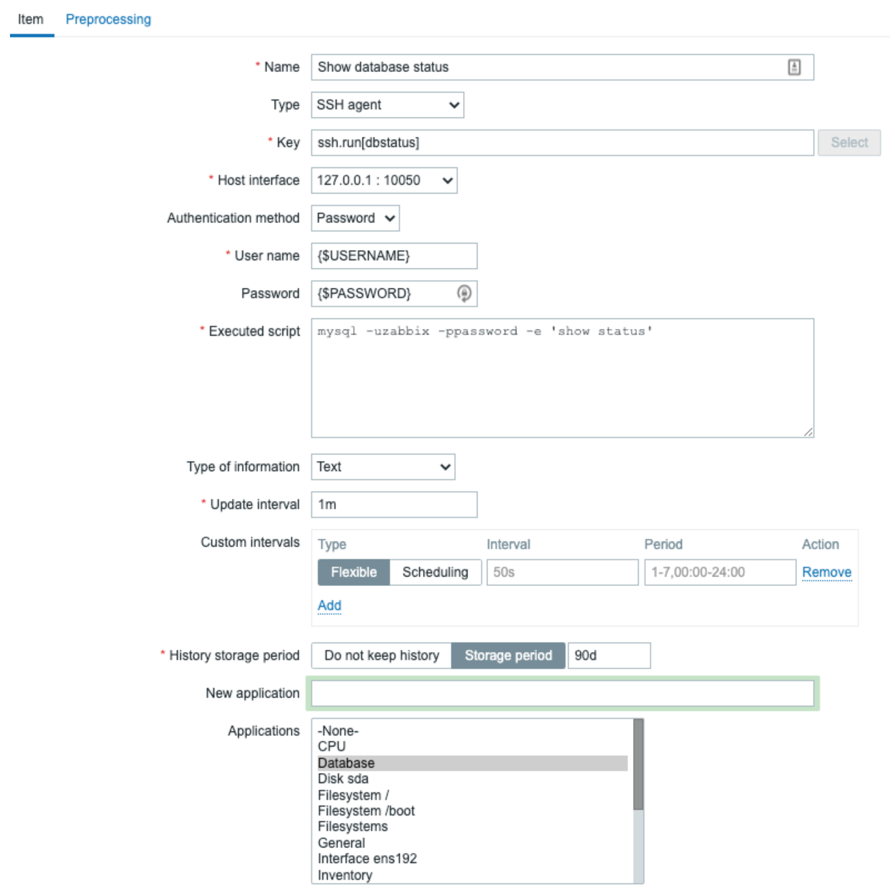
Figure 2.26 – Zabbix Latest data page for host lar-book-agent_passive, memory items

Now we can clearly see that we are calculating the 15-minute average of the memory utilization on our newly created item.

Working with dependent items

Time to make our first dependent item. I'll use the **lar-book-agent_centos** host or our (as it's called by default) **Zabbix server**. Let's say we want to request some variables from our MySQL database in one big batch. We can then create dependent items on top of the first item to further process the data:

1. Let's start by creating the main check. Navigate to **Configuration | Hosts**, select our host, and then let's click the **Create item** button to start creating our first new item. We want an item with the following variables:



* Name	Show database status
Type	SSH agent
* Key	ssh.run[dbstatus]
* Host interface	127.0.0.1 : 10050
Authentication method	Password
* User name	{$USERNAME}
Password	{$PASSWORD}
* Executed script	`mysql -uzabbix -ppassword -e 'show status'`
Type of information	Text
* Update interval	1m
Custom intervals	Type: Flexible / Scheduling; Interval: 50s; Period: 1-7,00:00-24:00; Action: Remove; Add
* History storage period	Do not keep history / Storage period 90d
New application	
Applications	-None-, CPU, Database, Disk sda, Filesystem /, Filesystem /boot, Filesystems, General, Interface ens192, Inventory

Figure 2.27 – Zabbix item configuration page, database status

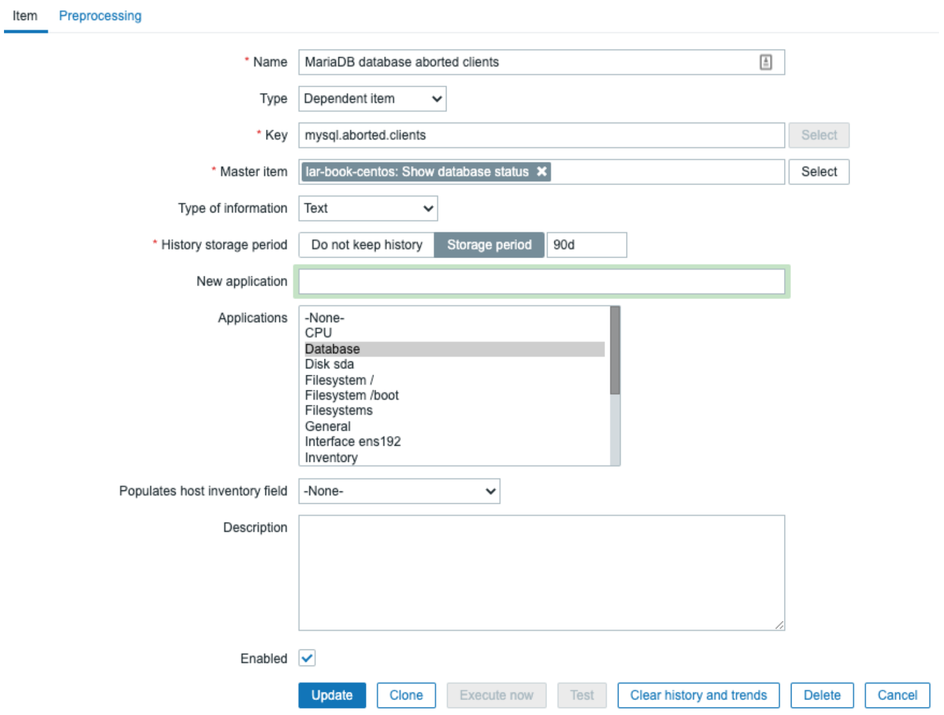
Now, this item is an SSH check that logs in to our Zabbix server machine and executes the code in the script. The code will then log in to our MariaDB database and it will show the status. Make sure to enter your credentials correctly.

Tip

SSH checks can be used for a variety of cool scenarios, such as this one. We simply log in to our CLI and execute a piece of code. Perfect for your custom setups.

2. Go to **Monitoring | Hosts** and check out the **Latest data** page for our new check. There should be a long list of MariaDB values. If so, we can continue with our next step of creating the dependent item.

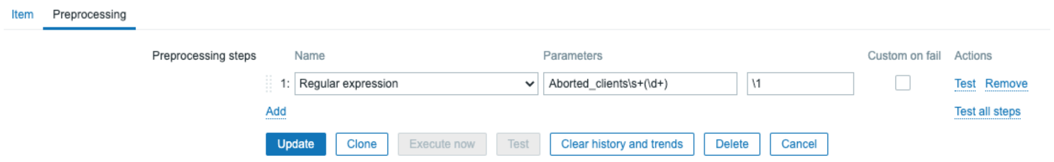
3. To create the dependent item, navigate to **Configuration** | **Hosts**, select our host, and let's click the **Create item** button. We want this item to get the following variables:



* Name	MariaDB database aborted clients
Type	Dependent item
* Key	mysql.aborted.clients
* Master item	lar-book-centos: Show database status
Type of information	Text
* History storage period	Do not keep history / Storage period: 90d
New application	
Applications	-None-, CPU, Database, Disk sda, Filesystem /, Filesystem /boot, Filesystems, General, Interface ens192, Inventory
Populates host inventory field	-None-
Description	
Enabled	✓

Figure 2.28 – Zabbix item configuration page, MariaDB aborted clients

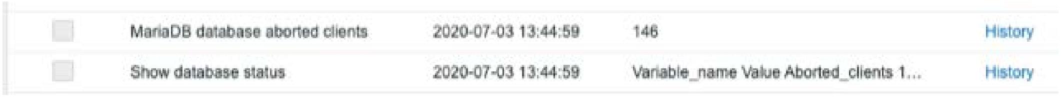
4. It's very important to add preprocessing to this item as well, otherwise we will simply get the same data as our master item. So, let's add the following:



Preprocessing steps	Name	Parameters		Custom on fail	Actions
1:	Regular expression	Aborted_clients\s+(\d+)	\1		Test Remove

Figure 2.29 – Zabbix item preprocessing page, MariaDB aborted clients

The result will be the number of aborted clients for our MariaDB:



MariaDB database aborted clients	2020-07-03 13:44:59	146	History
Show database status	2020-07-03 13:44:59	Variable_name Value Aborted_clients 1...	History

Figure 2.30 – Zabbix Latest data page, MariaDB aborted clients

How it works...

The types we've discussed in this *How to do it…* section can be quite complicated; let's go over how the items actually work.

Calculated items

Working with calculated items can be a great way to get even more statistics out of your existing data. Sometimes you just need to combine multiple items into one specific value.

What we did just now works by taking several values in a period of 15 minutes of 1 item and calculating the average like this:

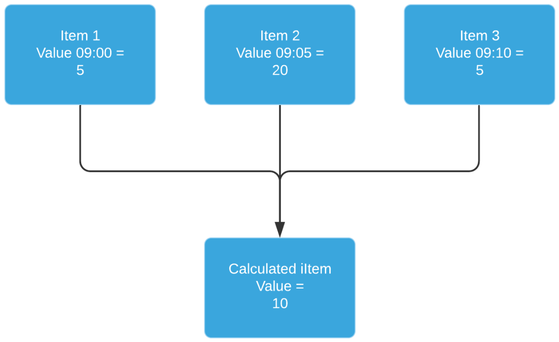


Figure 2.31 – Zabbix dependent item diagram

We're taking those values and calculating the average every 15 minutes. It gives us a nice indication of what we are doing over a set period of time.

Dependent items

Dependent items work simply by taking the data from a master item and processing that data into other data. This way, we can structure our data and keep our check interval for all these items the same:

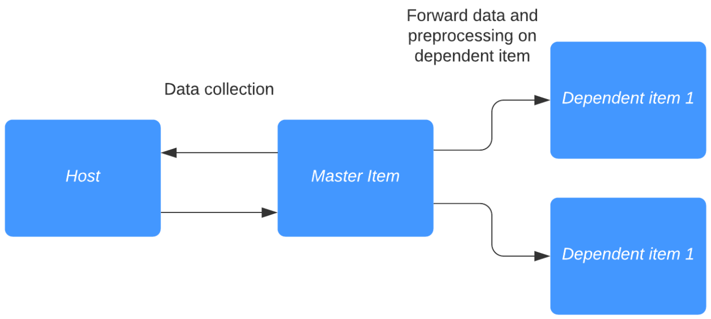


Figure 2.32 – Dependent item diagram

As seen here, dependent items basically work as duplicators, with additional preprocessing options. Do note that preprocessing must be used to extract data from the master item. Without preprocessing, we can't extract our information as in the example.

Tip

Sometimes we do not require our master item to be saved in our database; we already have the information in our dependent items. When we don't want the master item to be saved, we simply select the **Do not keep history** option.

Creating external checks

To further extend Zabbix functionality, we can use our own custom scripts, which are used by Zabbix external checks. Not everything that you want to monitor will always be standard in Zabbix, though a lot is. There's always something that could be missing, and external checks are just the way to bypass some of these.

Getting ready

For this recipe, we are going to need just our Zabbix server. We can create an item on our `lar-book-centos` host, which is our Zabbix server monitored host.

How to do it…

1. First let's create a script that we will execute in `/usr/lib/zabbix/externalscripts/` called `test_external` with the following command:

   ```
   vim /usr/lib/zabbix/externalscripts/test_external
   ```

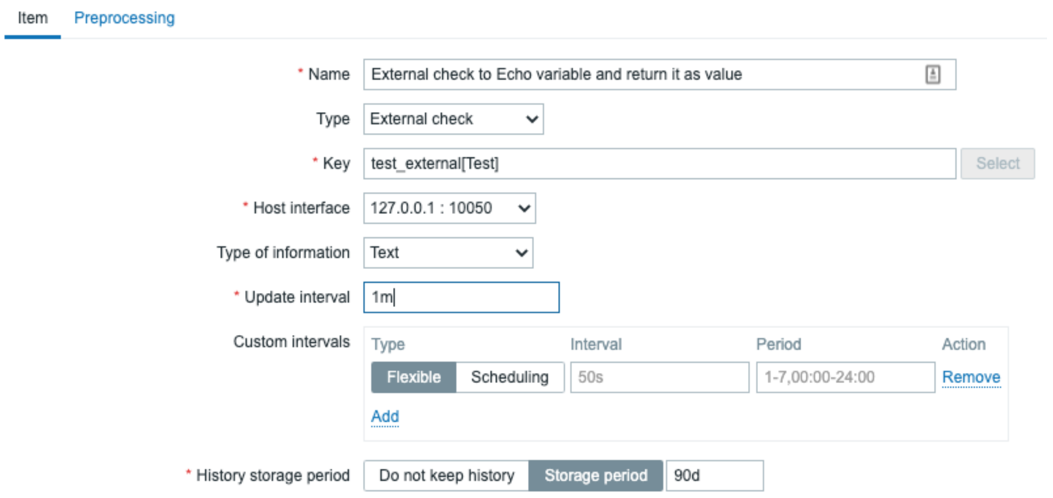
   Add the following code to this file and save:

   ```
   #!/bin/bash
   echo $1
   ```

   Tip

   Make sure Zabbix server can execute the script by adding the right permissions to the file. Zabbix needs to be able to access and execute the file.

2. Let's navigate to our host to create a new item. Navigate to **Configuration | Hosts**, select our host, **lar-book-centos**, and click the **Create item** button. We want this item to get the following variables:

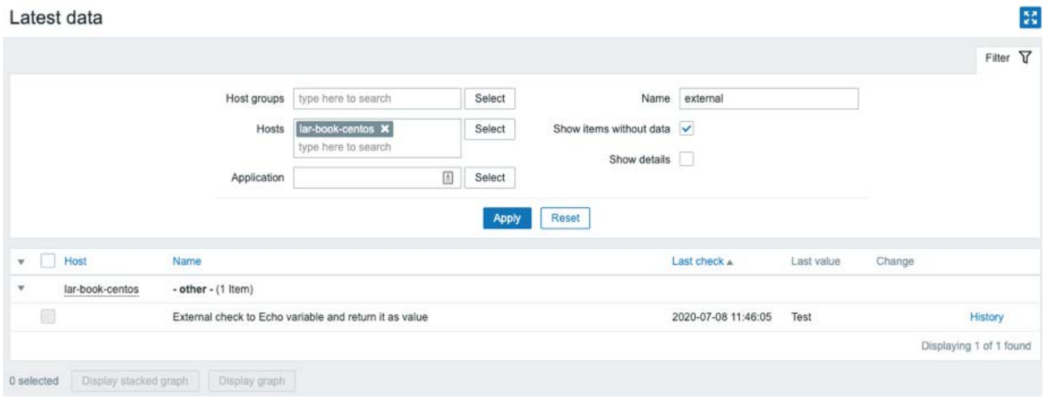


Item | Preprocessing

* Name	External check to Echo variable and return it as value
Type	External check
* Key	test_external[Test]
* Host interface	127.0.0.1 : 10050
Type of information	Text
* Update interval	1m
Custom intervals	Type: Flexible / Scheduling; Interval: 50s; Period: 1-7,00:00-24:00; Action: Remove; Add
* History storage period	Do not keep history / Storage period: 90d

Figure 2.33 – Zabbix item configuration page

3. After adding this new item, let's navigate to **Monitoring | Hosts** and check the **Latest data** page for our host. We should get our **Test** variable returned by our script as **Value** in Zabbix, as shown in the following screenshot:



Latest data

Filter

Host groups	type here to search	Name	external
Hosts	lar-book-centos	Show items without data	✓
Application		Show details	

Apply | Reset

Host	Name	Last check ▲	Last value	Change	
lar-book-centos	- **other** - (1 Item)				
	External check to Echo variable and return it as value	2020-07-08 11:46:05	Test		History

Displaying 1 of 1 found

0 selected | Display stacked graph | Display graph

Figure 2.34 – Zabbix Latest data page

Tip

Use the macros in the frontend as variables to send data from your frontend to your scripts. You can further automate your checks with this to enhance your external checks.

How it works...

External checks seem like they have a steep learning curve, but they are actually quite simple from the Zabbix side. All we do is send a command to an external script and expect a result output:

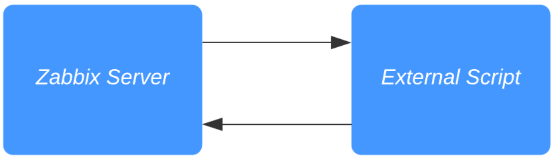


Figure 2.35 – Zabbix server external script communication diagram

Like in our example, we sent the value `Test` to our script, which the script then in turn echoed back to use as `$1`.

When you have good knowledge of a programming language such as Python, for example, we can use this function to build a lot more expansion to our Zabbix capabilities. A simple yet powerful tool to work with.

Setting up JMX monitoring

Built into Zabbix is JMX monitoring, so we can monitor our Java applications. In this recipe, we'll check out how to monitor Apache Tomcat with Zabbix JMX so we can get a feel for what this monitoring option is all about.

Getting ready

To get ready for this recipe, we are going to need our Zabbix server to monitor our JMX application.

I also used a CentOS 7 machine for this recipe, with Tomcat installed. It can be quite tricky to use Tomcat on CentOS 8 due to package dependencies, so I recommend sticking with 7 for now. You can add the following to your Tomcat configuration after installation to get it working in our recipe:

```
JAVA_OPTS="-Djava.rmi.server.hostname=10.16.16.155 -Dcom.sun.
management.jmxremote -Dcom.sun.management.jmxremote.port=12345
-Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.
management.jmxremote.ssl=false"
```

How to do it...

To set up JMX monitoring, we are going to add a host to our Zabbix server that will monitor our Apache Tomcat installation. But first, we will need to add some settings to our `/etc/zabbix/zabbix_server.conf` file:

1. Let's edit the `zabbix_server.conf` file by logging in to our Zabbix server and executing the following command:

   ```
   Vim /etc/zabbix/zabbix_server.conf
   ```

2. We will then need to add the following lines to this file:

   ```
   JavaGateway=127.0.0.1
   StartJavaPollers=5
   ```

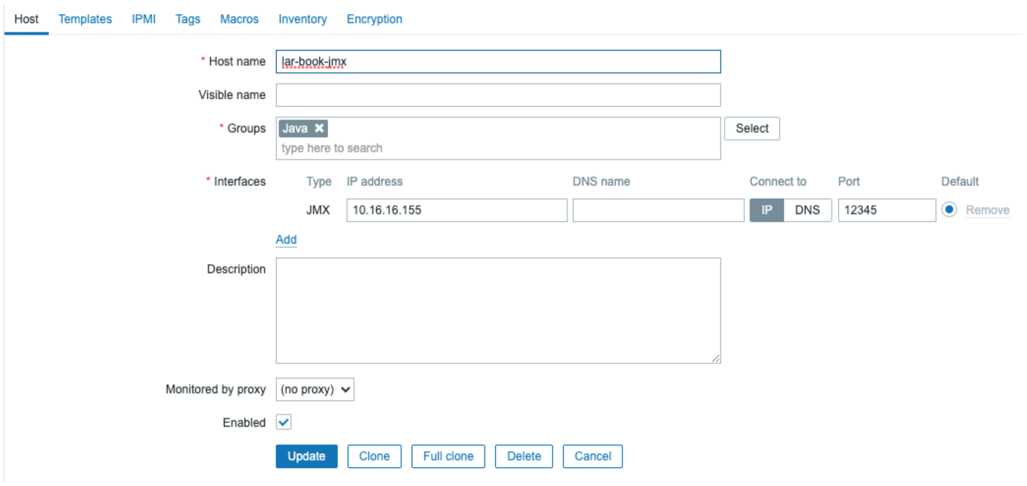
3. We will also need to install the `zabbix-java-gateway` application on our Zabbix server with the following command:

   ```
   dnf -y install zabbix-java-gateway
   ```

   That is all we need to do on the server side of things to get JMX monitoring to work. Zabbix doesn't include these settings by default, so this is why we need to add the text to our file and install the application.

4. To get started with monitoring our JMX host, go to **Configuration | Hosts** in your Zabbix frontend and click **Create host** in the top-right corner.

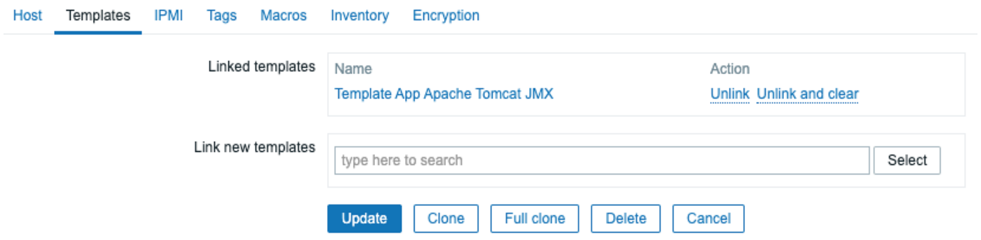
   We will then add a host with the following settings:



* Host name	lar-book-jmx
Visible name	
* Groups	Java
* Interfaces	Type: JMX; IP address: 10.16.16.155; DNS name: ; Connect to: IP; Port: 12345; Default: ●
Description	
Monitored by proxy	(no proxy)
Enabled	✓

Figure 2.36 – Zabbix item configuration page

5. Let's also make sure to add the following template to our host so we can actually start monitoring the JMX host:



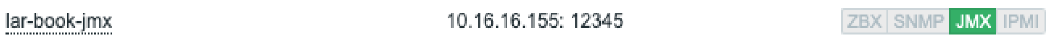
Linked templates

Name	Action
Template App Apache Tomcat JMX	Unlink Unlink and clear

Link new templates: type here to search [Select]

Figure 2.37 – Zabbix host template page

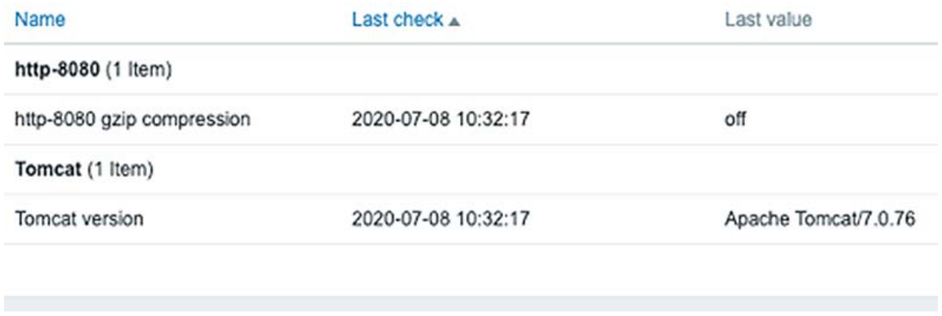
6. After this, our JMX icon should turn green; let's check this under **Monitoring | Hosts**. It should look like this:



lar-book-jmx 10.16.16.155: 12345

Figure 2.38 – Zabbix hosts page

7. If we click on the **Latest data** for our new JMX monitored host, we should also see our incoming data. Check it out; it should return stats like these:



Name	Last check ▲	Last value
http-8080 (1 Item)		
http-8080 gzip compression	2020-07-08 10:32:17	off
Tomcat (1 Item)		
Tomcat version	2020-07-08 10:32:17	Apache Tomcat/7.0.76

Figure 2.39 – Zabbix Latest data

How it works...

Zabbix utilizes a Java gateway either hosted on Zabbix server itself or hosted on another server (proxy) to monitor JMX applications:

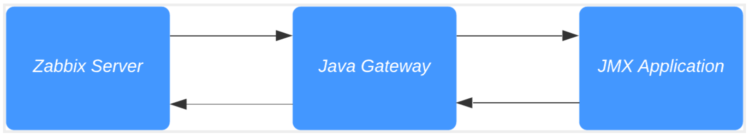


Figure 2.40 – Communication diagram between Zabbix server and Java

Zabbix communicates with the Java gateway and the Java gateway in turn communicates with our JMX application, as it does with Tomcat in our example. The data in turn is then returned through the same path and we can see our data in our Zabbix server.

See also

There are loads of applications that can be monitored through Zabbix JMX. Check out the Zabbix monitoring and integrations page for more uses of Zabbix JMX monitoring: `https://www.zabbix.com/integrations/jmx`.

Setting up database monitoring

Databases are a black hole to a lot of engineers; there's data being written to them and there's something being done with this data. But what if you want to know more about the health of your database? That's where Zabbix database monitoring comes in – we can use it to monitor the health of our database to a greater extent.

Getting ready

We'll be monitoring our Zabbix database, for convenience. This means that all we are going to need is our installed Zabbix server with our database on it. We'll be using MariaDB in this example, so if you have a PostgreSQL setup, make sure to install a MariaDB instance on a Linux host.

How to do it…

Before getting started with the item configuration, we'll have to do some stuff on the CLI side of the server:

1. Let's start by installing the required modules to our server:

```
dnf install unixODBC unixODBC-devel mariadb-connector-
odbc
```

2. Now let's verify whether our ODBC configuration files exist:

```
odbcinst -j
```

Your output should look as follows:

```
unixODBC 2.3.7
DRIVERS............: /etc/odbcinst.ini
SYSTEM DATA SOURCES: /etc/odbc.ini
```

```
FILE DATA SOURCES..: /etc/ODBCDataSources
USER DATA SOURCES..: /root/.odbc.ini
SQLULEN Size.......: 8
SQLLEN Size........: 8
SQLSETPOSIROW Size.: 8
```

3. If the output is correct, we can go to the Linux CLI and continue by editing `odbc.ini` to connect to our database:

   ```
   vim /etc/odbc.ini
   ```

   Now fill in your Zabbix database information. It will look like this:

   ```
   [book]
   Description = MySQL book test database
   Driver      = mariadb
   Server      = 127.0.0.1
   User        = zabbix
   Password    = password
   Port        = 3306
   Database    = zabbixdb
   ```

4. Now let's test whether our connection is working as expected by executing this:

   ```
   isql book
   ```

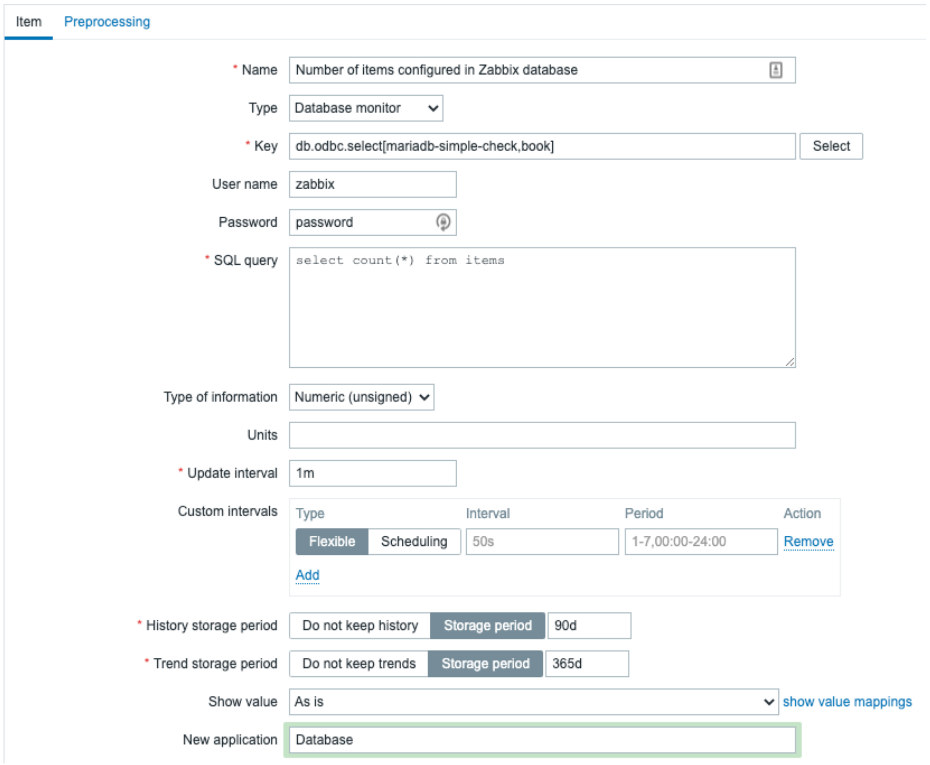
   You should get a message saying **Connected**; if you don't, then check your configuration files and try again.

5. Now let's move to the Zabbix frontend to configure our first database check. Navigate to **Configuration | Hosts** and click the host called **lar-book-centos**, or it might still be called **Zabbix server**. Now go to **Items**; we want to create a new item here by clicking the **Create item** button.

Tip

If you haven't already, a great way to keep your Zabbix structured is to keep all hostnames in Zabbix equal to the real server hostname. Rename your default *Zabbix server* host in the frontend to what you've actually called your server.

We want to add an item with the following parameters:

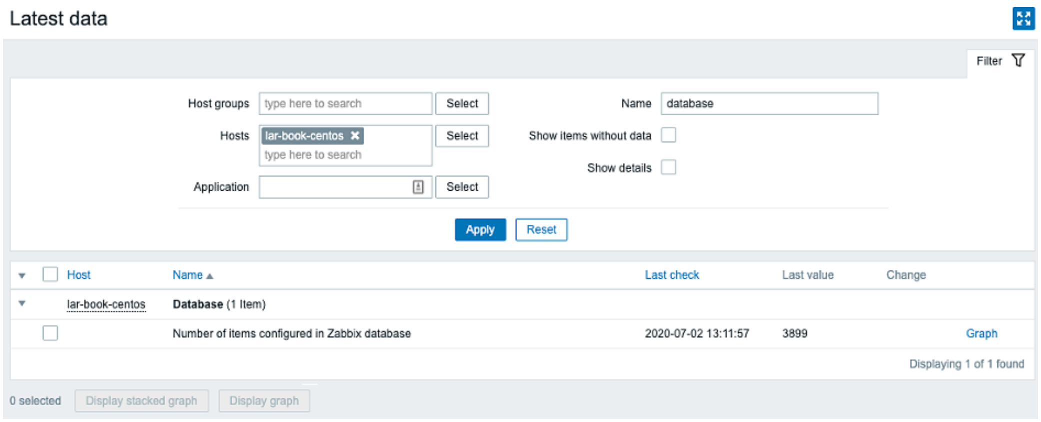


Item | Preprocessing

* Name	Number of items configured in Zabbix database
Type	Database monitor
* Key	db.odbc.select[mariadb-simple-check,book]
User name	zabbix
Password	password
* SQL query	select count(*) from items
Type of information	Numeric (unsigned)
Units	
* Update interval	1m
Custom intervals	Type: Flexible / Scheduling; Interval: 50s; Period: 1-7,00:00-24:00; Action: Remove; Add
* History storage period	Do not keep history / Storage period: 90d
* Trend storage period	Do not keep trends / Storage period: 365d
Show value	As is (show value mappings)
New application	Database

Figure 2.41 – Zabbix item configuration page, items in Zabbix database

6. Now if you go to **Monitoring** | **Hosts** and click the **Latest data** for our host, you'll get to see this:



Latest data

Filter

Host groups	type here to search	Name	database
Hosts	lar-book-centos	Show items without data	☐
Application		Show details	☐

Apply | Reset

Host	Name ▲	Last check	Last value	Change	
lar-book-centos	**Database** (1 Item)				
	Number of items configured in Zabbix database	2020-07-02 13:11:57	3899		Graph

Displaying 1 of 1 found

0 selected | Display stacked graph | Display graph

Figure 2.42 – Zabbix Latest data page for host lar-book-centos, items in Zabbix database

We can now see directly from the database how many items are written to it.

How it works...

The Zabbix database monitoring works by connecting to your database with the ODBC middleware API. Any database supported by ODBC can be queried with Zabbix database monitoring:

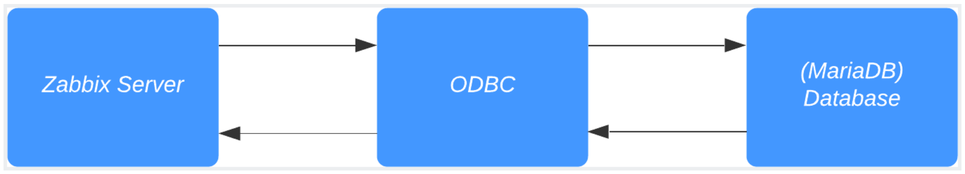


Figure 2.43 – Diagram showing Zabbix server ODBC communication

Basically, your Zabbix server sends a command with, for instance, your MySQL query to the ODBC connector. Your ODBC connecter sends this query to the database, which in turn returns a value to ODBC. ODBC then forwards the value to Zabbix server and hey presto: we have a value under our item.

There's more...

You can do loads of queries to your databases with Zabbix database monitoring, but keep in mind that you are working with actual queries. Querying a database takes time and processing power, so keep your database monitoring structured and define the right execution times.

Setting up HTTP agent monitoring

With the Zabbix HTTP agent, we can monitor a web page by retrieving data from it. For instance, if there's a counter on a web page and we want to keep an eye on that counter value, we can do so with Zabbix HTTP monitor.

Getting ready

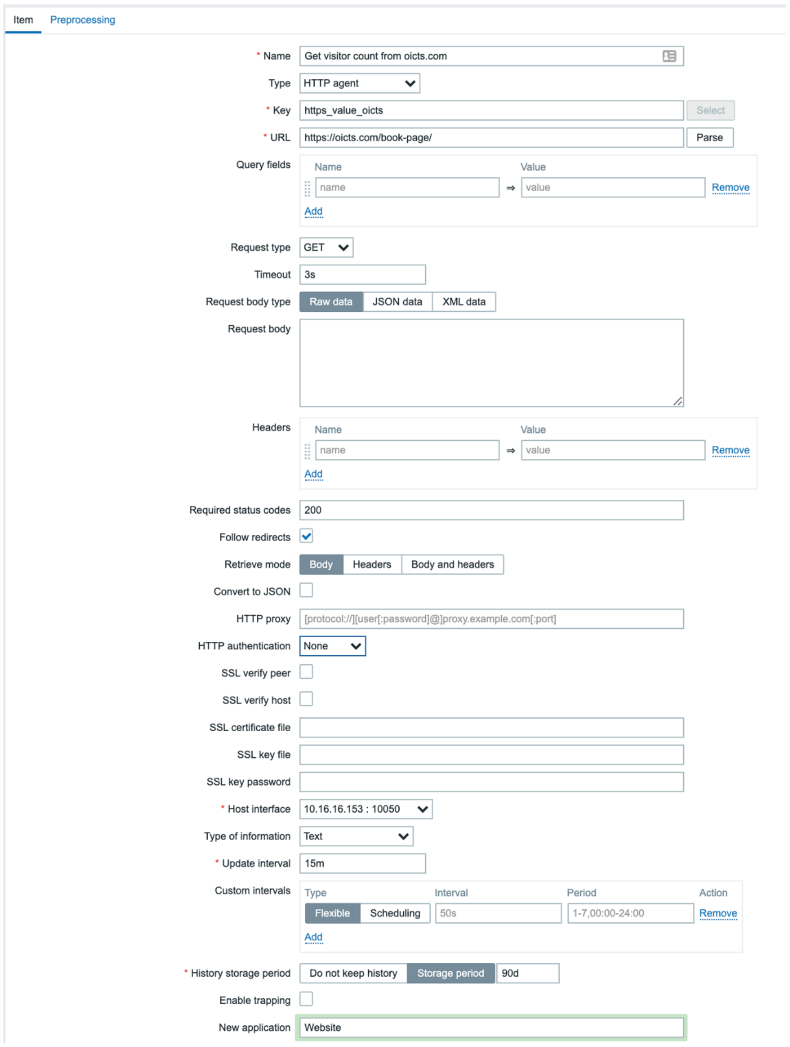
We are going to need a web page to monitor and we will need our Zabbix server. For your convenience, we've added a page to our own website to retrieve a value from. Here's the page: `https://oicts.com/book-page/`.

Please also note that your Zabbix server will need an active internet connection for this recipe.

How to do it...

Let's poll this special web page we've created for you for the visitor count that's currently configured on it. This is a real counter for the number of times (unique) visitors have opened the URL:

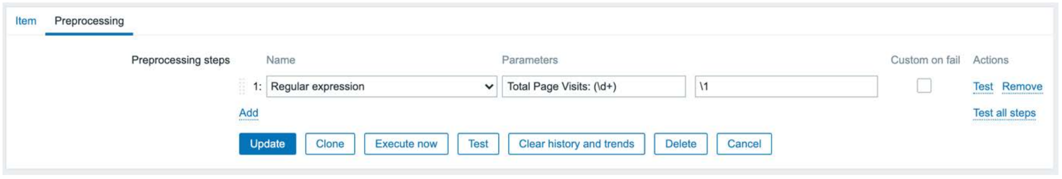
1. Navigate to your Zabbix frontend and navigate to **Configuration** | **Hosts**, then click the host called **lar-book-agent_simple**. Now go to **Items**; we want to create a new item here by clicking the **Create item** button. Now we are going to need to create an HTTP agent item as shown in the following screenshot:



* Name	Get visitor count from oicts.com
Type	HTTP agent
* Key	https_value_oicts
* URL	https://oicts.com/book-page/
Query fields	Name: name ⇒ Value: value
Request type	GET
Timeout	3s
Request body type	Raw data
Request body	
Headers	Name: name ⇒ Value: value
Required status codes	200
Follow redirects	✓
Retrieve mode	Body
Convert to JSON	
HTTP proxy	[protocol://][user[:password]@]proxy.example.com[:port]
HTTP authentication	None
SSL verify peer	
SSL verify host	
SSL certificate file	
SSL key file	
SSL key password	
* Host interface	10.16.16.153 : 10050
Type of information	Text
* Update interval	15m
Custom intervals	Type: Flexible; Interval: 50s; Period: 1-7,00:00-24:00
* History storage period	Storage period 90d
Enable trapping	
New application	Website

Figure 2.44 – Zabbix Item configuration page, visitor count on oicts.com page

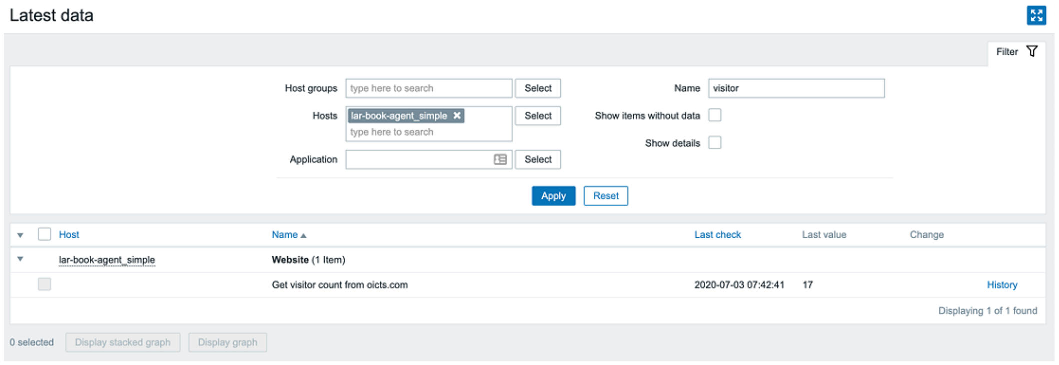
Use the following preprocessing steps:



Preprocessing steps	Name	Parameters		Custom on fail	Actions
1:	Regular expression	Total Page Visits: (\d+)	\1	☐	Test Remove

Figure 2.45 – Zabbix Item preprocessing page

2. Now navigate to **Monitoring | Hosts** and open the **Latest data** page for our **lar-book-agent_simple** host. If everything is working as it should, we should now be requesting this page's visitor count every 15 minutes as follows:



Host groups: type here to search
Hosts: lar-book-agent_simple
Application:
Name: visitor
Show items without data: ☐
Show details: ☐

Host	Name ▲	Last check	Last value	Change	
lar-book-agent_simple	**Website** (1 Item)				
	Get visitor count from oicts.com	2020-07-03 07:42:41	17		History

Displaying 1 of 1 found

Figure 2.46 – Zabbix Latest data page

How it works...

What we do here is request the complete web page from Zabbix by navigating to the page with the HTTP agent and downloading it. When we have the complete content of the page, in this case, an HTML/PHP page, we can process the data:

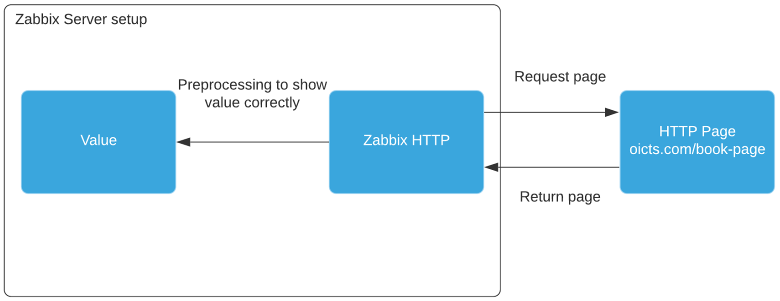

Zabbix Server setup
Preprocessing to show value correctly
Value
Zabbix HTTP
Request page
HTTP Page oicts.com/book-page
Return page

Figure 2.47 – Diagram showing Zabbix HTTP agent communication

We ask our preprocessor via a regex to go through the requested code and only show the number behind where it says **Total Page Visits:**.

All that's left is the number, ready for us to use in graphs and other types of data visualization.

Using Zabbix preprocessing to alter item values

Preprocessing item values is an important functionality in Zabbix; we can use it to create all kinds of checks. We've already done some preprocessing in this chapter, but let's take a deeper dive into it and what it does.

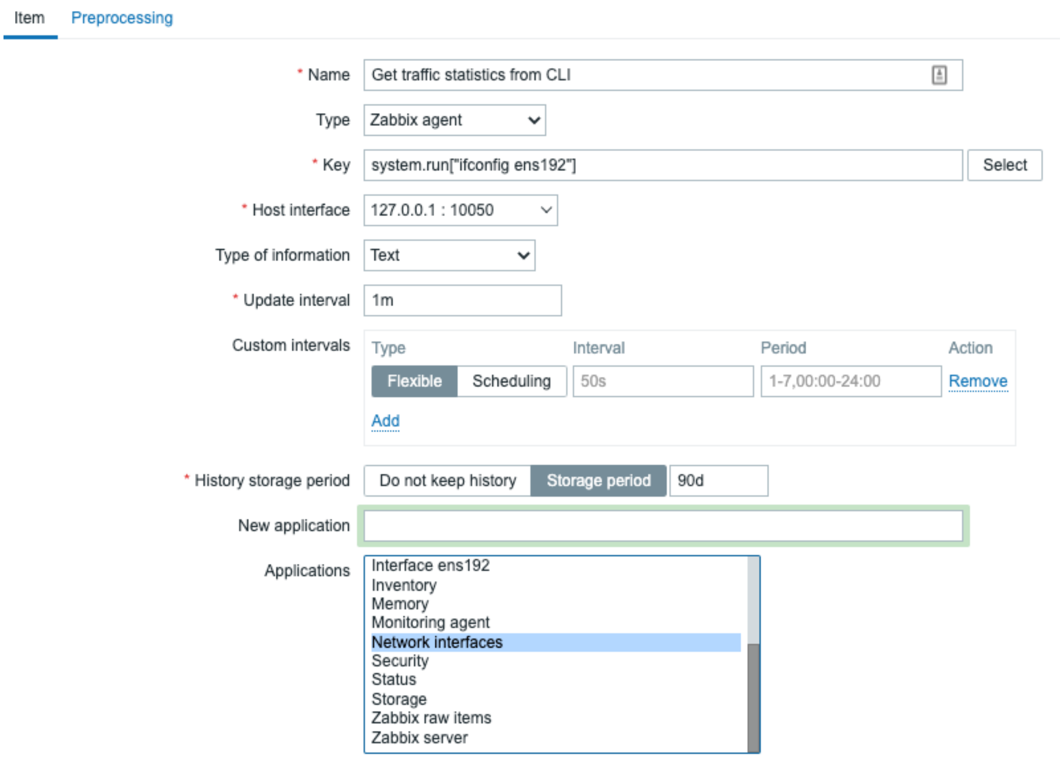
Getting started

We are going to need a Zabbix server to create our check for. We will also need a passive Zabbix agent on a Linux host to get our values from and preprocess them. We can use the agent that is running on our Zabbix server for this; in my case, this is **lar-book-centos**.

How to do it…

1. Let's start by logging in to our Zabbix frontend and going to **Configuration** | **Hosts**.
2. Click on your **Zabbix server** host; in my case, it's called **lar-book-centos**.

3. Now go to **Items** and click on the blue **Create item** button in the top-right corner. Let's create a new item with the following information:



Item | Preprocessing

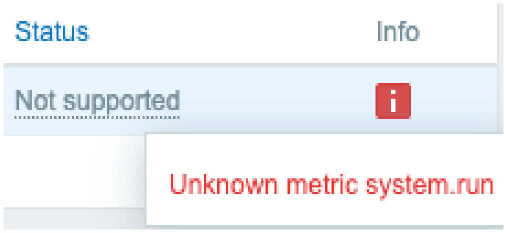
* Name	Get traffic statistics from CLI
Type	Zabbix agent
* Key	system.run["ifconfig ens192"]
* Host interface	127.0.0.1 : 10050
Type of information	Text
* Update interval	1m
Custom intervals	Type: Flexible / Scheduling; Interval: 50s; Period: 1-7,00:00-24:00; Action: Remove; Add
* History storage period	Do not keep history / Storage period: 90d
New application	
Applications	Interface ens192, Inventory, Memory, Monitoring agent, Network interfaces (selected), Security, Status, Storage, Zabbix raw items, Zabbix server

Figure 2.48 – New item creation screen, Get traffic statistics from CLI

4. Make sure to change `ens192` to your own primary network interface. You can find your primary network interface by logging in to the Linux CLI and executing the following:

```
ifconfig
```

5. Back at the create item screen, click on the blue **Add** button. This item will use the Zabbix agent to execute a remote command on the Linux CLI.
6. When we navigate to this new item, we can see that the item becomes unsupported. This is because when we use the `system.run` key, we need to allow it in the Zabbix agent configuration:



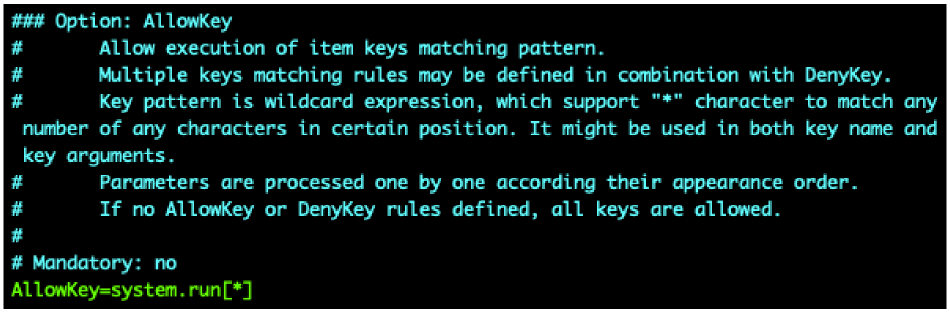
```
Unknown metric system.run
```

Figure 2.49 – Unsupported item information, Unknow metric system.run

7. Log in to the Linux CLI of the monitored host and edit the Zabbix agent configuration with this:

```
vim /etc/zabbix/zabbix_agent2.conf
```

8. Go to the **Option: AllowKey** line and add `AllowKey=system.run[*]` as here:



```
### Option: AllowKey
#       Allow execution of item keys matching pattern.
#       Multiple keys matching rules may be defined in combination with DenyKey.
#       Key pattern is wildcard expression, which support "*" character to match any
 number of any characters in certain position. It might be used in both key name and
 key arguments.
#       Parameters are processed one by one according their appearance order.
#       If no AllowKey or DenyKey rules defined, all keys are allowed.
#
# Mandatory: no
AllowKey=system.run[*]
```

Figure 2.50 – Zabbix agent configuration file, AllowKey system.run

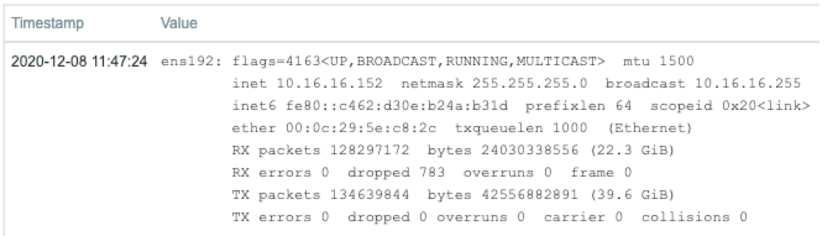
9. Save the file and restart the Zabbix agent with this:

```
systemctl restart zabbix-agent2
```

10. Back at the Zabbix frontend, the error we noticed in *step 6* should be gone after 1 minute.

11. Navigate to **Monitoring | Latest data** and filter on your Zabbix server host **lar-book-centos** and the name of the new **Get traffic statistics from CLI** item.

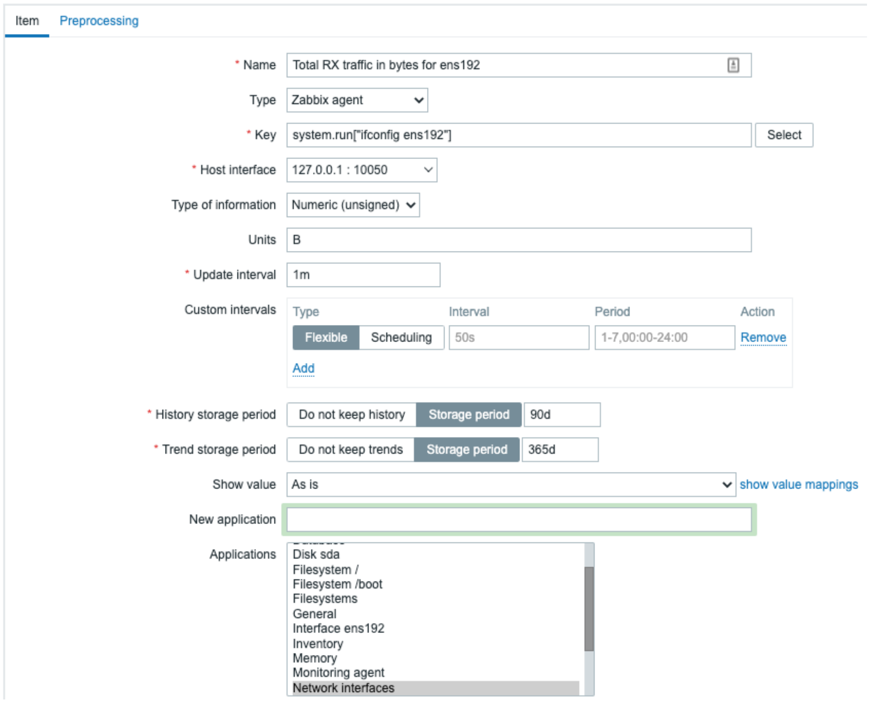
12. The value should now be pulled from the host. If we click on **History**, we can see the full value; it should look as follows:



```
Timestamp            Value
2020-12-08 11:47:24  ens192: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
                             inet 10.16.16.152  netmask 255.255.255.0  broadcast 10.16.16.255
                             inet6 fe80::c462:d30e:b24a:b31d  prefixlen 64  scopeid 0x20<link>
                             ether 00:0c:29:5e:c8:2c  txqueuelen 1000  (Ethernet)
                             RX packets 128297172  bytes 24030338556 (22.3 GiB)
                             RX errors 0  dropped 783  overruns 0  frame 0
                             TX packets 134639844  bytes 42556882891 (39.6 GiB)
                             TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 2.51 – Zabbix agent system.run command executing 'ifconfig ens192' results

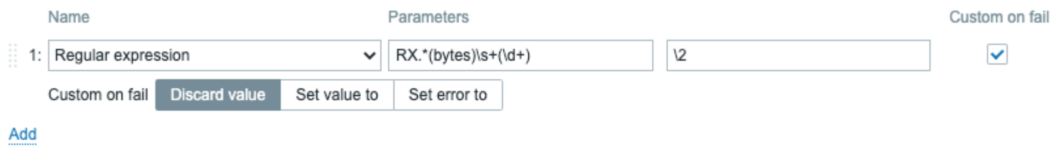
13. The information seen in the image is way too much for just one item. We need to split this up. Let's use pre-processing to get the number of RX bytes from the information.
14. Go back to **Configuration | Hosts** and click on your Zabbix server host. Go to **Items** on this host.
15. Click on the **Get traffic statistics from CLI** item to edit it. Change the name to `Total RX traffic in bytes for ens192` and add `B` to **Units**. It will look like this:



* Name	Total RX traffic in bytes for ens192
Type	Zabbix agent
* Key	system.run["ifconfig ens192"]
* Host interface	127.0.0.1 : 10050
Type of information	Numeric (unsigned)
Units	B
* Update interval	1m
Custom intervals	Type: Flexible; Interval: 50s; Period: 1-7,00:00-24:00; Action: Remove
* History storage period	Storage period 90d
* Trend storage period	Storage period 365d
Show value	As is
New application	
Applications	Disk sda, Filesystem /, Filesystem /boot, Filesystems, General, Interface ens192, Inventory, Memory, Monitoring agent, Network interfaces

Figure 2.52 – Zabbix agent system.run item

16. Now click on **Preprocessing** and click on the underlined **Add** button.
17. A **Regular expression** (regex) field will be added, which we are going to fill to match the total number of bytes for your interface. Fill in the following:

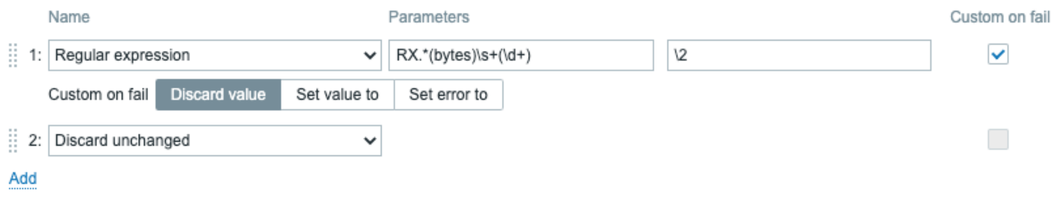


Name	Parameters		Custom on fail
1: Regular expression	RX.*(bytes)\s+(\d+)	\2	✓

Custom on fail: Discard value | Set value to | Set error to

Figure 2.53 – Zabbix agent system.run item preprocessing

18. Make sure to also select the box under **Custom on fail**.
19. Let's click on the underlined **Add** button again and use the drop-down menu for this new step to select **Discard unchanged**. The end result will look like this:

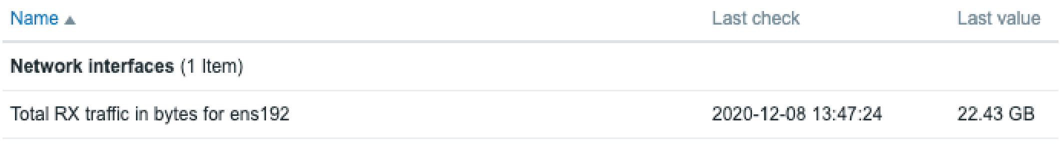


Name	Parameters		Custom on fail
1: Regular expression	RX.*(bytes)\s+(\d+)	\2	✓
2: Discard unchanged			

Custom on fail: Discard value | Set value to | Set error to

Figure 2.54 – Zabbix agent system.run item preprocessing

20. We can now press the blue **Update** button to finish editing this item.
21. Navigate back to **Monitoring | Latest data and filter** on your host and the new item name, **Total RX traffic in bytes for ens192**. Make sure to use your own interface name.
22. We can now see our value coming in, and we have an item displaying our total RX traffic for our main interface:



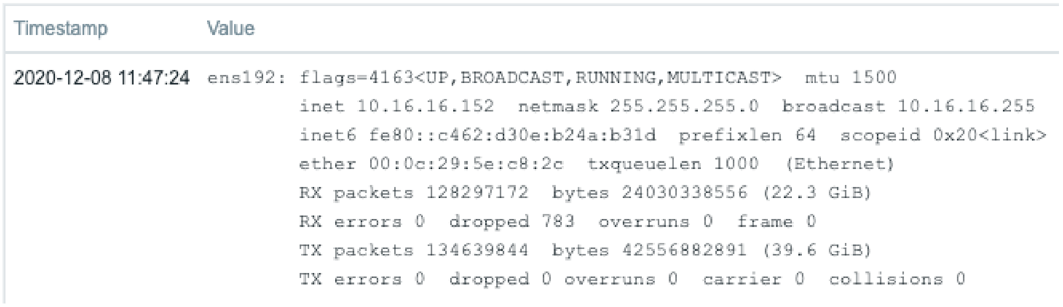
Name ▲	Last check	Last value
Network interfaces (1 Item)		
Total RX traffic in bytes for ens192	2020-12-08 13:47:24	22.43 GB

Figure 2.55 – Zabbix Total RX traffic item latest data

How it works...

We've already done some preprocessing in the *Working with calculated and dependent items* recipe to get data from a master item. We also used preprocessing in the *Setting up HTTP agent monitoring* recipe to get a specific value from a web page. We didn't go over the preprocessing concepts used in those recipes, though, so let's go over them.

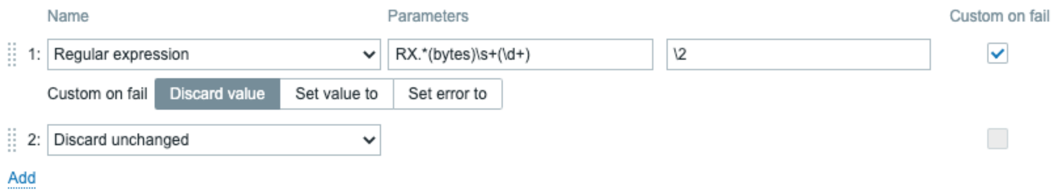
When working with preprocessing, it's important to know the basic setup. Let's take a look at the incoming data before we used preprocessing:



```
Timestamp            Value

2020-12-08 11:47:24  ens192: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
                             inet 10.16.16.152  netmask 255.255.255.0  broadcast 10.16.16.255
                             inet6 fe80::c462:d30e:b24a:b31d  prefixlen 64  scopeid 0x20<link>
                             ether 00:0c:29:5e:c8:2c  txqueuelen 1000  (Ethernet)
                             RX packets 128297172  bytes 24030338556 (22.3 GiB)
                             RX errors 0  dropped 783  overruns 0  frame 0
                             TX packets 134639844  bytes 42556882891 (39.6 GiB)
                             TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 2.56 – Zabbix agent system.run command executing 'ifconfig ens192' results

This is a lot of information. When we look at how Zabbix items are used, we try to put graspable information in a single item. Luckily, we can preprocess this item before we store the value in Zabbix. In the following figure, we can see the preprocessing steps we added to our item:



	Name	Parameters		Custom on fail
1:	Regular expression	RX.*(bytes)\s+(\d+)	\2	✓
	Custom on fail: [✓ Discard value] Set value to / Set error to			
2:	Discard unchanged			

Figure 2.57 – Zabbix agent system.run item preprocessing with 2 steps

Our first step is a regex. This step will make sure to only use the numbers we need. We match on the word **RX**, then the word **bytes** and a sequence of numbers after them. This way, we end up with the total number of RX bytes in capture group 2. This is why we fill in \2 in the **output** field. We also specify **Custom on fail**, which will discard any value if the regex doesn't match.

Our second step is to discard any values that are the same as the value received before. Instead of storing duplicate values, we simply discard them and save some space in our Zabbix database.

Tip

It's a lot easier to build a regex when using an online tool such as `https://regex101.com/`. You can see what number your capture groups will get and there's a lot of valuable information in the tools as well.

It's important to note here that steps are executed in the sequence they are defined in the frontend. If the first step fails, the item becomes unsupported unless **Custom on fail** is set to do something else.

By adding preprocessing to Zabbix, we open up a whole range of options for our items, and we are able to alter our data in almost any way required. These two steps are just the beginning of the options opened up when diving into the world of Zabbix preprocessing.

See also

Preprocessing in Zabbix is an important subject and it's impossible to cover every aspect of it in a single recipe. The two preprocessing steps in this recipe's example are just two of the many options we can use. Check out the official Zabbix documentation to see the other options we can use:

`https://www.zabbix.com/documentation/current/manual/config/items/preprocessing`

3 Working with Triggers and Alerts

Now, what use would all of that collected data in Zabbix be without actually doing some alerting with it? Of course, we can use Zabbix to collect our data and just go over it, but it's way better when we actually start sending out notifications to users. This way, we don't have to always keep an eye on our Zabbix frontend, but we can just let our triggers and alerts do the work for us, redirecting us to the frontend only when we need it.

We will learn all about setting up effective triggers and alerts in the following recipes:

- Setting up triggers
- Setting up alerts
- Keeping alerts effective
- Customizing alerts

Technical requirements

For this chapter, we will need a Zabbix server—for instance, the one used in the previous chapter, which would be the following:

- A server with Zabbix server installed on a Linux distribution of your choice, such as CentOS or Ubuntu, but a distribution such as Debian or any other will suit you just as well
- MariaDB set up to work with your Zabbix server
- Nginx or Apache set up to serve the Zabbix frontend

We will also need a Linux host to monitor, so that we can actually build some cool triggers to use.

Setting up triggers

Triggers are important in Zabbix because they notify you as to what's going on with your data. We want to get a trigger when our data reaches a certain threshold.

So, let's get started with setting up some cool triggers. There are loads of different options for defining triggers, but after reading this recipe you should be able to set up some of the most prominent triggers. Let's take your trigger experience to the next level!

Getting ready

For this recipe, we will need our Zabbix server ready and we will need a Linux host. I will use the `lar-book-agent_simple` host from the previous chapter because we already have some items on that.

We'll also need one more host that is monitored by Zabbix agent with the Zabbix agent template. We'll use one of the items on this host to create a trigger. This will be the `lar-book-agent_passive` host from the previous chapter.

On this host, we will already have some triggers available, but we will extend these triggers further to inform us even better.

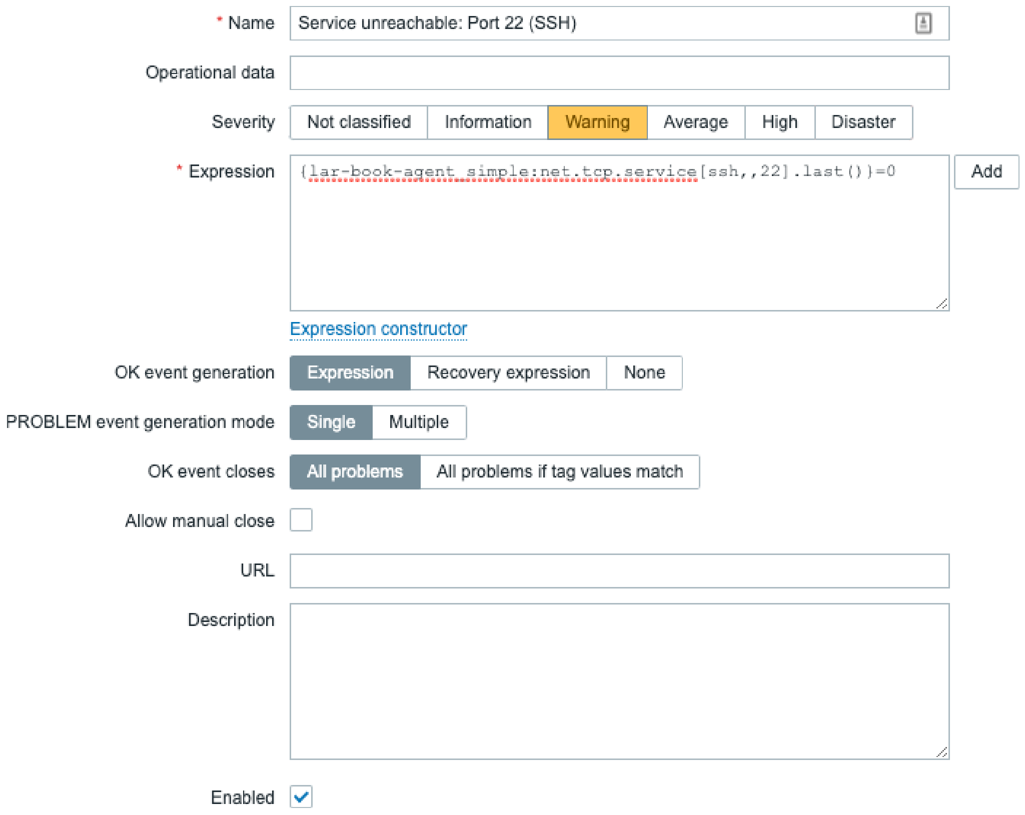
How to do it...

In this section, we are going to create three triggers to monitor state changes. Let's get started by creating our first trigger.

Trigger 1 – SSH service monitoring

Let's create a simple trigger on the `lar-book-agent_simple` host. We made a simple check on this host called `Check if port 22 is available`, but we haven't created anything to notify us on this yet:

1. First, let's get started by going to **Configuration** | **Hosts**, then clicking the host and going to **Triggers**. This is where we will find our triggers and where we can create them. We want to create a new trigger here by clicking the **Create trigger** button.
2. Let's create a new trigger with the following information:



* Name	Service unreachable: Port 22 (SSH)
Operational data	
Severity	Not classified / Information / [✓ Warning] / Average / High / Disaster
* Expression	{lar-book-agent_simple:net.tcp.service[ssh,,22].last()}=0
OK event generation	[✓ Expression] / Recovery expression / None
PROBLEM event generation mode	[✓ Single] / Multiple
OK event closes	[✓ All problems] / All problems if tag values match
Allow manual close	☐
URL	
Description	
Enabled	✓

Figure 3. 1 – The Zabbix trigger creation page, Service unreachable

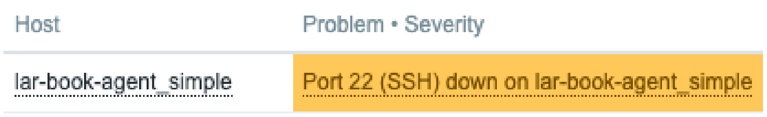
3. Click on **Add** and finish creating the trigger. This will create a trigger for us that will fire when our **Secure Shell** (**SSH**) port goes down.

4. Let's test this by navigating to our host **command-line interface** (**CLI**) and executing some commands to shut our Zabbix server off from port `22`. We will add an `iptables` rule to block off all incoming traffic on port `22` (SSH):

```
iptables -A INPUT -p tcp -i ens192 -s 10.16.16.152
--destination-port 22 -j DROP
```

```
service iptables save
```

5. Now, if we check out **Monitoring** | **Dashboard**, after a while we should see the following:



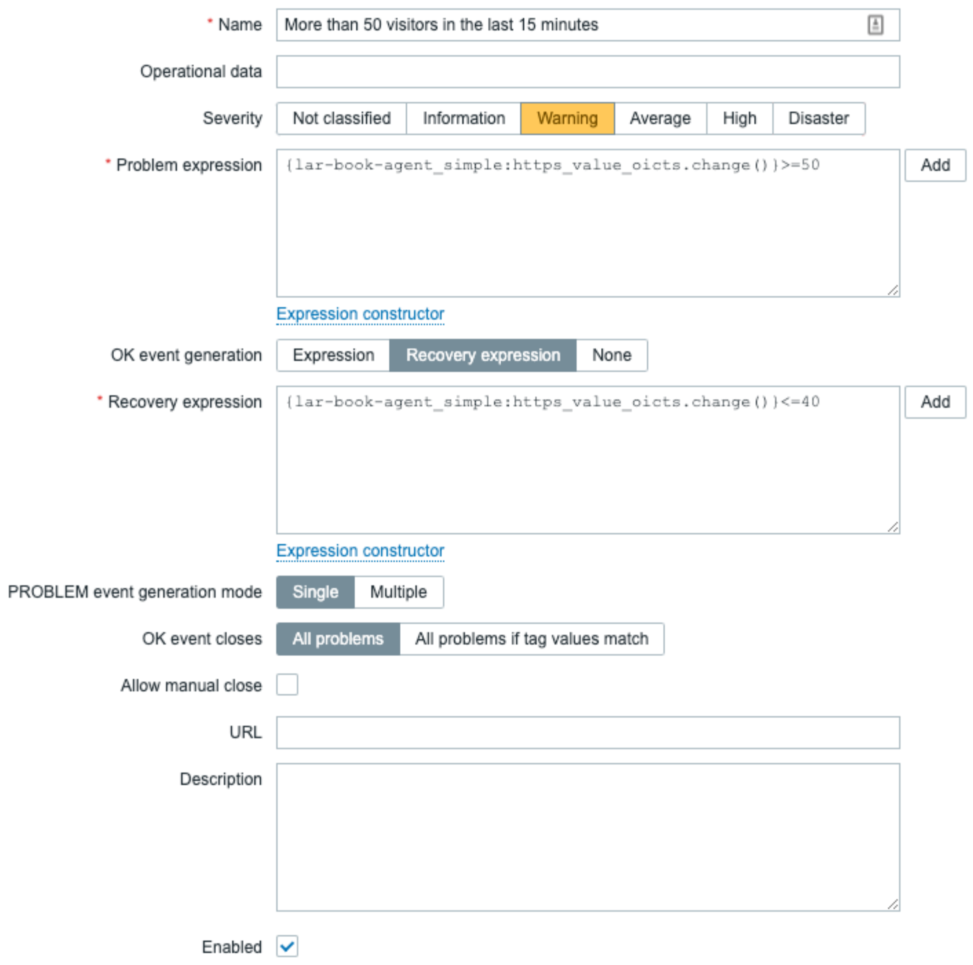
Host	Problem • Severity
lar-book-agent_simple	Port 22 (SSH) down on lar-book-agent_simple

Figure 3.2 – The Zabbix problem page, port 22 down

Trigger 2 – Triggering on page visitor count

Now, for creating our second trigger, let's ramp it up a bit. If you followed *Chapter 2*, *Setting up Zabbix Monitoring*, in the recipe titled *Setting up HTTP agent monitoring* we created an item that polls one of our website pages for its visitor count. Now, what we probably want to do ourselves is keep an eye out for how well all of our readers are doing reaching the web page part of the book and building an item for it:

1. Let's navigate to **Configuration** | **Hosts** and click the `lar-book-agent_simple` host.
2. Now, go to **Triggers** and click the **Create trigger** button. We will build our trigger with the following settings:



* Name	More than 50 visitors in the last 15 minutes
Operational data	
Severity	Not classified / Information / [✓ Warning] / Average / High / Disaster
* Problem expression	{lar-book-agent_simple:https_value_oicts.change()}>=50
OK event generation	Expression / [✓ Recovery expression] / None
* Recovery expression	{lar-book-agent_simple:https_value_oicts.change()}<=40
PROBLEM event generation mode	[✓ Single] / Multiple
OK event closes	[✓ All problems] / All problems if tag values match
Allow manual close	☐
URL	
Description	
Enabled	✓

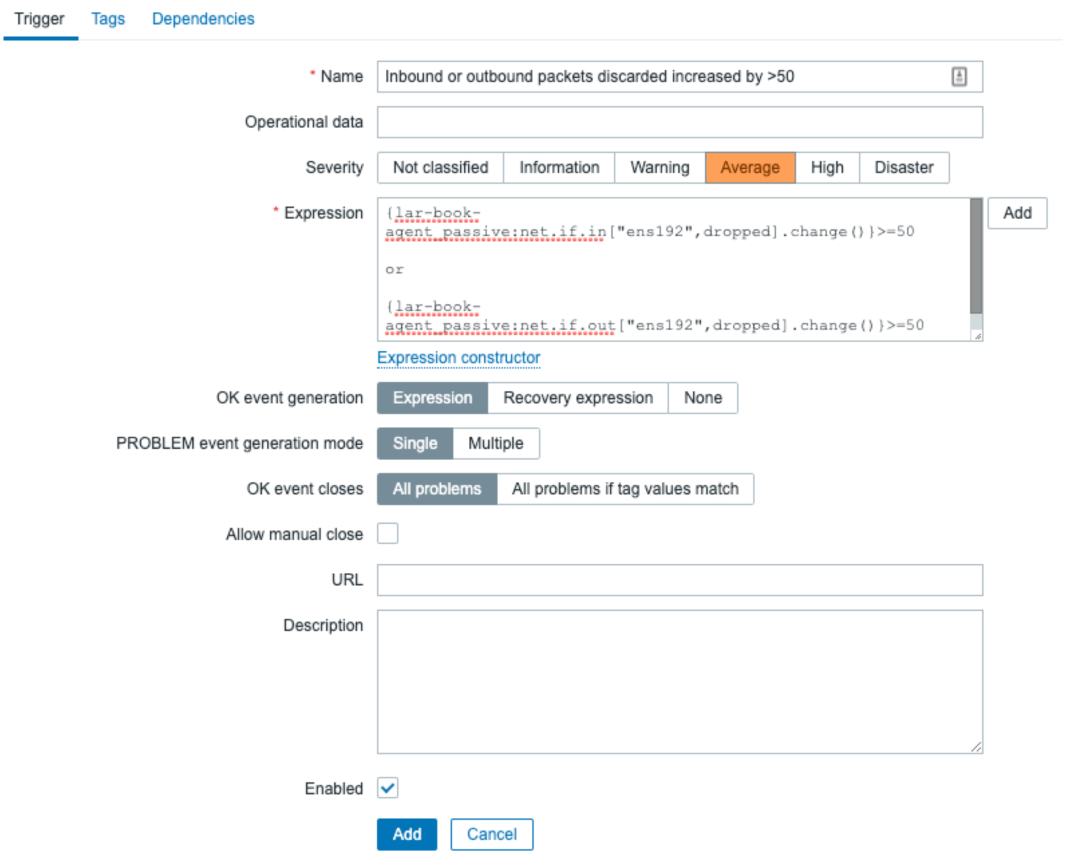
Figure 3.3 – The Zabbix trigger creation page, More than 50 visitors trigger

3. Click on **Add** and finish creating the trigger.
4. Now, this might not actually trigger for you in the frontend, but I'll explain to you just how this trigger works in the *How it works…* section of this recipe.

Trigger 3 – Using multiple items in a trigger

We have seen triggers that use one item, but we can also use multiple items in a single trigger. Let's build a new trigger by using multiple items in the same expression:

1. Let's navigate to **Configuration** | **Hosts** and click the `lar-book-agent_passive` host. Now, go to **Triggers** and click the **Create trigger** button.
2. We are going to create a trigger with the following settings:



* Name	Inbound or outbound packets discarded increased by >50
Operational data	
Severity	Not classified / Information / Warning / [✓ Average] / High / Disaster
* Expression	{lar-book-agent_passive:net.if.in["ens192",dropped].change()}>=50 or {lar-book-agent_passive:net.if.out["ens192",dropped].change()}>=50
OK event generation	[✓ Expression] / Recovery expression / None
PROBLEM event generation mode	[✓ Single] / Multiple
OK event closes	[✓ All problems] / All problems if tag values match
Allow manual close	☐
URL	
Description	
Enabled	✓

Figure 3.4 – The Zabbix trigger creation page, inbound or outbound packets trigger

3. Please note that your items might actually have different interface names. In my case, the interface is called `ens192`, so use the correct name for your interface in its place.
4. Click on **Add** and finish creating the trigger.

Tip

Use the **Add** button next to the **Expression** field to select an item and build your expression easily. To understand expressions better, there's an explanation for every trigger function included in the **Expression constructor** on the trigger creation page.

5. That's all we need to do to build a trigger that will function on two items.

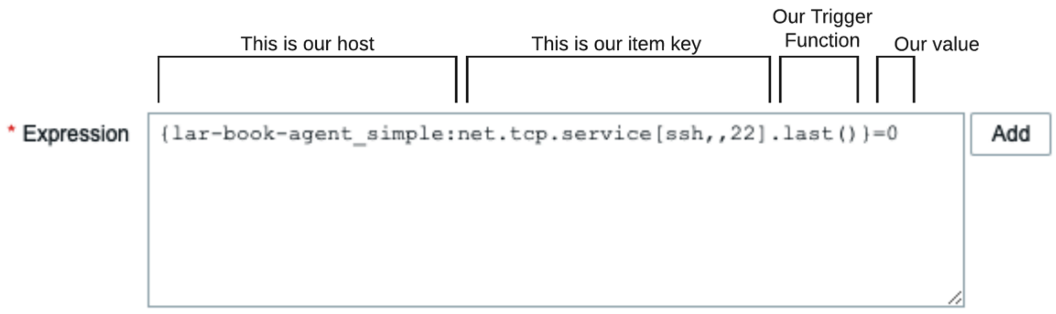
How it works...

We need a good understanding of how to build triggers and how they work, so we can create a well-set-up monitoring system. Let's discover why we built our triggers like we did.

Trigger 1 – SSH service monitoring

This is a very simple but effective trigger to set up in Zabbix. When our value returns us either a `1` for `UP` or a `0` for `DOWN`, we can easily create triggers such as these—not just for monitoring logical ports that are UP or DOWN, but for everything that returns us a simple value change from, for example, `1` to `0` and vice versa.

Now, if we break down our expression, we will see the following:



```
{lar-book-agent_simple:net.tcp.service[ssh,,22].last()}=0
```

Figure 3.5 – A Zabbix trigger expression, port 22 (SSH)

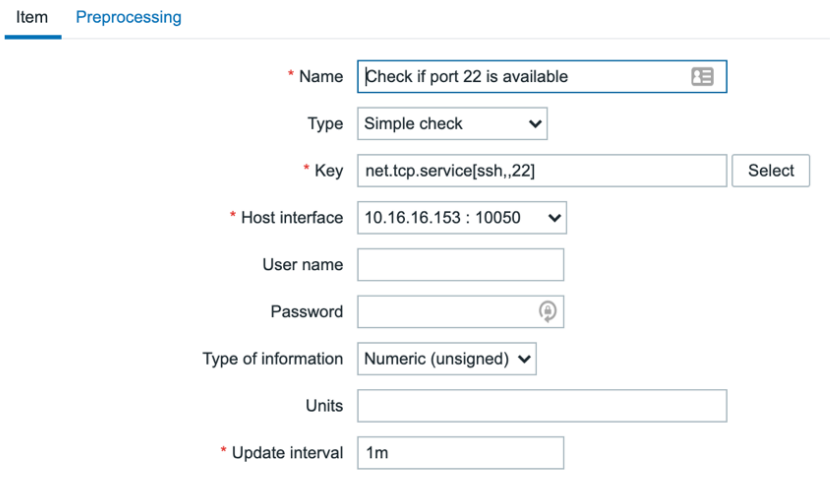
When building an expression, we have four parts:

- **Host**: The host part of the expression is where we define which host we are using to trigger on. Most of the time, it's simply just the host we are working on.
- **Item key**: The item key is the part of the expression where we define which item key we'll be using to retrieve a value on a host.

- **Trigger function**: The trigger function is the part of the expression that determines what we expect of the value, such as whether we want just the value or an average value over a period of time.
- **Value**: The value is the actual trigger value that our trigger function uses to determine whether the trigger should be in an `OK` or a `PROBLEM` state.

Now, for our first trigger, we defined our host and the item that gives us the SSH status. What we are asking in the trigger function is that we want the last value to be 0 before triggering it.

For this item, that would mean it would trigger within a minute, because in our item we specified the following:



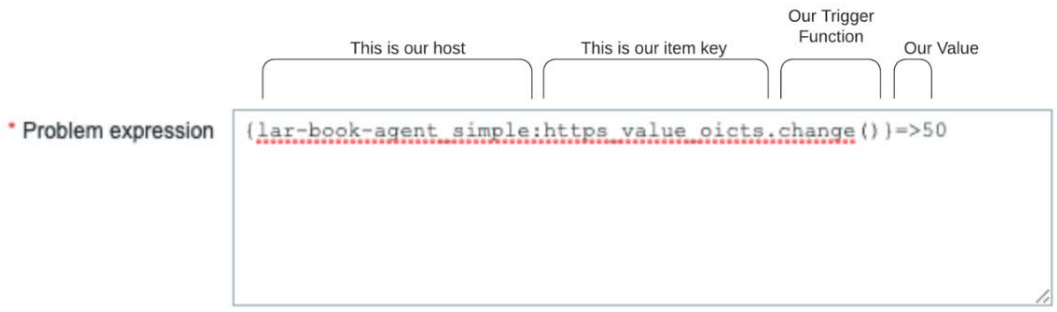
* Name	Check if port 22 is available
Type	Simple check
* Key	net.tcp.service[ssh,,22]
* Host interface	10.16.16.153 : 10050
User name	
Password	
Type of information	Numeric (unsigned)
Units	
* Update interval	1m

Figure 3.6 – The Zabbix trigger creation page, port 22 availability trigger

Looking at the **Update interval** field, we can determine that when building this trigger, we are expecting our value to be `0` and that will take at a maximum 1 minute of SSH port `22` downtime due to the 1-minute interval.

Trigger 2 – Triggering on page visitor count

Now, for our second trigger, we did something different. We not only made an expression for triggering this problem, but also one for recovering from the trigger. What we do in the **Problem expression** option is define a trigger function, telling our host to compare the last value with the latest newest value. We then tell it this function has to be `=>50`, meaning equal to or higher than 50:

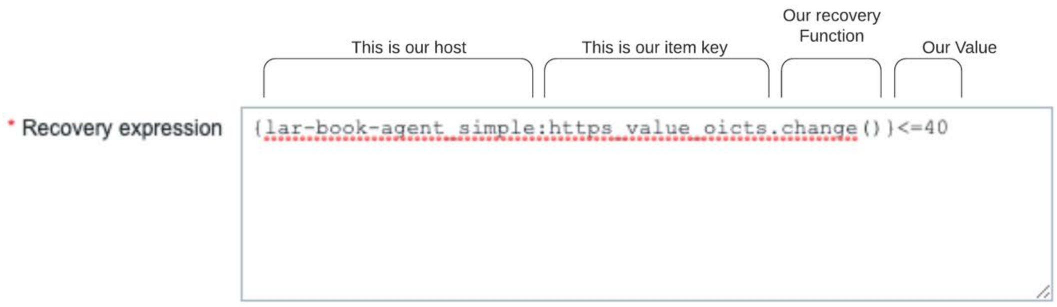


```
{lar-book-agent_simple:https_value_oicts.change()}=>50
```

Figure 3.7 – A Zabbix trigger expression, HTTPS check

So, our trigger will only be activated when we have more than 50 visitors between the last and previous value for this specific page. Now, we could do the same with the previous trigger and let it recover once it hits the same value again, but the other way around. This means that, for this trigger, it would recover once our visitor count between the last and previous value drops below 50 again. But I want to keep this trigger in the PROBLEM state just a little longer.

Therefore, I defined a recovery expression as well. I'm telling it that this problem can only recover if the visitor count between the last and previous value has dropped below or was equal to 40. Check out the recovery expression up close:



```
{lar-book-agent_simple:https_value_oicts.change()}<=40
```

Figure 3.8 – Another Zabbix trigger expression, HTTPS check with different value

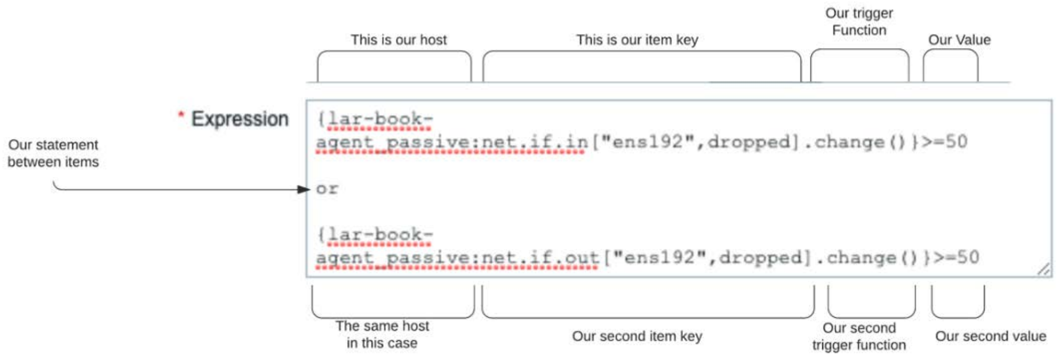
Recovery expressions are powerful when you want to extend your trigger functionality with just a bit more control over when it comes back into the OK state.

Tip

You can use the recovery expression for extending the trigger's `PROBLEM` state beyond what you defined in the **Problem expression** option. This way, we know we are still close to the `PROBLEM` state and only go back to the `OK` state after we've really moved away from the trigger value.

Trigger 3 – Using multiple items in a trigger

Now, trigger 3 might seem complicated because we've used more than one item, but it's basically the same setup:



```
{lar-book-
agent_passive:net.if.in["ens192",dropped].change()}>=50

or

{lar-book-
agent_passive:net.if.out["ens192",dropped].change()}>=50
```

Figure 3.9 – A Zabbix trigger expression using several items

We have the same setup for the expression, with host/item/function/value. Yet when we are working with multiple items, we can add an `or` statement between the items. This way, we can say we need to match one of the items before triggering the PROBLEM state. In this case, we trigger when either item reaches above the threshold.

Important note

In this trigger expression, we have some empty lines between the different item expressions. Empty lines between item expressions are totally fine and actually make for good readability. Use this wisely when building triggers.

There's more...

Not only can we match on one of the items in a trigger expression—we can also do an `and` statement. This way, you can make sure our trigger only goes into a `PROBLEM` state when multiple items are reaching a certain value.

Triggers are very powerful like this, allowing us to define our own criteria in great detail. There's no predefinition—we can add as many `and/or` statements and different functions as we like in the trigger expressions. Customize your triggers to exactly what you need, and suddenly you are going to have a lot more peace of mind because you know your triggers will notify you when something is up.

See also

To know more about trigger expressions, check out the Zabbix documentation There's a lot of information on which functions you can use to build the perfect trigger. For more details, go to `https://www.zabbix.com/documentation/current/manual/config/triggers/expression`.

Setting up alerts

Alerting can be a very important part of your Zabbix setup. When we set up alerts, we want the person on the other end to be informed of just what is going on. It's also important to not spam someone with alerts; we want it to be effective.

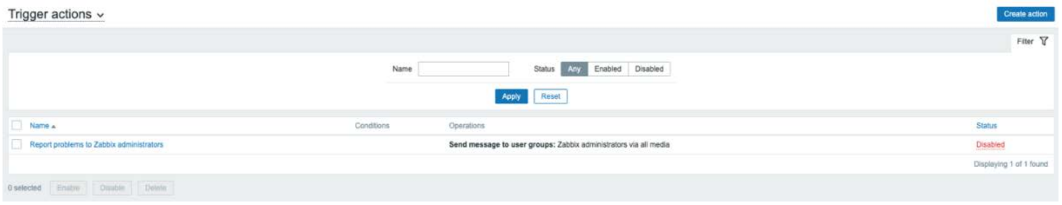
So, in this recipe, we will go over the basics of setting up alerts, so we know just how to get it right from the start.

Getting ready

For this recipe, we will only need two things. We will have to use our Zabbix server to create our alerts and we will need our triggers from the previous recipe. The triggers will be used to initiate the alerting process to see just how Zabbix server will convey this information.

How to do it...

1. Let's start by setting up our action in the Zabbix frontend. To do this, we will navigate to **Configuration | Actions** and we will be served with this screen:



Name Status [✓ Any] Enabled Disabled

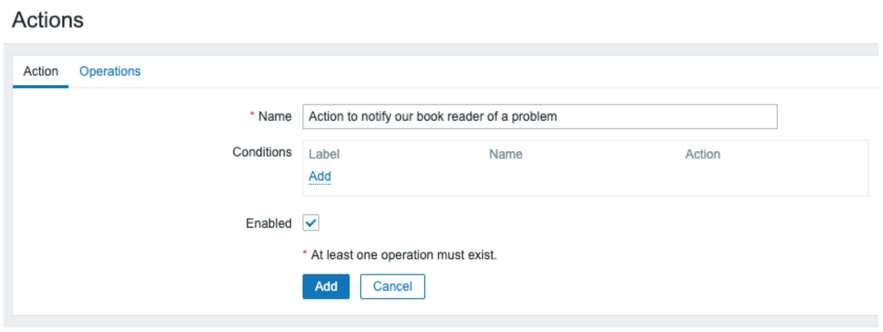
	Name ▲	Conditions	Operations	Status
☐	Report problems to Zabbix administrators		**Send message to user groups:** Zabbix administrators via all media	Disabled

Displaying 1 of 1 found

Figure 3.10 – The Zabbix Trigger actions page with 1 trigger action

There already is one action set up to notify **Zabbix administrators** of problem events. In Zabbix 5, a lot of features such as **Actions** and **Media** are predefined. Most of the time, all we need to do is enable them and fill out some information.

2. We will set up our own action, so let's create a new action to notify a user in the **Zabbix administrators** group of our new triggers. Click the blue **Create action** button to be taken to the next page:



Actions

Action | Operations

* Name: Action to notify our book reader of a problem

Conditions: Label | Name | Action — Add

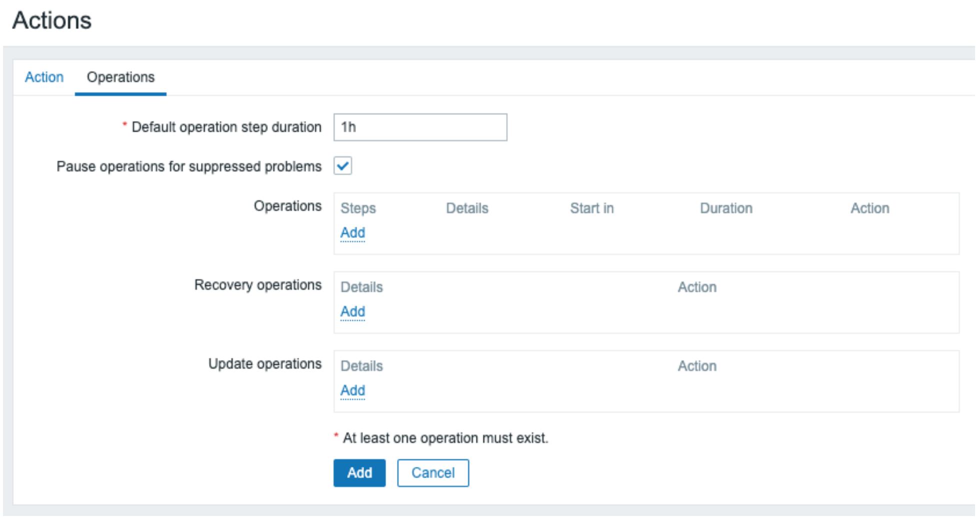
Enabled: ✓

* At least one operation must exist.

Add | Cancel

Figure 3.11 – The Zabbix Action creation page, Notify book reader

3. On this page, we are going to want to check the **Enabled** checkmark to make sure that this action will actually do something. Make sure to name your action clearly so that you won't have any issues differentiating between actions.
4. Now, move on to our **Operations** tab:



Actions

Action | Operations

* Default operation step duration: 1h

Pause operations for suppressed problems: ✓

Operations: Steps | Details | Start in | Duration | Action — Add

Recovery operations: Details | Action — Add

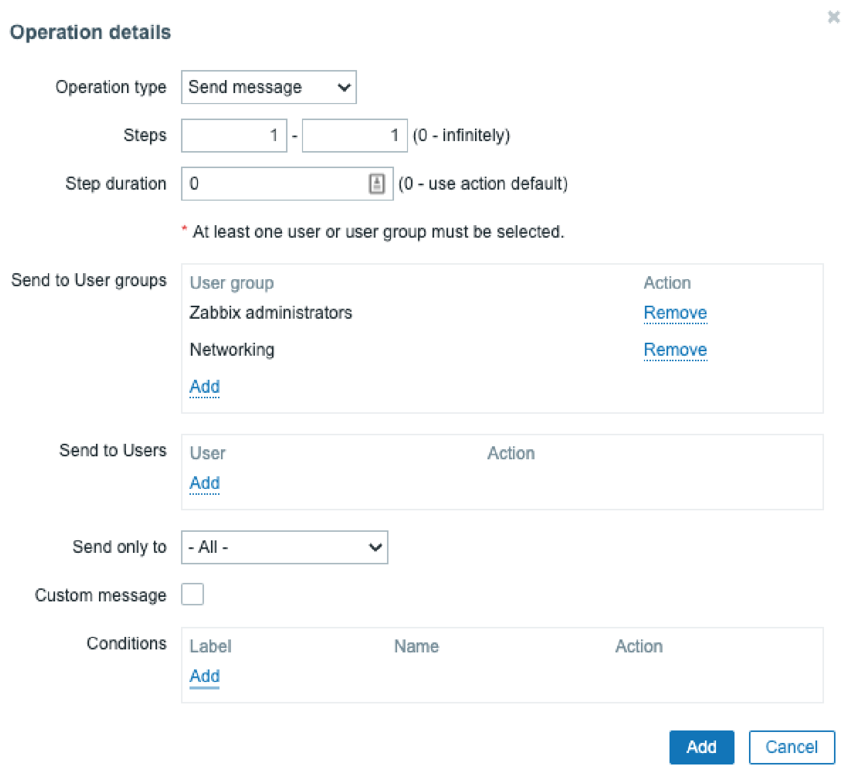
Update operations: Details | Action — Add

* At least one operation must exist.

Add | Cancel

Figure 3.12 – The Zabbix Action creation page at the Operations tab, Notify book reader

5. The **Operations** tab is empty by default, so we are going to want to create some operations here. There are three forms of operation that we are going to create here—let's start with the **Operations** operation by clicking **Add**:



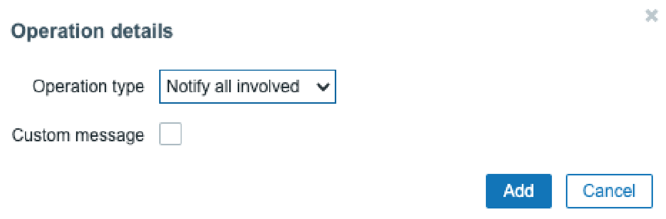
Operation details

Operation type	Send message
Steps	1 - 1 (0 - infinitely)
Step duration	0 (0 - use action default)
	* At least one user or user group must be selected.
Send to User groups	User group: Zabbix administrators, Networking
Send to Users	
Send only to	- All -
Custom message	☐
Conditions	

Figure 3.13 – Zabbix Operation details page, Notify book reader

6. We have the option to add **Users** and **User groups** here that we want to alert. If you've followed along with *Chapter 1*, *Getting Started with Zabbix and User Management*, we can just select the **Networking** user group here. If not, selecting just your **Zabbix administrators** group is fine.
7. After clicking the blue **Add** button, we will be taken back to the **Actions** screen.

8. Now, we will create the next operation, named **Recovery operations**. What we do here is create an operation, as follows:



Operation details

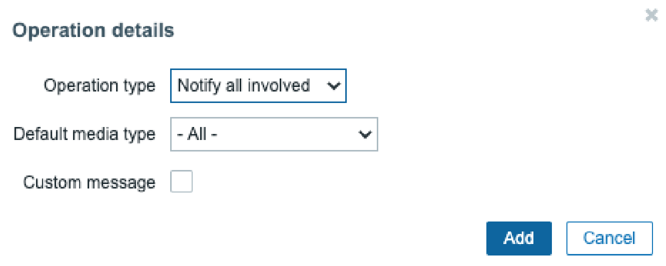
Operation type	Notify all involved
Custom message	☐

Figure 3.14 – Zabbix recovery Operation details, Notify book reader

9. This option will notify all users involved in the initial operation defined earlier. All users that got a PROBLEM generation notification will also receive the recovery this way. Click **Add**, and let's continue.

   Now, if you're like me and you want to stay on top of things, you are able to create an update notification. This way, we know that—for instance—someone acknowledged a problem and is working on it. Normally, I would select different channels for stuff such as this—for instance, using **SMS** for high-priority alerts and a Slack or Teams channel for everything else.

10. Let's click **Add** under **Update operations** to add the following to our setup:



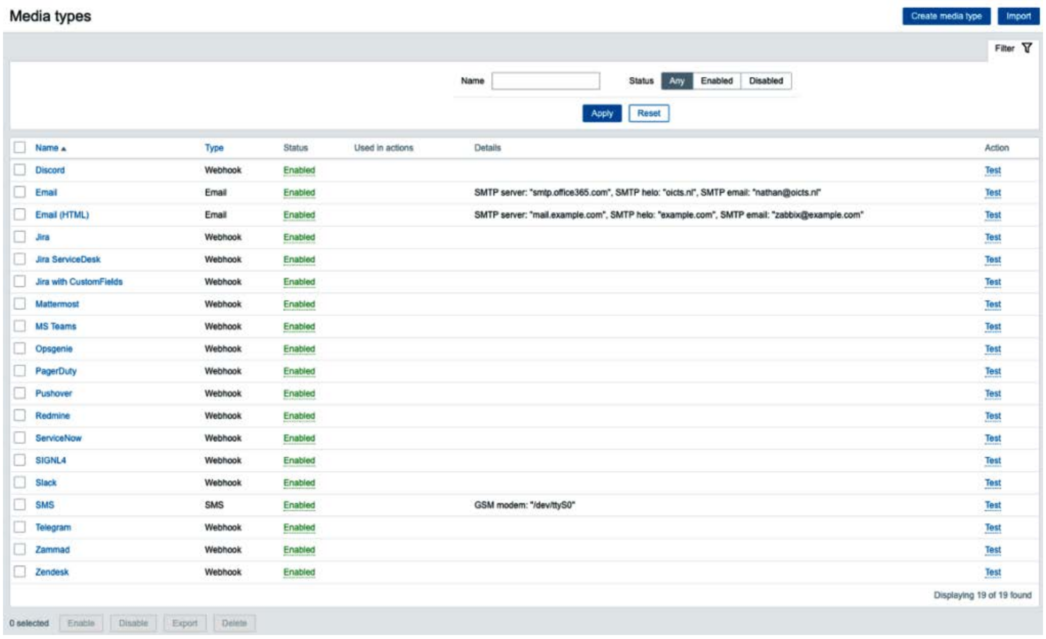
Operation details

Operation type	Notify all involved
Default media type	- All -
Custom message	☐

Figure 3.15 – Zabbix Update Operation details, Notify book reader

11. We will do the same thing here as we did for the **Recovery operations** option. Notify all users involved of any update to this problem. After clicking **Add** here, click the blue **Add** button again in our **Actions** screen to finish creating the action.

12. Now, the next thing we want to do is create a media type for actually notifying our users of the issue. Go to **Administration | Media types**, and you will be presented with the following screen:



Name ▲	Type	Status	Used in actions	Details	Action
Discord	Webhook	Enabled			Test
Email	Email	Enabled		SMTP server: "smtp.office365.com", SMTP helo: "oicts.nl", SMTP email: "nathan@oicts.nl"	Test
Email (HTML)	Email	Enabled		SMTP server: "mail.example.com", SMTP helo: "example.com", SMTP email: "zabbix@example.com"	Test
Jira	Webhook	Enabled			Test
Jira ServiceDesk	Webhook	Enabled			Test
Jira with CustomFields	Webhook	Enabled			Test
Mattermost	Webhook	Enabled			Test
MS Teams	Webhook	Enabled			Test
Opsgenie	Webhook	Enabled			Test
PagerDuty	Webhook	Enabled			Test
Pushover	Webhook	Enabled			Test
Redmine	Webhook	Enabled			Test
ServiceNow	Webhook	Enabled			Test
SIGNL4	Webhook	Enabled			Test
Slack	Webhook	Enabled			Test
SMS	SMS	Enabled		GSM modem: "/dev/ttyS0"	Test
Telegram	Webhook	Enabled			Test
Zammad	Webhook	Enabled			Test
Zendesk	Webhook	Enabled			Test

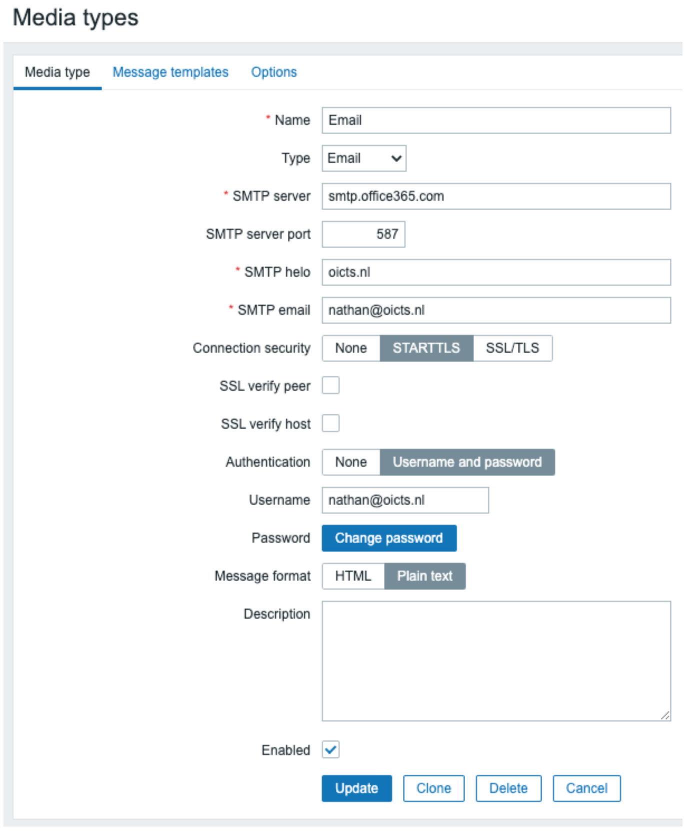
Displaying 19 of 19 found

0 selected Enable Disable Export Delete

Figure 3.16 – The Zabbix Media types page with predefined media options

As you can see, there are quite a lot of predefined media types in Zabbix 5. We have them for Slack, Opsgenie, and even Telegram. Let's start with something almost everyone has, though: email.

13. Click the **Email** media type and let's edit this one to suit our needs:

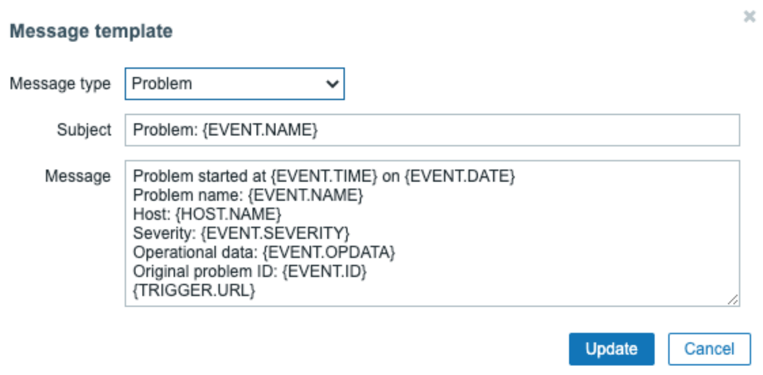

Media types

* Name	Email
Type	Email
* SMTP server	smtp.office365.com
SMTP server port	587
* SMTP helo	oicts.nl
* SMTP email	nathan@oicts.nl
Connection security	None [✓ STARTTLS] SSL/TLS
SSL verify peer	☐
SSL verify host	☐
Authentication	None [✓ Username and password]
Username	nathan@oicts.nl
Password	Change password
Message format	HTML [✓ Plain text]
Description	
Enabled	✓

Figure 3.17 – The Zabbix Media type creation page for email

14. I set it up to reflect my Office 365 settings, but any **Simple Mail Transfer Protocol** (**SMTP**) server should work. Fill in your SMTP settings, and we should be able to receive notifications.

15. Be sure to also check the next tab, **Message templates**. For example, the message template for a PROBLEM generation event looks like this:



Message template

Message type	Problem
Subject	Problem: {EVENT.NAME}

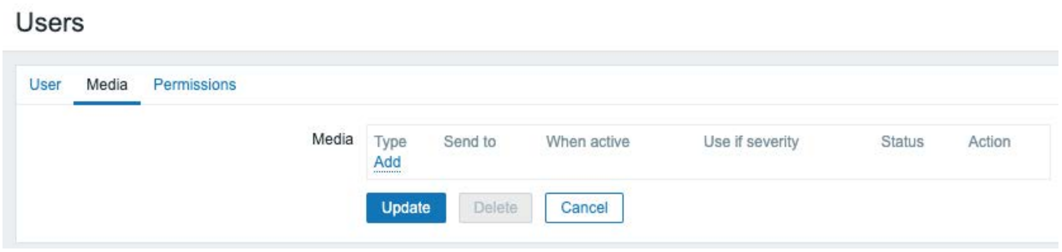
Message

```
Problem started at {EVENT.TIME} on {EVENT.DATE}
Problem name: {EVENT.NAME}
Host: {HOST.NAME}
Severity: {EVENT.SEVERITY}
Operational data: {EVENT.OPDATA}
Original problem ID: {EVENT.ID}
{TRIGGER.URL}
```

Figure 3.18 – The default Zabbix Message template page for problems

We set this up like this so that we get a message telling us just what's going on. This is fully customizable as well, to reflect just what we want to know.

16. Let's keep the default settings for now. Last, but not least, go to **Administration | Users** and edit a user in the **Zabbix administrators** or **Networking** group. I will use the **Admin** user as an example:



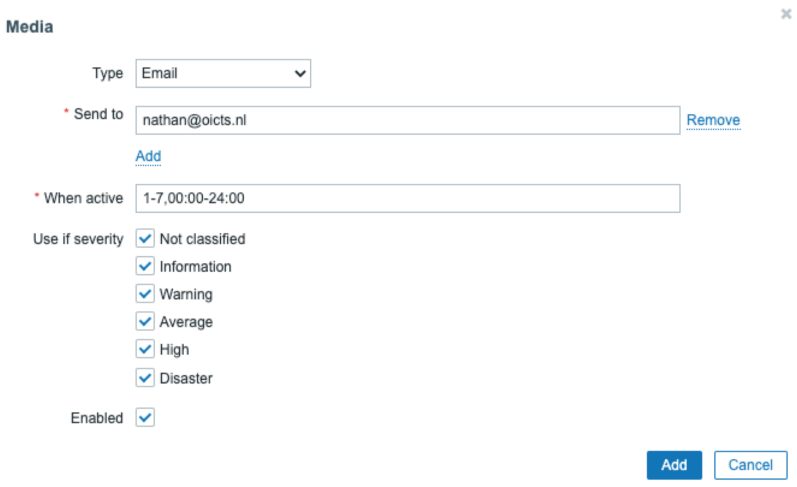
Users

Media

Type	Send to	When active	Use if severity	Status	Action
Add					

Figure 3.19 – The Zabbix User Media page for the Admin user

17. At this **Users** edit window, go to **Media** and click the **Add** button. We want to add the following to notify us of all trigger severities at our email address:



Media

Type	Email
* Send to	nathan@oicts.nl
* When active	1-7,00:00-24:00
Use if severity	✓ Not classified ✓ Information ✓ Warning ✓ Average ✓ High ✓ Disaster
Enabled	✓

Figure 3.20 – The Zabbix User Media creation page for the Admin user

18. Now, click on the blue **Add** button and finish creating this user media.

How it works...

Now, that's how we set up alerts in Zabbix. You will now receive alerts at your email, as shown in the following flowchart:

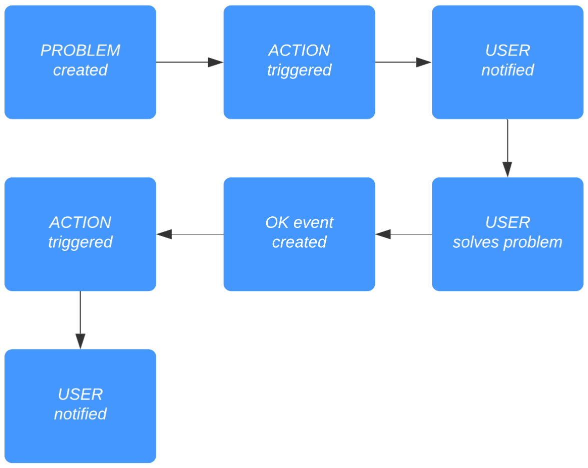


Figure 3.21 – Diagram showing Zabbix problem flow

When something breaks, a PROBLEM in Zabbix is triggered by our trigger configuration. Our ACTION will then be triggered by our PROBLEM event and it will use the **Media Type** and **User Media** configuration to notify our user. Our user then fixes the issue (for instance, rebooting a stuck server), and then an OK event will be generated. We will then trigger the ACTION again and get an OK event generation.

> **Tip**
> Before building alerts such as this, make a workflow (as shown previously) for yourself, specifying just which user groups and users should be notified. This way, you keep it clear for yourself just how you will use Zabbix for alerting.

There's more...

There are loads of media types and integrations, and we've just touched the tip of the iceberg by seeing a list of predefined ones. Make sure to check out the Zabbix integration list (`https://www.zabbix.com/integrations`) for more options, or build your own using the Zabbix webhooks and other extensions available.

Keeping alerts effective

It's important to keep our alerts effective to make sure we are neither overwhelmed nor underwhelmed by notifications. To do this, we will change our trigger and the **Email** media type to reflect just what we want to see.

Getting ready

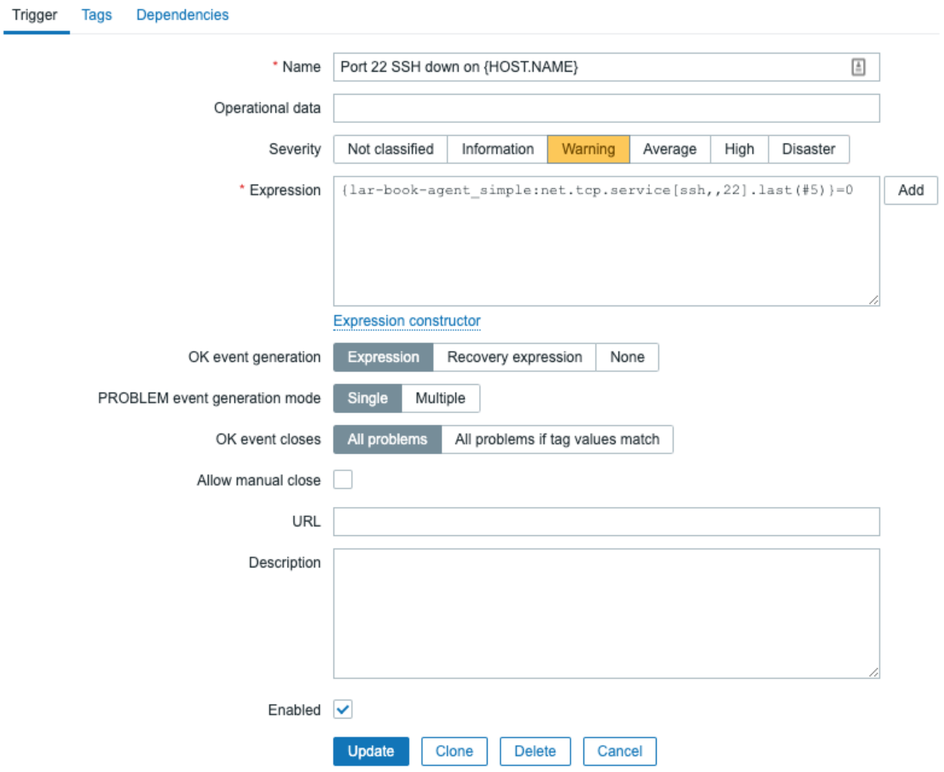
We will be using *Trigger 1* from the first recipe and the default email media type in Zabbix 5. Furthermore, of course, we'll also be using our Zabbix server.

How to do it...

To create effective alerts, let's follow these steps:

1. Let's get started on *Trigger 1*, which we created in this chapter's *Setting up triggers* recipe. Navigate to our `lar-agent-simple` host by going to **Configuration | Hosts** and clicking the Triggers for the host.

2. Here, we select our trigger named `Port 22 SSH down on {HOST.NAME}`:


Trigger | Tags | Dependencies

* Name	Port 22 SSH down on {HOST.NAME}
Operational data	
Severity	Not classified / Information / [✓ Warning] / Average / High / Disaster
* Expression	{lar-book-agent_simple:net.tcp.service[ssh,,22].last(#5)}=0
	Expression constructor
OK event generation	[✓ Expression] / Recovery expression / None
PROBLEM event generation mode	[✓ Single] / Multiple
OK event closes	[✓ All problems] / All problems if tag values match
Allow manual close	☐
URL	
Description	
Enabled	✓

Update | Clone | Delete | Cancel

Figure 3.22 – Trigger 1 from the previous recipe

This trigger is quite simple, so fortunately there isn't a lot to change here. We already used the hostname in the name, but we can change the name to reflect a message that is clearer.

3. Change the name, as follows:

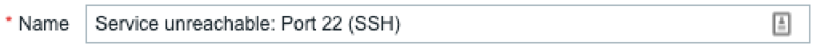
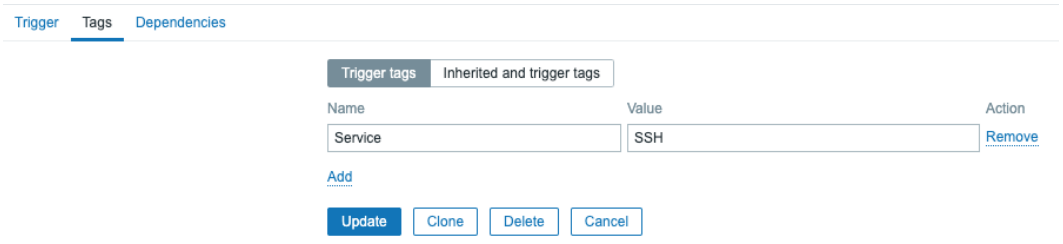

* Name	Service unreachable: Port 22 (SSH)

Figure 3.23 – New trigger 1 name

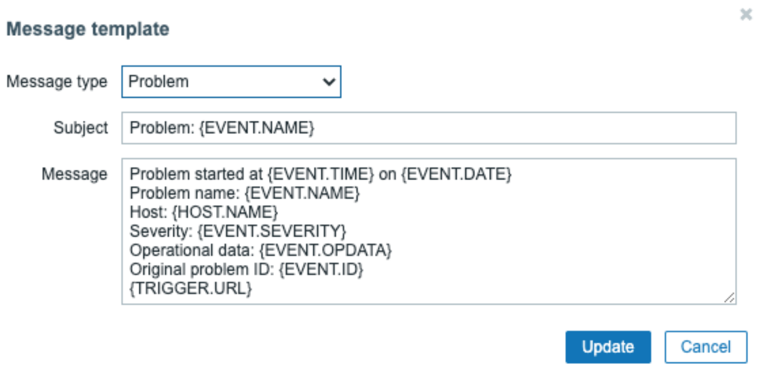
4. Next, navigate to the **Tags** page to add some tags for keeping the triggers organized. Let's add the following:



Name	Value	Action
Service	SSH	Remove

Figure 3.24 – Trigger 1 tags

5. Another great way of keeping everything organized is changing media type messages. Let's change a media type to reflect our own structured needs. Navigate to **Administrations** | **Media types** and select our media type named **Email**.
6. Select **Message templates** and click **Edit** next to our first **Problem**. This will bring us to the following window:



Message type: Problem

Subject: Problem: {EVENT.NAME}

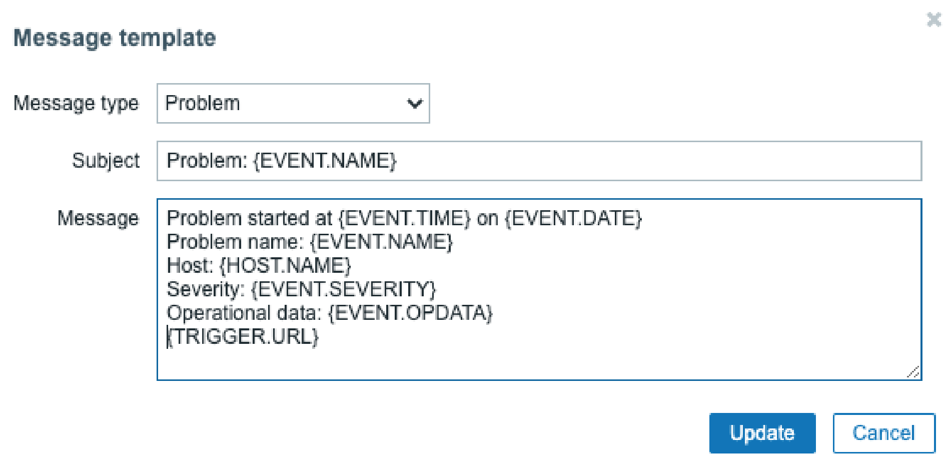
Message:

```
Problem started at {EVENT.TIME} on {EVENT.DATE}
Problem name: {EVENT.NAME}
Host: {HOST.NAME}
Severity: {EVENT.SEVERITY}
Operational data: {EVENT.OPDATA}
Original problem ID: {EVENT.ID}
{TRIGGER.URL}
```

Figure 3.25 – Standard email media type message

Now, Zabbix uses the default configured message under the media type when we do not use a custom message. But if we want to change that message, we can do that here by creating a custom message. Our default under the **Email** media type looks like the previous screenshot.

7. We can change the message on the media type. For instance, if we don't want to see the **Original problem ID** or when we want a more customized message, simply remove that line, as shown in the following screenshot:



Message type: Problem

Subject: Problem: {EVENT.NAME}

Message:
```
Problem started at {EVENT.TIME} on {EVENT.DATE}
Problem name: {EVENT.NAME}
Host: {HOST.NAME}
Severity: {EVENT.SEVERITY}
Operational data: {EVENT.OPDATA}
{TRIGGER.URL}
```

Figure 3.26 – Custom email media type message

How it works...

We've done two things in this recipe. We've changed our trigger name and we've added a tag to our trigger.

Keeping trigger names clear and defined in a structured way is important to keeping our Zabbix environment structured. Instead of just naming our trigger `Port 22 SSH down on {HOST.NAME}`, we've added standardization to our setup and can now do cool structuration such as this with our future triggers:

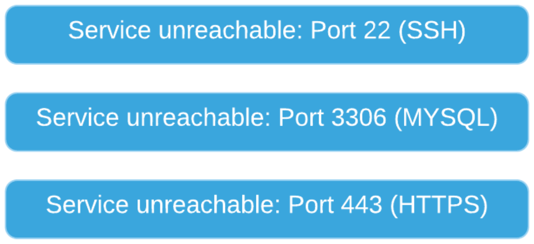
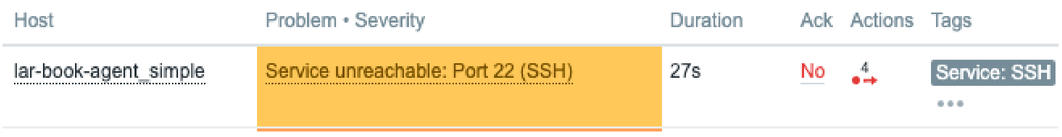

Service unreachable: Port 22 (SSH)
Service unreachable: Port 3306 (MYSQL)
Service unreachable: Port 443 (HTTPS)

Figure 3.27 – Trigger structure diagram

Our triggers are all clear and we can immediately see which host, port, and service are down.

On top of that, we've added a tag for the service that is down, which will now immediately display our service in a clear way, alerting us to exactly what is going on:



Host	Problem • Severity	Duration	Ack	Actions	Tags
lar-book-agent_simple	Service unreachable: Port 22 (SSH)	27s	No	4	Service: SSH •••

Figure 3.28 – Trigger down, structured

Another thing we've done is remove the macro {HOST.NAME} macro, as we can already see which host this trigger is on by checking the **Host** field.

We've also changed our action in this recipe. Changing a message on **Media types** is a powerful way to keep our problem channels structured. Sometimes, we want to see less or more information on certain channels, and changing media type messages is one way to do this.

We can also create custom messages on an **Action** level, changing all the messages sent to all channels.

There's more...

What I'm trying to teach you in this recipe is that although it might be simple to set up Zabbix, it is not simple to set up a good monitoring solution with Zabbix—or any monitoring tool, for that matter—if you don't plan. Carefully plan out how you want your triggers to be structured before you build everything in your Zabbix installation.

An engineer that works in a structured way and takes time to build a good monitoring solution will save a lot of hours in the future because they understand the problem before anyone else.

Customizing alerts

Alerting is very useful, especially in combination with some of the tricks we've learned in this book so far to keep everything structured. But sometimes, we need a little more from our alerts than what we are already getting from Zabbix out of the box.

In this recipe, we'll do a small bit of customization to make the alerts more our own.

Getting ready

For this chapter, all we are going to need is our current Zabbix server installation.

How to do it…

To customize alerts, follow these steps:

1. Let's create some custom severities in our Zabbix server to reflect our organization's needs. Navigate to **Administration | General** and select **Trigger severities** from the drop-down menu:

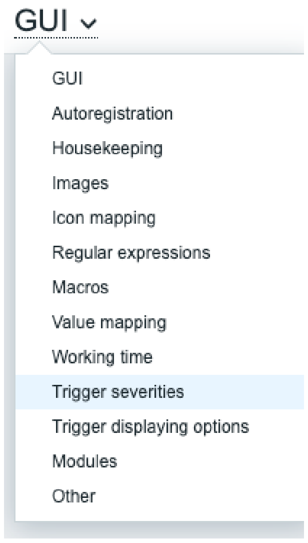
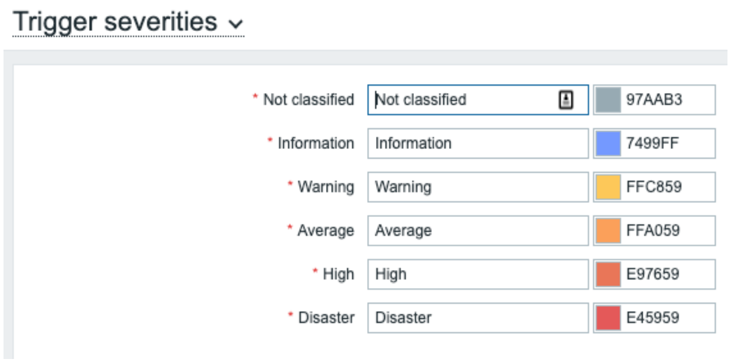


Figure 3.29 – Drop-down menu at Administration | General

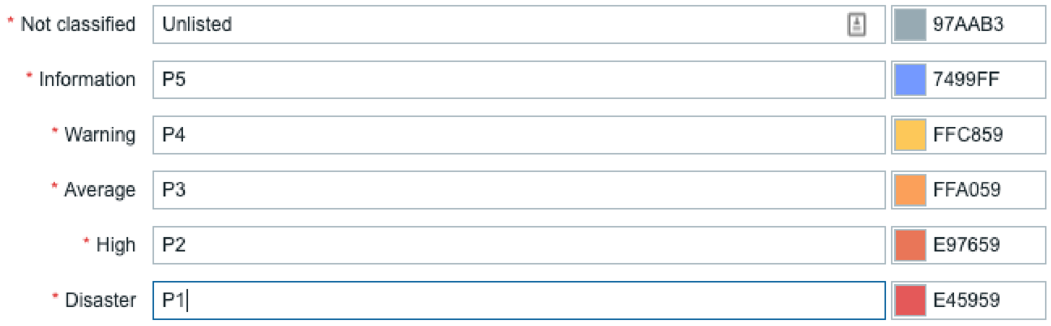
After selecting this, we'll be taken to our next page. This window contains the default Zabbix's **Trigger severities**, as shown in the following screenshot:


Trigger severities

* Not classified	Not classified	97AAB3
* Information	Information	7499FF
* Warning	Warning	FFC859
* Average	Average	FFA059
* High	High	E97659
* Disaster	Disaster	E45959

Figure 3.30 – Default Trigger severities window page

2. Next up, we will customize the default **Trigger severities**, as follows:



* Not classified	Unlisted	97AAB3
* Information	P5	7499FF
* Warning	P4	FFC859
* Average	P3	FFA059
* High	P2	E97659
* Disaster	P1	E45959

Figure 3.31 – Custom Trigger severities window page

How it works...

Not all companies like using terms such as **High** and **Disaster**, but prefer using different severities such as **P2** and **P1**. Using custom severities, we can customize Zabbix to make it more our own and thus reflect what we've already been using in different tools, for example.

Changing custom severities is not a necessity by any means, but it can be a good way to adopt Zabbix more easily if you are used to something different.

4
Building Your Own Structured Templates

It's time to start with one of the most important tasks in Zabbix: building structured templates. A good Zabbix setup relies heavily on templating, and there is a huge difference between a good and a bad template. So, if you're new to Zabbix or you haven't started building your own templates yet, pay close attention to this chapter.

In this chapter, we will go over how to set up your templates, and how to fill them with the right items and triggers. Also important is the use of macros and **Low-Level Discovery** (**LLD**). After following these recipes, you will be more than ready to build solid Zabbix templates with the right format, and even LLD.

In this chapter, we'll cover the following recipes:

- Creating your Zabbix template
- Setting up template applications
- Creating template items
- Creating template triggers

- Setting up different kinds of macros
- Using LLD on templates
- Nesting Zabbix templates

Technical requirements

We will need our Zabbix server from *Chapter 3*, *Working with Triggers and Alerts*, to monitor our **Simple Network Management Protocol** (**SNMP**) host. For the SNMP host, we can use the host we set up in the *Working with SNMP monitoring* recipe in *Chapter 2*, *Setting up Zabbix Monitoring*.

Creating your Zabbix template

In this recipe, we will start with the basics of creating a Zabbix template. We will go over the structure of Zabbix templating and why we need to pay attention to certain aspects of templating.

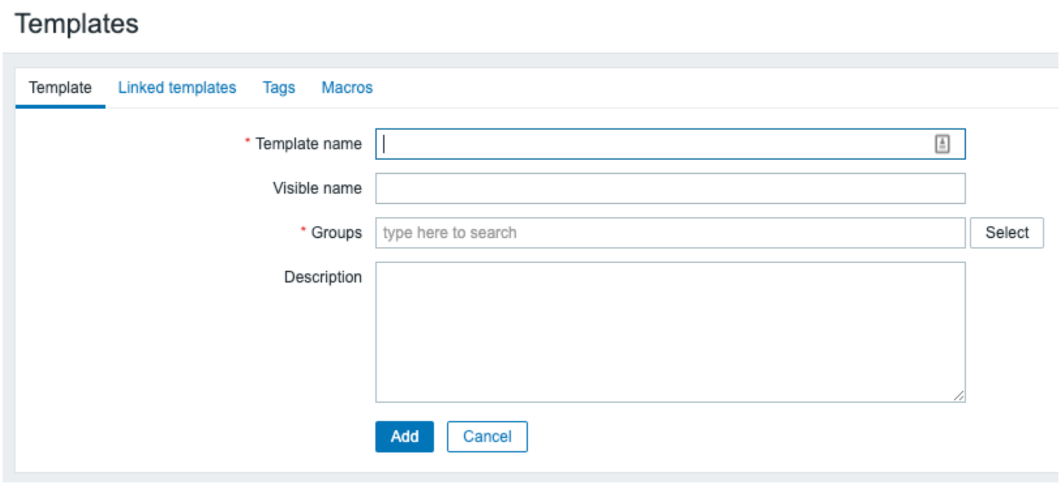
Getting ready

All you will need in this recipe is your Zabbix server.

How to do it...

Now, let's get started with building our structured Zabbix template.

1. Open your Zabbix frontend and navigate to **Configuration | Templates**.
2. At this page, click the **Create template** button in the top-right corner. This will lead you to the following page:



Templates

Template | Linked templates | Tags | Macros

* Template name	
Visible name	
* Groups	type here to search
Description	

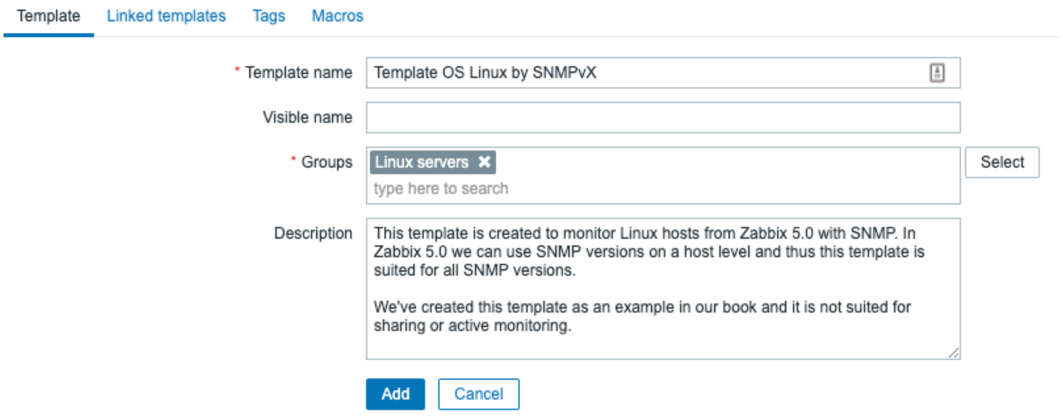
Figure 4.1 – The create template page, empty

At this point, we are going to need to name our template and assign a group to it. We will be creating an SNMP template to monitor a Linux host. I'll be using SNMP in the example to show how the templates are structured.

Important note

Use SNMP to monitor network equipment, custom equipment supporting SNMP, and more. SNMP is very versatile and easy to understand, and is implemented by a lot of hardware manufacturers. For Linux hosts, I'd still recommend the very powerful Zabbix agent, which we covered in the *Setting up Zabbix agent monitoring* recipe in *Chapter 2*, *Setting up Zabbix Monitoring*.

3. Create your template with the following information:



Template | Linked templates | Tags | Macros

* Template name	Template OS Linux by SNMPvX
Visible name	
* Groups	Linux servers
Description	This template is created to monitor Linux hosts from Zabbix 5.0 with SNMP. In Zabbix 5.0 we can use SNMP versions on a host level and thus this template is suited for all SNMP versions. We've created this template as an example in our book and it is not suited for sharing or active monitoring.

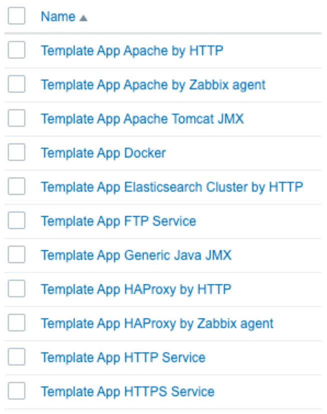
Figure 4.2 – The create template page filled with information for the SNMP template

We will not use any **Linked templates**, **Tags**, and **Macros** yet, but we'll address some of these functionalities later. That's all there is to creating our template, but there's nothing in it besides a name, group, and description so far.

How it works...

Now, there's not a lot of work involved in creating our first template—it's quite straightforward. What we need to keep in mind is the right naming convention here.

Now, you might think to yourself: *Why is naming a template so important?* Well, we are going to create a lot of templates when working with Zabbix. For example, this is a small part of the list of out-of-the-box templates:



Name ▲
Template App Apache by HTTP
Template App Apache by Zabbix agent
Template App Apache Tomcat JMX
Template App Docker
Template App Elasticsearch Cluster by HTTP
Template App FTP Service
Template App Generic Java JMX
Template App HAProxy by HTTP
Template App HAProxy by Zabbix agent
Template App HTTP Service
Template App HTTPS Service

Figure 4.3 – Some out-of-the-box templates

As you can see, this is already a large list, and all of these templates are following a singular straightforward naming convention. The template we built ourselves follows that same convention, which is the following:

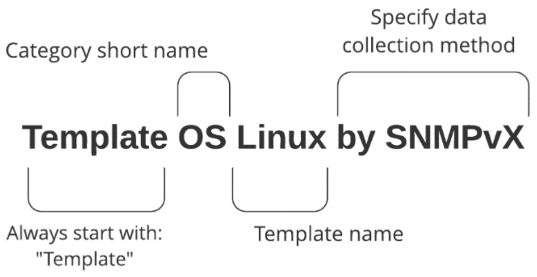

Category short name
Specify data collection method
Template OS Linux by SNMPvX
Always start with: "Template"
Template name

Figure 4.4 – Template naming convention explanation

Looking at the list and comparing it to the naming convention we went over in *Figure 4.4*, we can see the following pattern:

- **Prefix**: (Template) We provide all the templates with the Template prefix.
- **Category short name**: (OS) We provide a quick description of the category this template will pertain. In this case we'll name it OS, as we'll be monitoring a host OS.
- **Template name**: (Linux) We name the template—in this case, we'll call it Linux because the OS we monitor will be Linux.
- **Data collection method**: (by SNMPvX) We will add our data collection method at the end of the template as we might monitor the Linux OS in other ways besides SNMP.

Adhering to the guidelines in this naming convention and thus using the correct template names is our first step in creating the correct structure for our template. This makes it easy to find out which templates we want to use on which hosts.

There's more...

When building templates, adhere to the Zabbix guidelines. That's what we will do in this book as well, combined with our experience in creating templates. If you want to learn more about Zabbix templating guidelines, check the following URL: `https://www.zabbix.com/documentation/guidelines/thosts`.

Setting up template applications

Our next step in setting up our Zabbix template is setting up template applications. Applications in Zabbix templates are used to keep our items structured under a nice identifier.

Getting ready

To get started with this recipe, you will need a Zabbix server and a template on that server, preferably our template created in the previous recipe.

How to do it…

Creating template applications is a way to easily sort items. To get started, the first thing you will need to do is navigate to the template and follow these steps:

1. Go to **Configuration** | **Templates** and click on our template, called **Template OS Linux by SNMPvX**.
2. Here, you will click the **Applications** option next to the template name and you'll be taken to this screen:

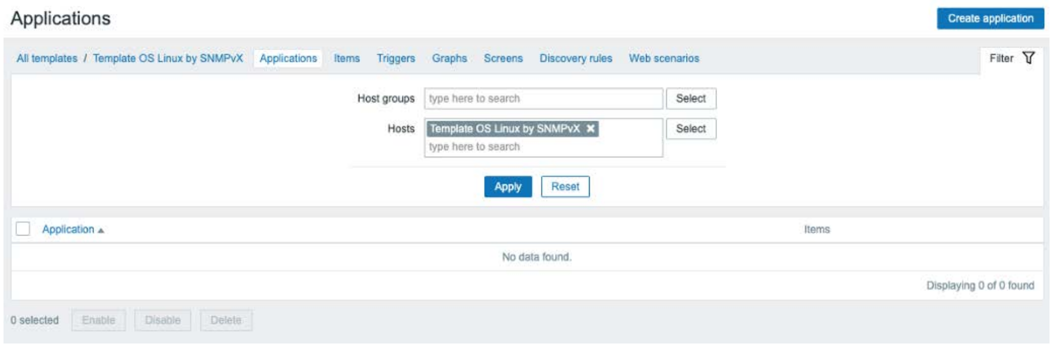
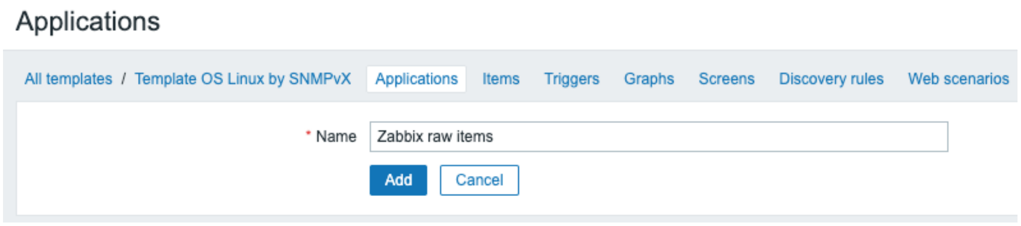


Figure 4.5 – Zabbix Applications screen for the SNMP template

Now, the first thing I always want to do is create an application for **Zabbix raw items** and for **General**.

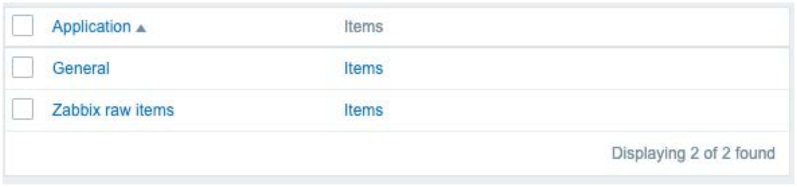
3. Start creating applications by clicking the **Create application** button in the top-right corner. This will take you to the following screen:



* Name: Zabbix raw items

Figure 4.6 – Zabbix application creation screen for the SNMP template

4. Fill in the **Name**, as I did in the preceding screenshot, and click the **Add** button.
5. Repeat the steps to create an application named **General** as well. You will end up with the following:



Application ▲	Items
General	Items
Zabbix raw items	Items

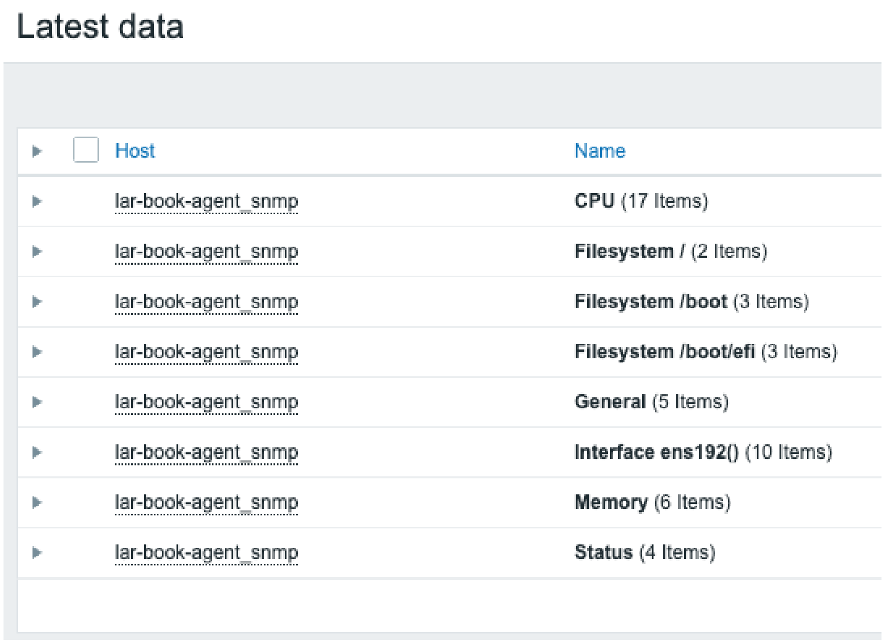
Displaying 2 of 2 found

Figure 4.7 – Zabbix filled application screen for the SNMP template

How it works...

Now, there's a lot more to creating applications than it might seem at first through following this recipe. Applications play a key part in keeping your Zabbix environment structured. You will see the template **Application** in the **Latest data** window, and with a lot of items on one host, they will create readability by making items easy to filter.

For example, when checking the latest data for an existing host, we can see the following:



Latest data

Host	Name
lar-book-agent_snmp	**CPU** (17 Items)
lar-book-agent_snmp	**Filesystem /** (2 Items)
lar-book-agent_snmp	**Filesystem /boot** (3 Items)
lar-book-agent_snmp	**Filesystem /boot/efi** (3 Items)
lar-book-agent_snmp	**General** (5 Items)
lar-book-agent_snmp	**Interface ens192()** (10 Items)
lar-book-agent_snmp	**Memory** (6 Items)
lar-book-agent_snmp	**Status** (4 Items)

Figure 4.8 – Example Latest data window for host lar-book-agent_snmp

All of the items have been divided up here into neat template groups, providing you with an easy list of where to look for your data.

We've pre-created two applications for now, but we will definitely not stop here. When creating new items, we will also create more applications, providing us with a list such as the one in the preceding screenshot.

For now, we have two applications, though:

- **General**: A general application for data such as general system information or data that has no other application to be in.
- **Zabbix raw items**: An application for raw item data. We pre-create this for use with items that will collect data but that we do not need to see, such as dependent and calculated item data.

See also

In this chapter, the recipe titled *Using LLD on templates* will also explain **application prototypes**, whereby we will create applications automatically based on the discovery settings. Application prototypes are the recommended way of working with applications when creating discovery and are amazing for keeping templates structured. More about that later.

Creating template items

Let's get started with finally creating some real template items, because in the end, items are what it is all about in Zabbix. Without items we don't have data, and without data we do not have anything to work with in our monitoring system.

Getting ready

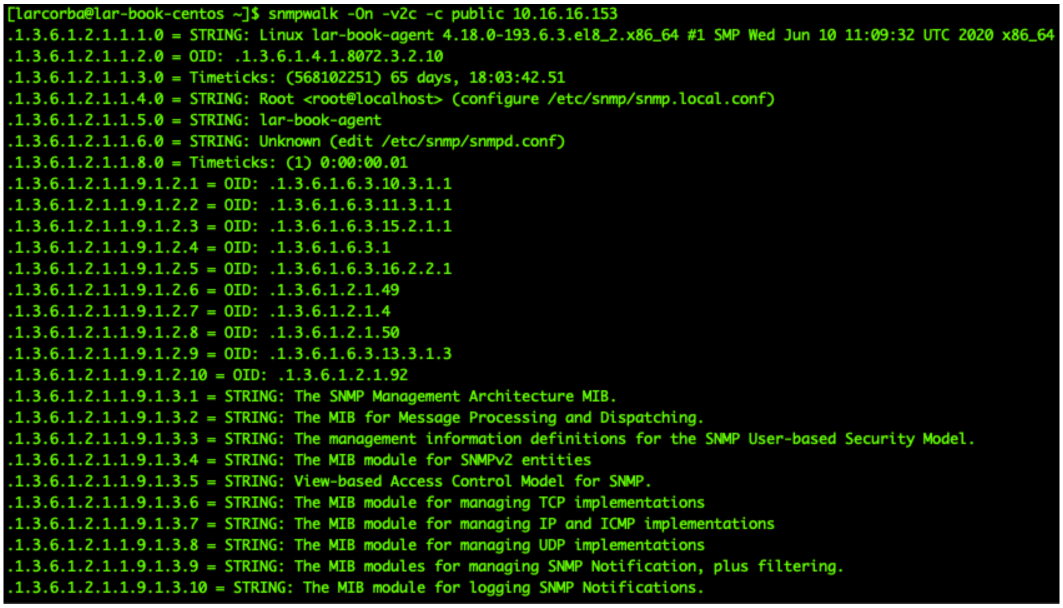
Now, moving along, we are going to need our Zabbix server and a host that we can monitor with SNMP. In *Chapter 2*, *Setting up Zabbix Monitoring*, we monitored a host with SNMP, so we can use this host again. We'll also use the Zabbix template from the previous recipes.

How to do it…

1. First of all, let's log in to our Zabbix server **command-line interface** (**CLI**) and enter `snmpwalk`, with the following command:

```
snmpwalk -On -v2c -c public 10.16.16.153
```

We will receive an answer such as this:



```
[larcorba@lar-book-centos ~]$ snmpwalk -On -v2c -c public 10.16.16.153
.1.3.6.1.2.1.1.1.0 = STRING: Linux lar-book-agent 4.18.0-193.6.3.el8_2.x86_64 #1 SMP Wed Jun 10 11:09:32 UTC 2020 x86_64
.1.3.6.1.2.1.1.2.0 = OID: .1.3.6.1.4.1.8072.3.2.10
.1.3.6.1.2.1.1.3.0 = Timeticks: (568102251) 65 days, 18:03:42.51
.1.3.6.1.2.1.1.4.0 = STRING: Root <root@localhost> (configure /etc/snmp/snmp.local.conf)
.1.3.6.1.2.1.1.5.0 = STRING: lar-book-agent
.1.3.6.1.2.1.1.6.0 = STRING: Unknown (edit /etc/snmp/snmpd.conf)
.1.3.6.1.2.1.1.8.0 = Timeticks: (1) 0:00:00.01
.1.3.6.1.2.1.1.9.1.2.1 = OID: .1.3.6.1.6.3.10.3.1.1
.1.3.6.1.2.1.1.9.1.2.2 = OID: .1.3.6.1.6.3.11.3.1.1
.1.3.6.1.2.1.1.9.1.2.3 = OID: .1.3.6.1.6.3.15.2.1.1
.1.3.6.1.2.1.1.9.1.2.4 = OID: .1.3.6.1.6.3.1
.1.3.6.1.2.1.1.9.1.2.5 = OID: .1.3.6.1.6.3.16.2.2.1
.1.3.6.1.2.1.1.9.1.2.6 = OID: .1.3.6.1.2.1.49
.1.3.6.1.2.1.1.9.1.2.7 = OID: .1.3.6.1.2.1.4
.1.3.6.1.2.1.1.9.1.2.8 = OID: .1.3.6.1.2.1.50
.1.3.6.1.2.1.1.9.1.2.9 = OID: .1.3.6.1.6.3.13.3.1.3
.1.3.6.1.2.1.1.9.1.2.10 = OID: .1.3.6.1.2.1.92
.1.3.6.1.2.1.1.9.1.3.1 = STRING: The SNMP Management Architecture MIB.
.1.3.6.1.2.1.1.9.1.3.2 = STRING: The MIB for Message Processing and Dispatching.
.1.3.6.1.2.1.1.9.1.3.3 = STRING: The management information definitions for the SNMP User-based Security Model.
.1.3.6.1.2.1.1.9.1.3.4 = STRING: The MIB module for SNMPv2 entities
.1.3.6.1.2.1.1.9.1.3.5 = STRING: View-based Access Control Model for SNMP.
.1.3.6.1.2.1.1.9.1.3.6 = STRING: The MIB module for managing TCP implementations
.1.3.6.1.2.1.1.9.1.3.7 = STRING: The MIB module for managing IP and ICMP implementations
.1.3.6.1.2.1.1.9.1.3.8 = STRING: The MIB module for managing UDP implementations
.1.3.6.1.2.1.1.9.1.3.9 = STRING: The MIB modules for managing SNMP Notification, plus filtering.
.1.3.6.1.2.1.1.9.1.3.10 = STRING: The MIB module for logging SNMP Notifications.
```

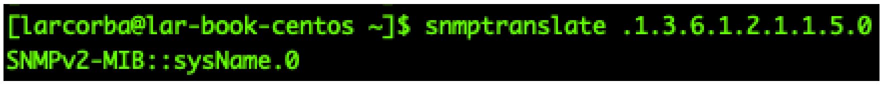
Figure 4.9 – snmpwalk reply

Now, let's capture our hostname in our template first, as it is an important item to have. When working with SNMP, I always like to work with untranslated SNMP **Object Identifiers** (**OIDs**). For our hostname, this is `.1.3.6.1.2.1.1.5.0`.

2. Translate this OID to make sure it is actually the system name. Enter the following command at the Zabbix server CLI:

```
snmptranslate .1.3.6.1.2.1.1.5.0
```

This will return the following reply:



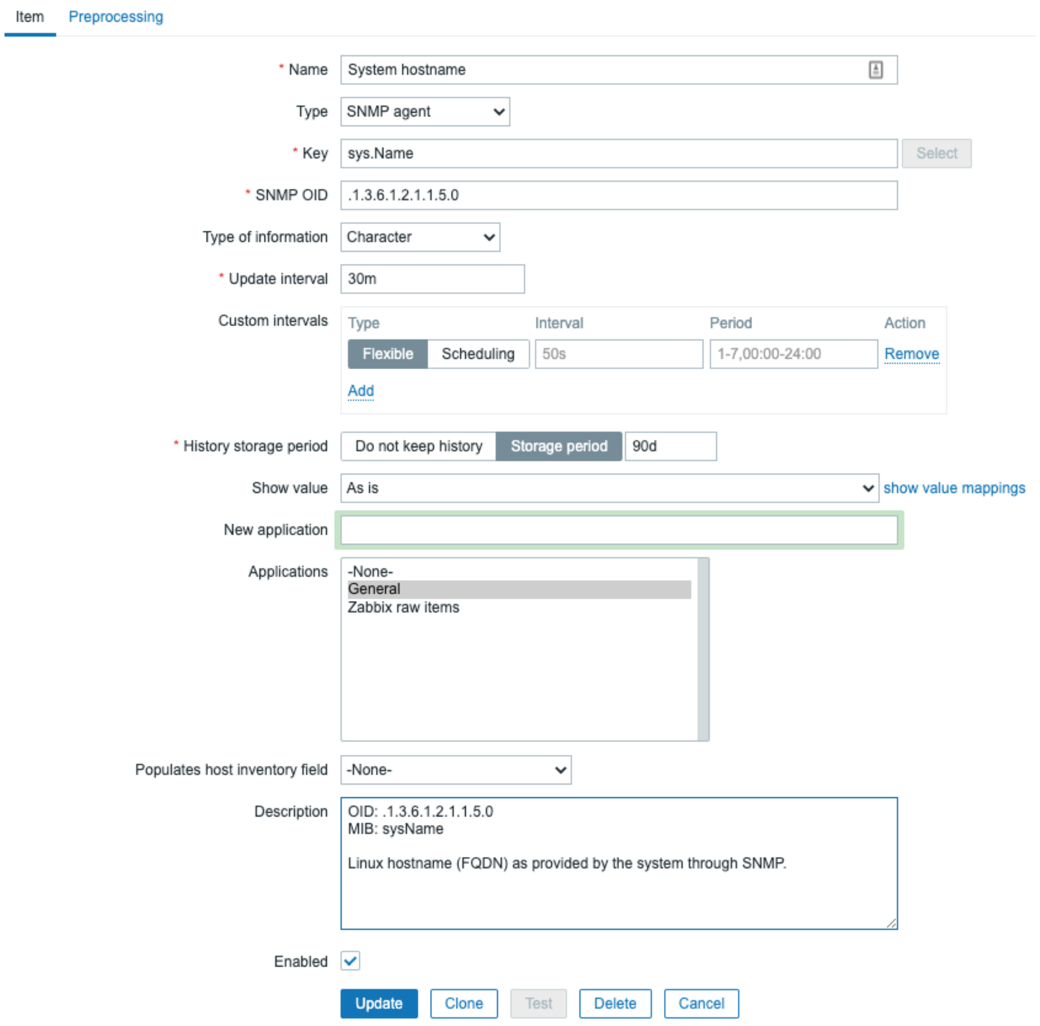
```
[larcorba@lar-book-centos ~]$ snmptranslate .1.3.6.1.2.1.1.5.0
SNMPv2-MIB::sysName.0
```

Figure 4.10 – snmptranslate reply

Tip

Using `-On` in your SNMP command makes sure that we are receiving the OIDs instead of the **Management Information Base** (**MIB**) translation. If we want to work the other way around, we can omit the `-On` in our command and `snmptranslate` the translated OID.

3. Now that we know how to get our hostname, add this to our template. Navigate to **Configuration | Templates** and select our **Template OS Linux by SNMPvX** template.
4. Here, we will go to **Items**. In the upper-right corner, select **Create item** to create the following item:

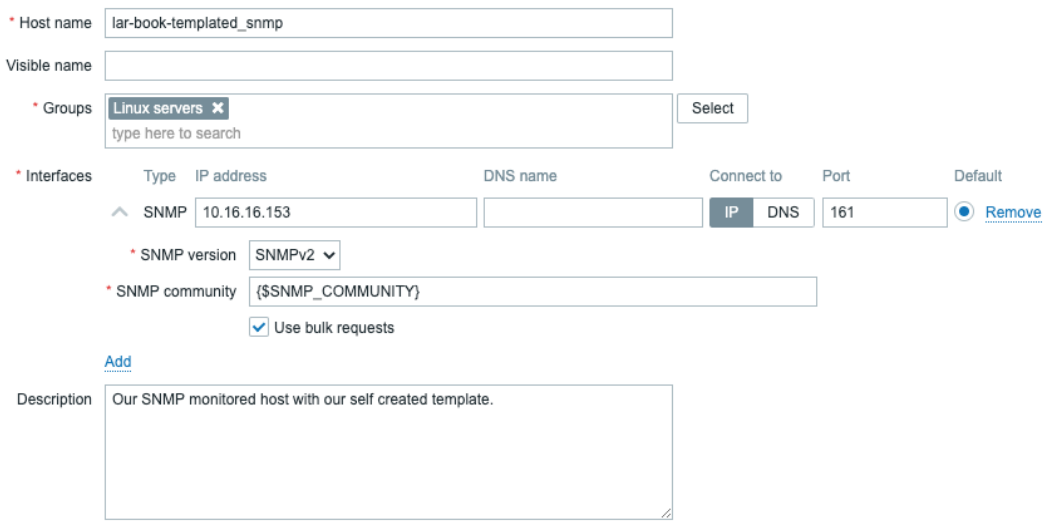


* Name	System hostname
Type	SNMP agent
* Key	sys.Name
* SNMP OID	.1.3.6.1.2.1.1.5.0
Type of information	Character
* Update interval	30m
Custom intervals	Type: Flexible; Interval: 50s; Period: 1-7,00:00-24:00
* History storage period	Storage period 90d
Show value	As is
New application	
Applications	-None-, General, Zabbix raw items
Populates host inventory field	-None-
Description	OID: .1.3.6.1.2.1.1.5.0 MIB: sysName Linux hostname (FQDN) as provided by the system through SNMP.
Enabled	✓

Figure 4.11 – Item for sysName SNMP OID

Now that we have our first item, let's create a host as well and assign it to this template.

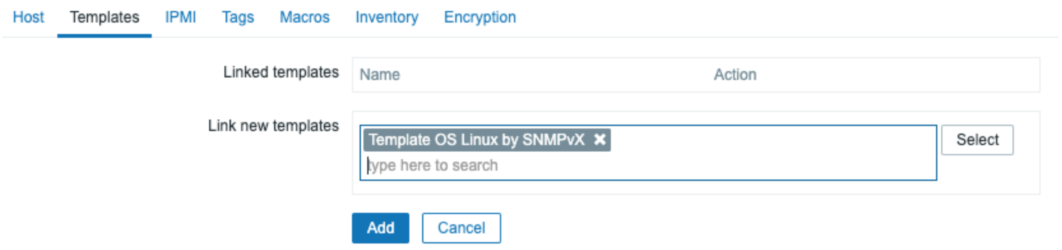
5. Navigate to **Configuration** | **Hosts** and click **Create host** in the top-right corner. Create a host with the following settings:



* Host name	lar-book-templated_snmp
Visible name	
* Groups	Linux servers
* Interfaces	SNMP, IP address 10.16.16.153, Connect to IP, Port 161, Default
* SNMP version	SNMPv2
* SNMP community	{$SNMP_COMMUNITY}
	✓ Use bulk requests
Description	Our SNMP monitored host with our self created template.

Figure 4.12 – New host with our self-created template

6. Don't forget to add the template to our new host before pressing the **Add** button. Click on **Templates** and fill in the following information:



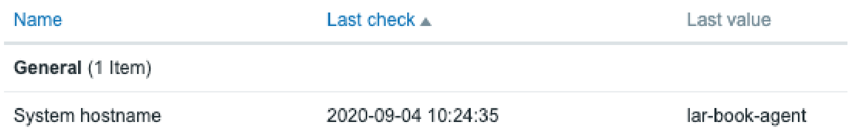
Link new templates: Template OS Linux by SNMPvX

Figure 4.13 – Add template screen on a host

7. Now, you can press the **Add** button, and our new host will be monitored.

How it works...

When we create items such as this on our template, every single host that we assign this template to will start using this item. For instance, our newly created host will show the following latest data:



Name	Last check ▲	Last value
General (1 Item)		
System hostname	2020-09-04 10:24:35	lar-book-agent

Figure 4.14 – Monitoring | Latest data for our new host

This will then be different for your all your monitored hosts, depending on the configured value on that host.

> **Important note**
> When creating an SNMP item, keep the following in mind. The Item SNMP OID always contains the non-translated OID. This is to make sure that we do not actually need MIB files for our templates to work.

Furthermore, the item key will be based on the translated OID. In our case, the translated OID was `sysName`, which we then turned into the `sys.Name` item key. These are general rules that we should all abide by when creating our templates, to make sure they are structured the same for everyone.

See also

To learn more about Zabbix and SNMP OIDs/MIBs, check out this awesome blog post by Zabbix's head of customer support, Dmitry Lambert:

`https://blog.zabbix.com/zabbix-snmp-what-you-need-to-know-and-how-to-configure-it/10345/#snmp-oid`

Creating template triggers

Now, creating templated triggers works in about the same way as creating templated items or normal triggers. Let's go over the process, to see how we do it and how to keep it structured.

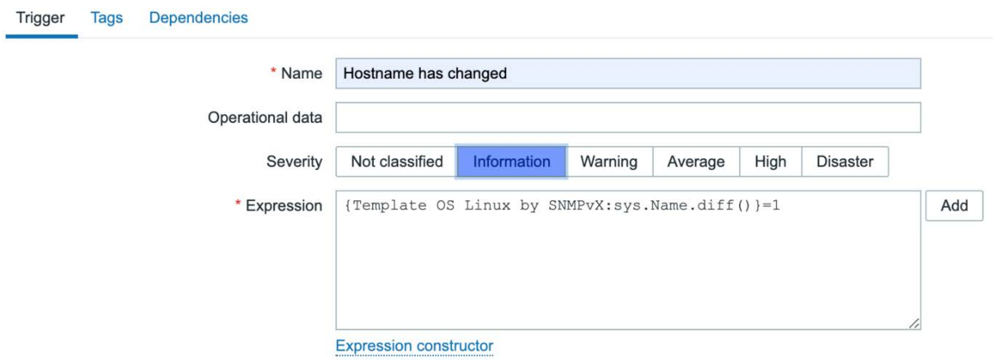
Getting ready

We will need the Zabbix server and the host from the previous recipe for this recipe.

How to do it…

We only have one item on our template yet, so let's trigger on this item:

1. Navigate to **Configuration** | **Templates** in our Zabbix frontend and select our **Template OS Linux by SNMPvX** template.
2. Now, click **Triggers** and then **Create Trigger** in the top-right corner. This will take us to the next page, where we will enter the following information:



* Name	Hostname has changed
Operational data	
Severity	Not classified / [✓ Information] / Warning / Average / High / Disaster
* Expression	{Template OS Linux by SNMPvX:sys.Name.diff()}=1

Figure 4.15 – Create trigger window for the SNMP template

3. Last, but not least, let's edit the hostname on our host to see if the trigger is working correctly. Change the hostname entry by executing the following command on the Linux host CLI:

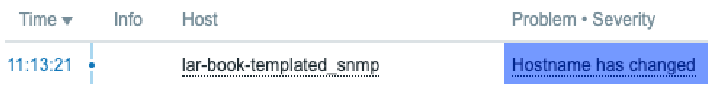
```
hostnamectl set-hostname lar-book-agent-t
```

4. Then, reboot using the following command:

```
shutdown -r now
```

How it works…

Now, this trigger will immediately be added to our host named `lar-book-templated_snmp` because we edited the template and the host is already configured with this template. When we changed the hostname, the trigger can immediately be triggered after the item is polled again:



Time ▾	Info	Host	Problem • Severity
11:13:21		lar-book-templated_snmp	Hostname has changed

Figure 4.16 – Hostname has changed trigger for host lar-book-templated_snmp

Because we used `.diff` in our trigger, the second time we poll this item the problem will automatically go away again. In our case, this will happen after 30 minutes.

> **Important note**
>
> Like a lot of other Zabbix users, I always liked to use the `{HOST.HOST}` macro in trigger names, but according to Zabbix guidelines this isn't recommended. If you prefer this you can still use it, but it's a lot more useful to use the `Host` field. This will keep trigger names short and won't show us redundant information.

Setting up different kinds of macros

Now, when we are working with templates, a very efficient way to make your templates more useful is through the use of macros. In this recipe, we'll discover how to use macros to do this.

Getting ready

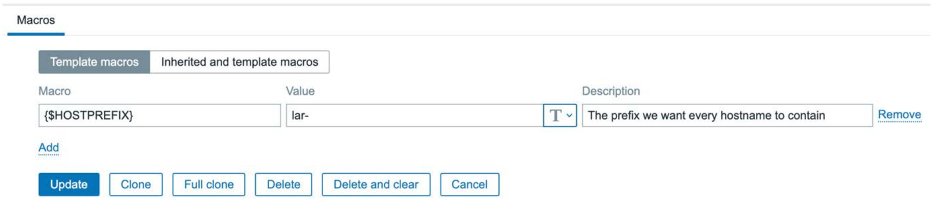
We are going to need our Zabbix server and our SNMP-monitored host from the previous recipes. We'll also need our Zabbix template, as created in the previous recipe.

How to do it…

Now, let's start with creating some macros on a template level. We'll be making two different types of macros.

Defining a user macro

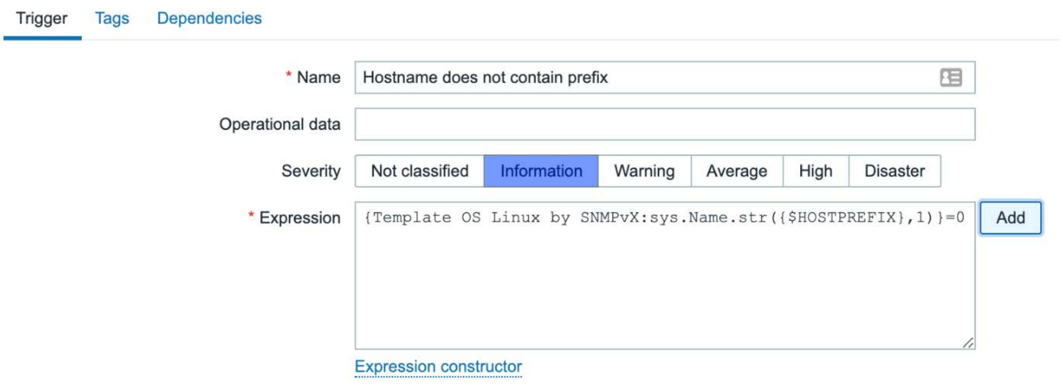
1. Now, let's start this recipe off by defining a user macro on our template. Navigate to **Configuration** | **Templates** and click our **Template OS Linux by SNMPvX** template.
2. Here, we will go to **Macros** and fill in the following fields:



Macro	Value	Description
{$HOSTPREFIX}	lar-	The prefix we want every hostname to contain

Figure 4.17 – Template-level macros

3. Click on **Update**, and let's move to **Trigger** to define a new trigger:



* Name	Hostname does not contain prefix
Operational data	
Severity	Not classified [✓ Information] Warning Average High Disaster
* Expression	{Template OS Linux by SNMPvX:sys.Name.str({$HOSTPREFIX},1)}=0

Figure 4.18 – Trigger creation window for the SNMP template

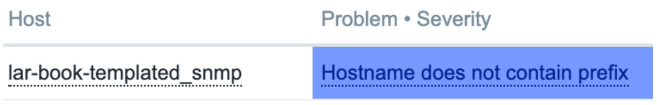
4. Now, change the hostname entry by executing the following command on the host CLI:

```
hostnamectl set-hostname dev-book-agent
```

5. Then, reboot using the following command:

```
shutdown -r now
```

6. Our trigger should fire, as shown in the following screenshot:

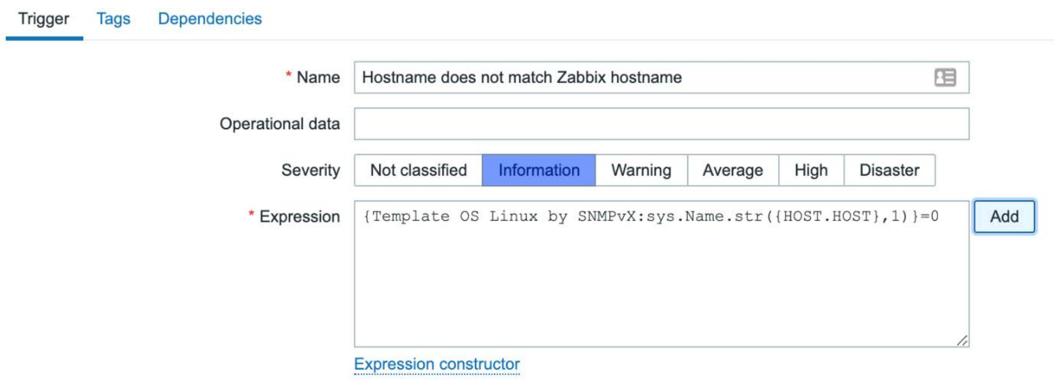


Host	Problem • Severity
lar-book-templated_snmp	Hostname does not contain prefix

Figure 4.19 – Trigger created problem for hostname prefix on host lar-book-templated_snmp

Defining a built-in macro

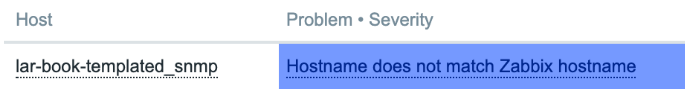
1. Now, let's work on defining a built-in macro on our template. Navigate to **Configuration | Templates** and click on our **Template OS Linux by SNMPvX** template.
2. Now, click **Triggers**, and in the top-right corner, click on **Create trigger**. Create a trigger with the following settings:



* Name	Hostname does not match Zabbix hostname
Operational data	
Severity	Not classified [✓ Information] Warning Average High Disaster
* Expression	{Template OS Linux by SNMPvX:sys.Name.str({HOST.HOST},1)}=0

Figure 4.20 – Trigger creation window for hostname match

3. This will then trigger a problem, as expected:



Host	Problem • Severity
lar-book-templated_snmp	Hostname does not match Zabbix hostname

Figure 4.21 – Trigger created problem Hostname does not match

How it works...

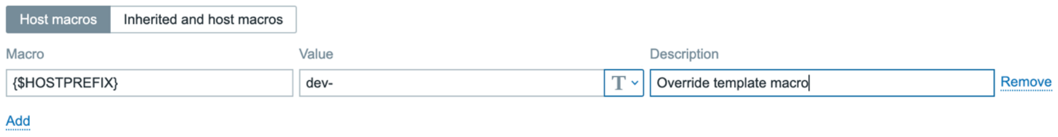
There are three types of macros: Built-in, User, and LLD. All of these macros can be used on templates, but also directly on hosts. Macros are powerful for creating unique values in places that would otherwise contain static information.

Let's discover how they work in this recipe.

How a user macro works

Now, because we want all of our hosts on this template to contain `lar` as a prefix, we create a user macro on a template level. This way, the user macro that will be used on every host with this template will be the same.

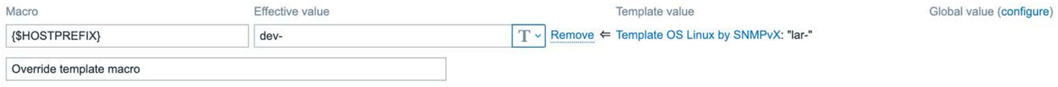
We then define our user macro in our trigger to use the value, which is `lar-` in this case. We can reuse this user macro in other triggers, items, and more. The great thing is that defining a user macro on a template level isn't the end of our possibilities. We can override template-level user macros by defining a host-level user macro. So, if we want a single host to contain a different prefix, we simply use that to override the template-level macro, like this:



Macro	Value	Description
{$HOSTPREFIX}	dev-	Override template macro

Figure 4.22 – Host-level macros page

If we then look at the inherited and host-level macros screen on our host, we will see the following:



Macro	Effective value	Template value
{$HOSTPREFIX}	dev-	⇐ Template OS Linux by SNMPvX: "lar-"
Override template macro		

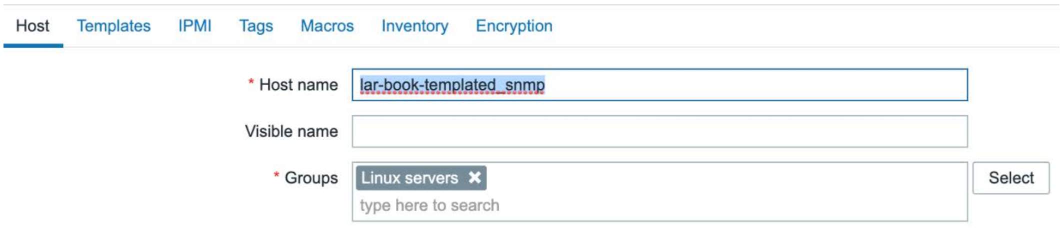
Figure 4.23 – Inherited and host-level macros page

We see the effective value now would be `dev-`, not `lar-`, which is exactly what we would be expecting to happen here.

How a built-in macro works

Now, a built-in macro comes from a predefined list of macros, defined by Zabbix. They are used to get Zabbix elements and put them in items, triggers, and more. This means that our built-in macro used in this case already contains a value.

In this case, we used `{HOST.HOST}`, which is the hostname we defined on our Zabbix host, like this:



* Host name	lar-book-templated_snmp
Visible name	
* Groups	Linux servers

Figure 4.24 – Zabbix host configuration page for host lar-book-templated_snmp

For every single host, this built-in macro would be different as our **Host name** value will be unique. This means that our trigger, although defined on a template level, will always be unique as well. This method is a very powerful way to use built-in macros in triggers, as we'll pull information from Zabbix directly into Zabbix again.

There's more...

A complete list of supported (built-in) macros can be found here:

`https://www.zabbix.com/documentation/current/manual/appendix/macros/supported_by_location`

This list will be updated by Zabbix, just as with every good Zabbix documentation page. This way, you can always use this page as a reference of up-to-date (built-in) macros for building your Zabbix elements.

Using LLD on templates

Now, let's get started on my favorite part of template creation: LLD. I think this is one of the most powerful and most used parts of Zabbix.

Getting ready

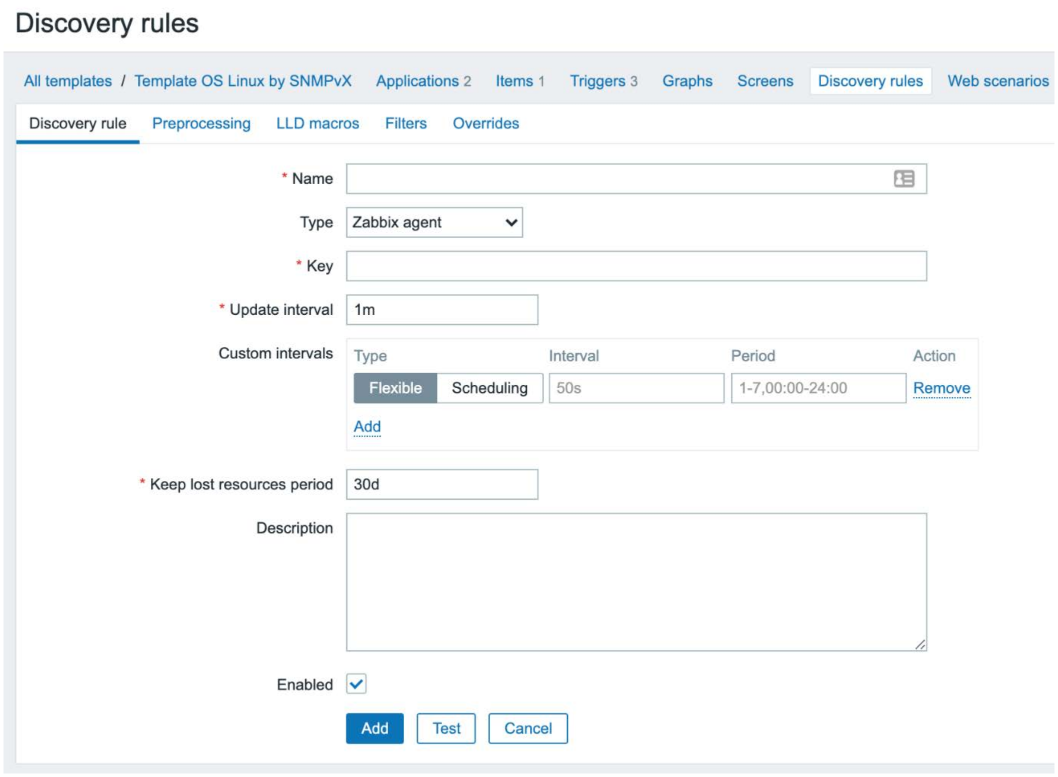
To get ready for this recipe, you will need your Zabbix server, the SNMP-monitored host from the previous recipes, and our template from the previous recipe.

A working knowledge of the SNMP tree structure is also recommended. So, make sure to read the *Working with SNMP monitoring* recipe in *Chapter 2*, *Setting up Zabbix Monitoring*, thoroughly.

How to do it...

1. Let's get started by navigating to **Configuration** | **Templates** and selecting our **Template OS Linux by SNMPvX** template.

2. Now, go to **Discovery rules**, and in the top-right corner, click **Create discovery rule**. This will lead you to the LLD creation page:


Discovery rules

All templates / Template OS Linux by SNMPvX   Applications 2   Items 1   Triggers 3   Graphs   Screens   Discovery rules   Web scenarios

Discovery rule   Preprocessing   LLD macros   Filters   Overrides

* Name	
Type	Zabbix agent
* Key	
* Update interval	1m
Custom intervals	Type: Flexible / Scheduling; Interval: 50s; Period: 1-7,00:00-24:00; Action: Remove; Add
* Keep lost resources period	30d
Description	
Enabled	✓

Add   Test   Cancel

Figure 4.25 – Zabbix LLD creation page, empty

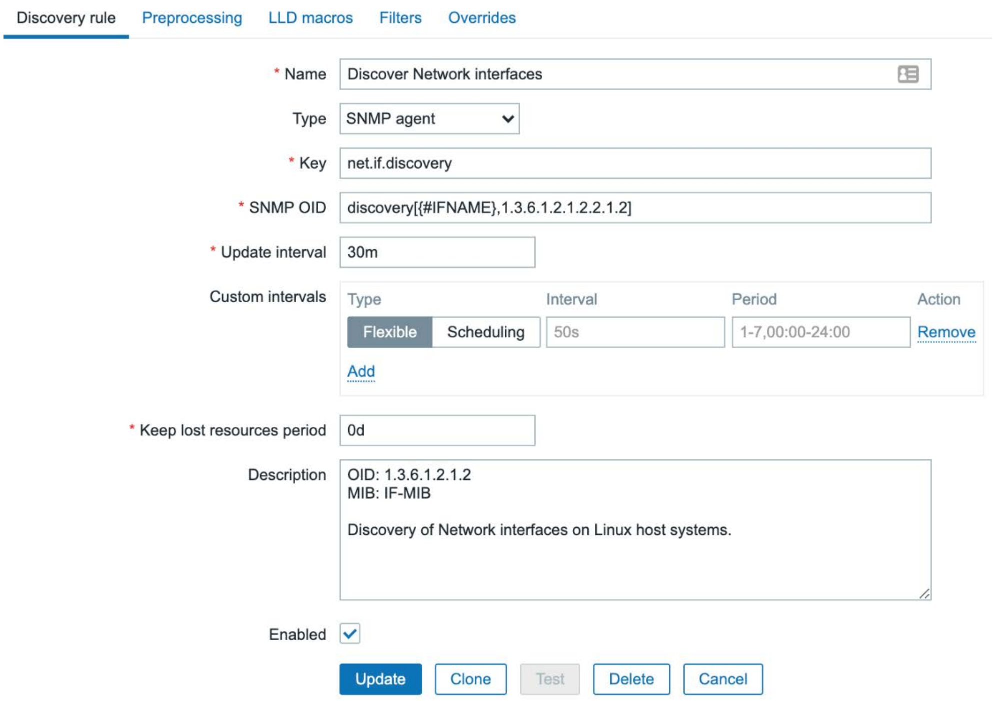
Now, we will be making a discovery rule to discover our interfaces on the Linux host. The Linux SNMP tree for interfaces is at OID `1.3.6.1.2.1.2`.

Important note

Make sure that Linux `net-snmp` is configured correctly in the `/etc/snmp/snmpd.conf` file. It's important to change the `view` in this file to show everything from `.1` and up, like this:

```
view systemview included .1
```

3. Now, let's continue with creating our LLD rule by adding the following to our LLD creation page:



* Name	Discover Network interfaces
Type	SNMP agent
* Key	net.if.discovery
* SNMP OID	discovery[{#IFNAME},1.3.6.1.2.1.2.2.1.2]
* Update interval	30m
Custom intervals	Type: Flexible; Interval: 50s; Period: 1-7,00:00-24:00
* Keep lost resources period	0d
Description	OID: 1.3.6.1.2.1.2 MIB: IF-MIB Discovery of Network interfaces on Linux host systems.
Enabled	✓

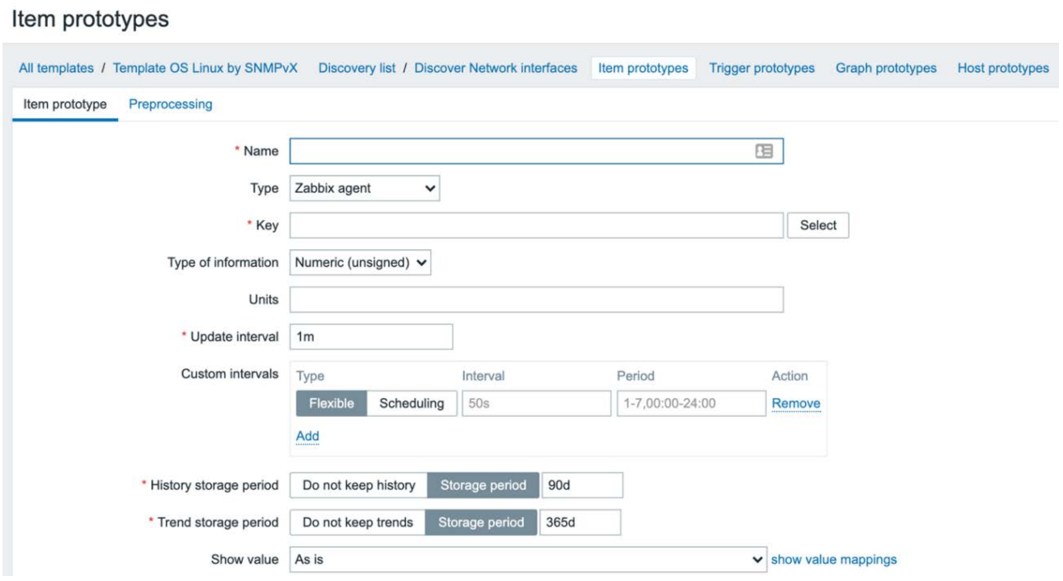
Figure 4.26 – Zabbix LLD creation page filled with our information for network interface discovery

4. After clicking the **Add** button, we can navigate back to our template at **Configuration | Templates** and click **Template OS Linux by SNMPvX**.

> **Important note**
>
> We define a **Keep lost resources period** of 0 days; we do this because this is a test template. Using 0 days can lead to lost resources, so make sure to adjust this value to your production environment's standard.

5. Go to **Discovery rules** and click our newly created rule, **Discover Network interfaces**.

6. Now, we will go to **Item prototypes** and click **Create item prototype** in the top-right corner. This will take us to the **Item prototype** creation screen, as shown in the following screenshot:

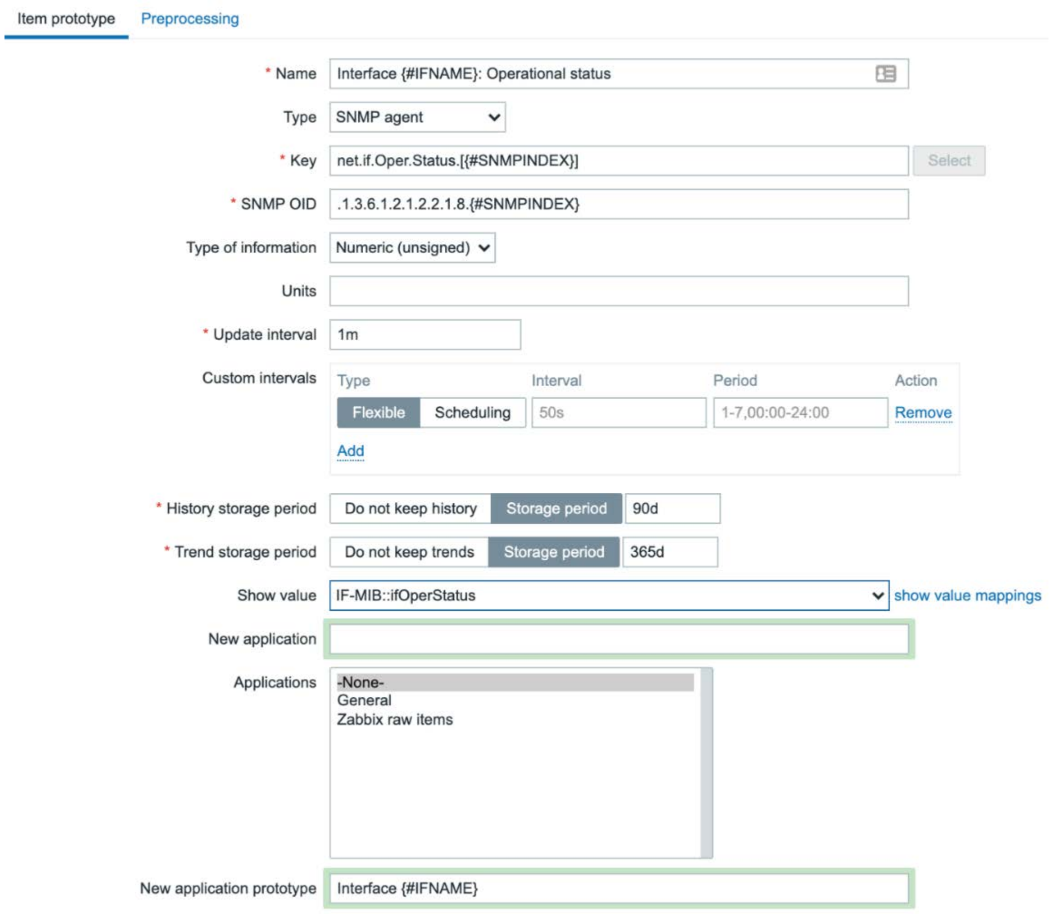


* Name	
Type	Zabbix agent
* Key	
Type of information	Numeric (unsigned)
Units	
* Update interval	1m
Custom intervals	Type: Flexible / Scheduling; Interval: 50s; Period: 1-7,00:00-24:00; Action: Remove
* History storage period	Do not keep history / Storage period: 90d
* Trend storage period	Do not keep trends / Storage period: 365d
Show value	As is

Figure 4.27 – Zabbix LLD Item prototype creation page, empty

Here, we will create our first prototype for creating items from LLD. This means we have to fill it with the information we want our items to contain.

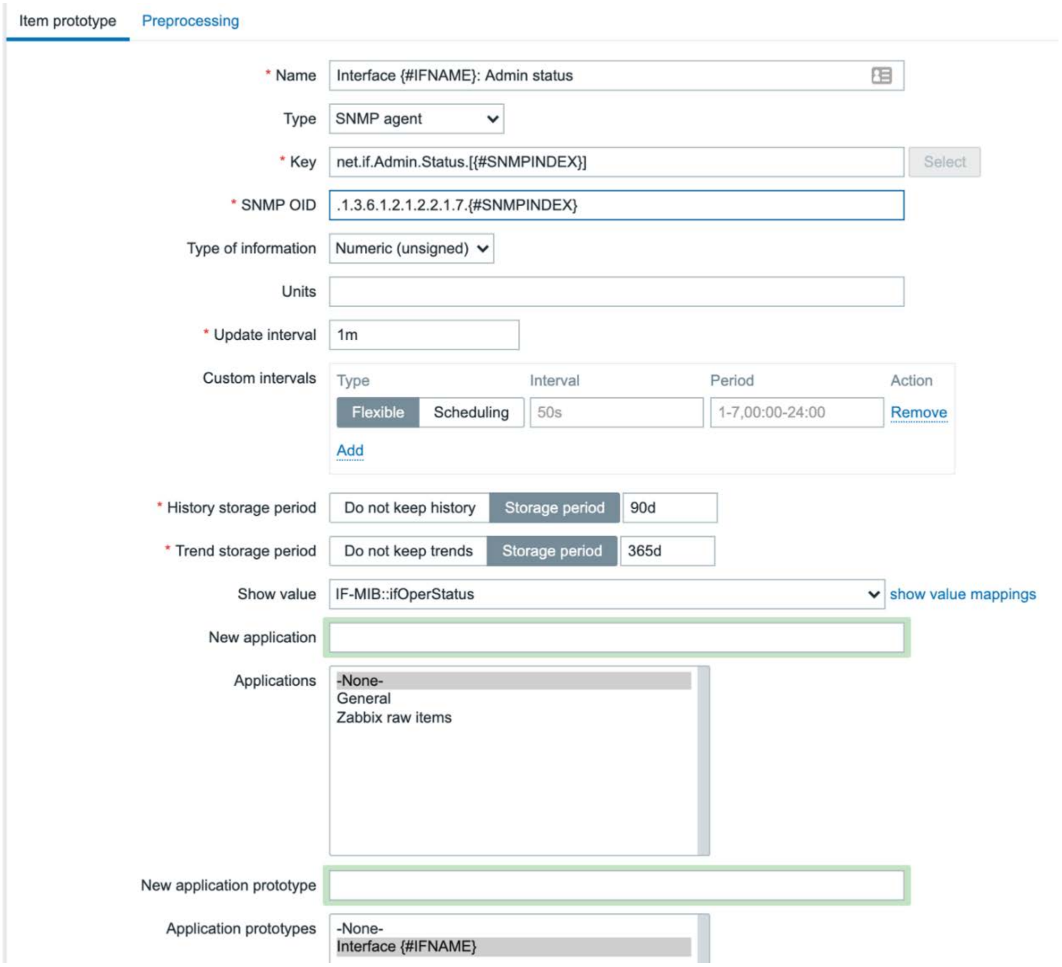
7. Let's start by filling in an item prototype for the interface operational status, like this:



* Name	Interface {#IFNAME}: Operational status
Type	SNMP agent
* Key	net.if.Oper.Status.[{#SNMPINDEX}]
* SNMP OID	.1.3.6.1.2.1.2.2.1.8.{#SNMPINDEX}
Type of information	Numeric (unsigned)
Units	
* Update interval	1m
Custom intervals	Type: Flexible / Scheduling; Interval: 50s; Period: 1-7,00:00-24:00; Action: Remove
* History storage period	Do not keep history / Storage period: 90d
* Trend storage period	Do not keep trends / Storage period: 365d
Show value	IF-MIB::ifOperStatus
New application	
Applications	-None-, General, Zabbix raw items
New application prototype	Interface {#IFNAME}

Figure 4.28 – Zabbix LLD item prototype creation page filled with our information for interface operational status

8. After clicking the **Add** button, let's also add the following item prototype:

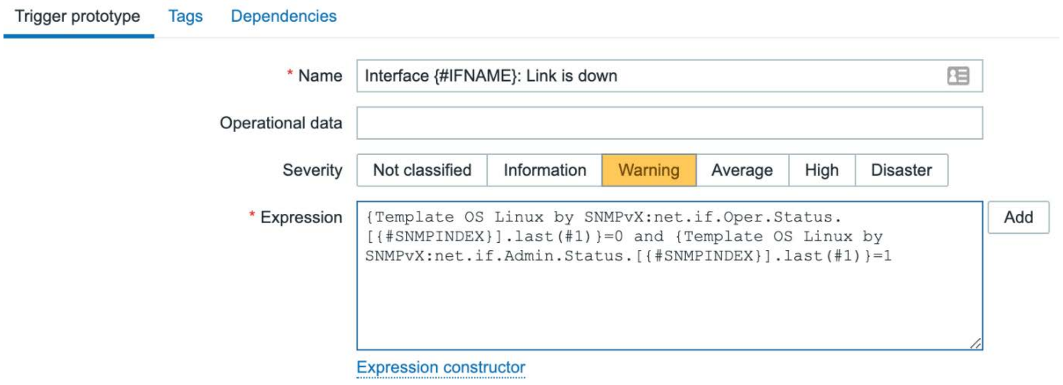


Item prototype | Preprocessing

* Name	Interface {#IFNAME}: Admin status
Type	SNMP agent
* Key	net.if.Admin.Status.[{#SNMPINDEX}]
* SNMP OID	.1.3.6.1.2.1.2.2.1.7.{#SNMPINDEX}
Type of information	Numeric (unsigned)
Units	
* Update interval	1m
Custom intervals	Type: Flexible / Scheduling; Interval: 50s; Period: 1-7,00:00-24:00; Action: Remove; Add
* History storage period	Do not keep history / Storage period: 90d
* Trend storage period	Do not keep trends / Storage period: 365d
Show value	IF-MIB::ifOperStatus (show value mappings)
New application	
Applications	-None-, General, Zabbix raw items
New application prototype	
Application prototypes	-None-, Interface {#IFNAME}

Figure 4.29 – Zabbix LLD item prototype creation page filled with our information for interface admin status

9. Last, but not least, move over to the **Trigger Prototype** page and click the **Create trigger** prototype button in the top-right corner, and create the following trigger:



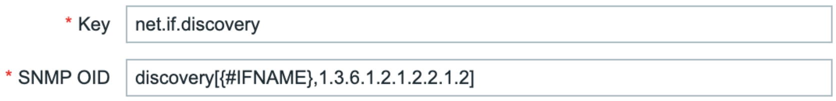
* Name	Interface {#IFNAME}: Link is down
Operational data	
Severity	Warning
* Expression	{Template OS Linux by SNMPvX:net.if.Oper.Status.[{#SNMPINDEX}].last(#1)}=0 and {Template OS Linux by SNMPvX:net.if.Admin.Status.[{#SNMPINDEX}].last(#1)}=1

Figure 4.30 – Zabbix LLD trigger prototype creation page filled with our information for interface link status

How it works...

Now, LLD is quite an extensive topic in Zabbix, but by following the steps in this recipe you should be able to build almost every form of LLD there is to configure in Zabbix. First of all, let's look at how the discovery works.

In the discovery rule, we just configured the following:



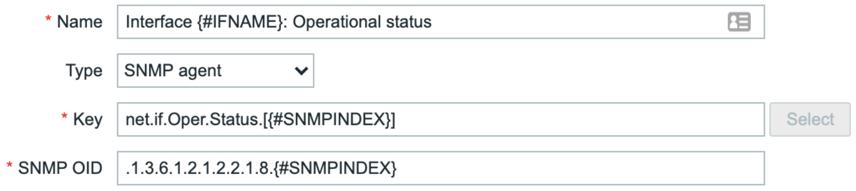
* Key	net.if.discovery
* SNMP OID	discovery[{#IFNAME},1.3.6.1.2.1.2.2.1.2]

Figure 4.31 – Zabbix LLD discovery key and OID for key net.if.discovery

What we are basically saying here is for every interface after OID `.1.3.6.1.2.1.2.2.1.2`, fill the `{#IFNAME}` LLD macro. In our case, this would be the following OIDs:

```
.1.3.6.1.2.1.2.2.1.2.1 = STRING: lo
.1.3.6.1.2.1.2.2.1.2.2 = STRING: ens192
```

So, we are saving these for use in our prototypes. Now, when we look at what we did at our **Operational status** prototype, this all comes together:



* Name	Interface {#IFNAME}: Operational status
Type	SNMP agent
* Key	net.if.Oper.Status.[{#SNMPINDEX}]
* SNMP OID	.1.3.6.1.2.1.2.2.1.8.{#SNMPINDEX}

Figure 4.32 – Zabbix LLD item prototype name, type, key, and OID

We are telling our item prototype to create an item for every single `{#IFNAME}` value using the key defined plus the `{#SNMPINDEX}` LLD macro. The SNMPINDEX is the last number of our SNMP poll. In this case, we would see the following:

```
.1.3.6.1.2.1.2.2.1.8.1 = INTEGER: up(1)
.1.3.6.1.2.1.2.2.1.8.2 = INTEGER: up(1)
```

For all the vendors in the world, there's a set of predefined SNMP rules they need to abide to. Our first interface entry when polling `.1.3.6.1.2.1.2.2.1.2` was the `.1` `SNMPINDEX` with the value `lo`. This means that when polling `.1.3.6.1.2.1.2.2.1.8`, the `.1` `SNMPINDEX` here should still contain a value for `lo`.

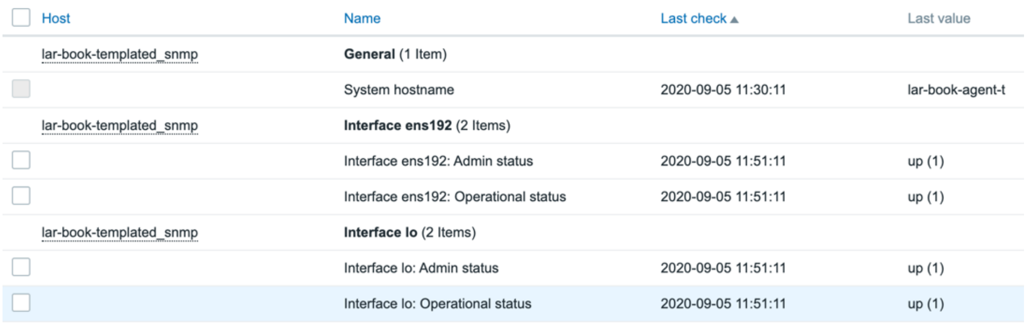
Zabbix LLD will now create an item with the name `Interface lo: Operational status`, which will poll the SNMP OID:

```
.1.3.6.1.2.1.2.2.1.8.1 = INTEGER: up(1)
```

It will also create an item with the name Interface `ens192`: Operational status, which will poll the SNMP OID:

```
.1.3.6.1.2.1.2.2.1.8.2 = INTEGER: up(1)
```

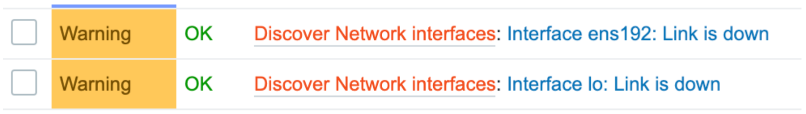
The created items will then look like this:



Host	Name	Last check ▲	Last value
lar-book-templated_snmp	**General** (1 Item)		
	System hostname	2020-09-05 11:30:11	lar-book-agent-t
lar-book-templated_snmp	**Interface ens192** (2 Items)		
	Interface ens192: Admin status	2020-09-05 11:51:11	up (1)
	Interface ens192: Operational status	2020-09-05 11:51:11	up (1)
lar-book-templated_snmp	**Interface lo** (2 Items)		
	Interface lo: Admin status	2020-09-05 11:51:11	up (1)
	Interface lo: Operational status	2020-09-05 11:51:11	up (1)

Figure 4.33 – Zabbix latest data screen for our SNMP-monitored host

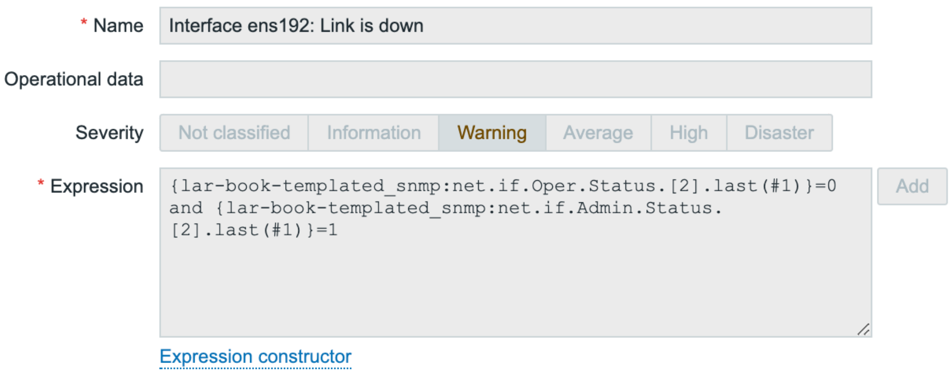
Besides creating these LLD items, we also created an LLD trigger prototype. This works in the same manner as item prototypes. If we check our host triggers, we can see two created triggers:



Warning	OK	Discover Network interfaces: Interface ens192: Link is down
Warning	OK	Discover Network interfaces: Interface lo: Link is down

Figure 4.34 – Our SNMP-monitored host triggers

These triggers have been created in the same manner as the items and are then filled with the correct items for triggering on:



* Name	Interface ens192: Link is down
Operational data	
Severity	Not classified / Information / [✓ Warning] / Average / High / Disaster
* Expression	{lar-book-templated_snmp:net.if.Oper.Status.[2].last(#1)}=0 and {lar-book-templated_snmp:net.if.Admin.Status.[2].last(#1)}=1

Figure 4.35 – Our SNMP-monitored host trigger for ens192

We can see that for Interface Operation Status we have an SNMPINDEX of 2, and for Interface Admin status as well. Our trigger will now trigger when the Operation status is `0` (`DOWN`) and our Admin status is `1` (`UP`).

A neat trigger, to make sure we only have a problem when the Admin status is `UP`; after all, we want our interface down when we configure it to be Admin `DOWN`.

Tip

It's possible to use discovery filters to only add the interfaces that have Admin status `UP` to our monitoring. This way, we keep our required Zabbix Server performance lower and our data cleaner. Consider using discovery filters for use cases such as this.

See also

Discovery is an extensive subject and takes a while to master. It's something that can be used like we did in this chapter with SNMP, but also with the Zabbix agent, and for a lot of other use cases. Once you start working with Zabbix discovery and you keep it structured, that's when you'll start building the best templates you've seen yet.

Check out the following link for the Zabbix LLD documentation:

`https://www.zabbix.com/documentation/current/manual/discovery/low_level_discovery`

Nesting Zabbix templates

Now, using a simple template per device or group of devices is one way to create Zabbix templates, but it isn't the only way. We can also use nested templates to break pieces of them apart and put them back together in the highest template in the nest.

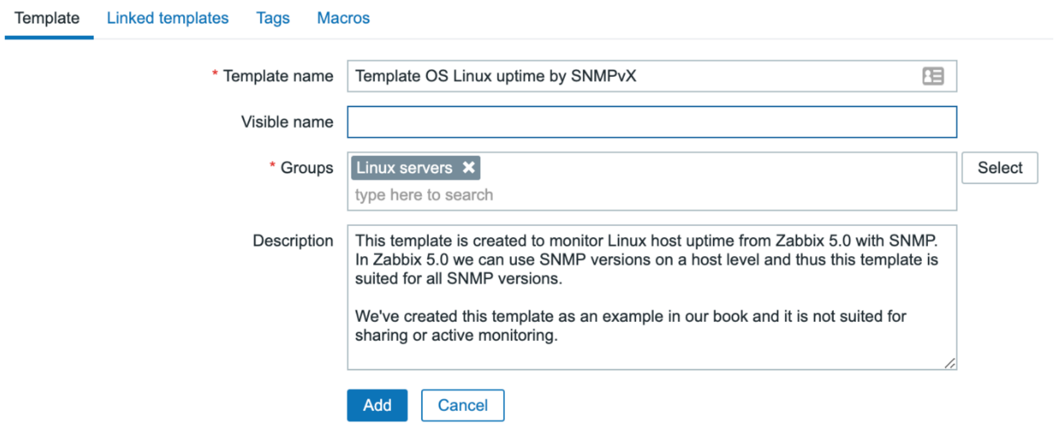
In this recipe, we'll go over how to configure this and why.

Getting ready

We are going to need our Zabbix server, our SNMP-monitored host, and the template we created in the previous recipe.

How to do it…

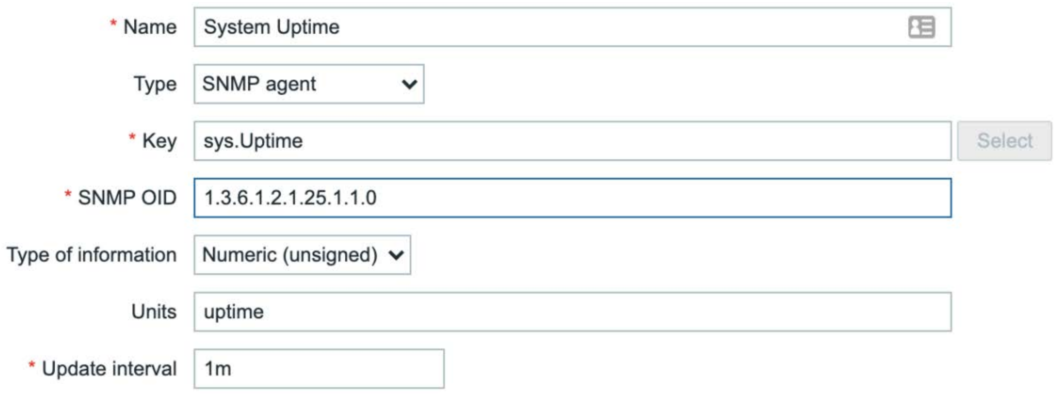
1. Let's start by navigating to our **Configuration** | **Templates** page and clicking the **Create template** button in the top-right corner.
2. We are going to create a new template for monitoring the uptime of our SNMP host. Input the following information:



* Template name	Template OS Linux uptime by SNMPvX
Visible name	
* Groups	Linux servers ✕ type here to search
Description	This template is created to monitor Linux host uptime from Zabbix 5.0 with SNMP. In Zabbix 5.0 we can use SNMP versions on a host level and thus this template is suited for all SNMP versions. We've created this template as an example in our book and it is not suited for sharing or active monitoring.

Figure 4.36 – New template creation page for uptime with SNMP

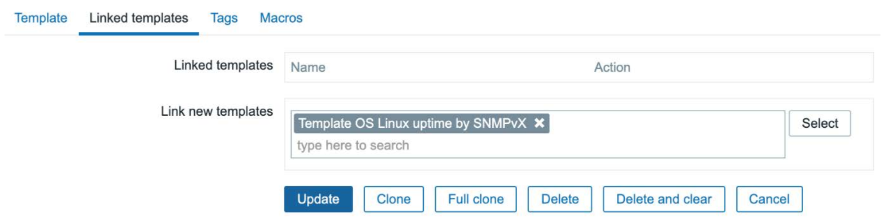
3. Next, we are going to click the **Add** button and click our **Template OS Linux uptime by SNMPvX** template name. This will take us to our template edit screen.
4. Click on **Items** and **Create item** in the top-right corner. We will create an example item here, like this:



* Name	System Uptime
Type	SNMP agent
* Key	sys.Uptime
* SNMP OID	1.3.6.1.2.1.25.1.1.0
Type of information	Numeric (unsigned)
Units	uptime
* Update interval	1m

Figure 4.37 – New item on template creation page, System Uptime

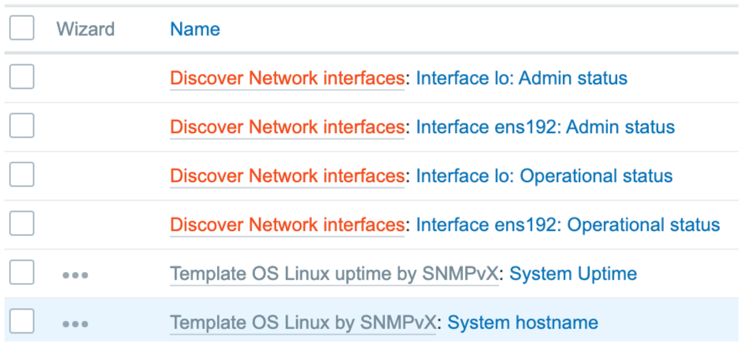
5. Now, let's navigate to our original template by going back to the **Configuration** | **Templates** page and clicking **Template OS Linux by SNMPvX**.
6. On this page, click **Linked templates** and add our new template as a linked template, like this:



Linked templates	Name	Action
Link new templates	Template OS Linux uptime by SNMPvX ✖ type here to search	

Figure 4.38 – Template link page for master SNMP template

7. Last, but not least, navigate to **Configuration** | **Hosts**, click our `lar-book-templated_snmp` SNMP-monitored host, and check out the **Items** page if the item is present:



	Wizard	Name
☐		Discover Network interfaces: Interface lo: Admin status
☐		Discover Network interfaces: Interface ens192: Admin status
☐		Discover Network interfaces: Interface lo: Operational status
☐		Discover Network interfaces: Interface ens192: Operational status
☐	•••	Template OS Linux uptime by SNMPvX: System Uptime
☐	•••	Template OS Linux by SNMPvX: System hostname

Figure 4.39 – Our Hosts | Items page for host lar-book-templated_snmp

The item is present, and it shows it's actually from another template. That's all there is to do to actually link a template—easy to work with but harder to keep it structured. Let's see how this works.

How it works...

Now, nesting templates works with a simple tree structure, just like this:

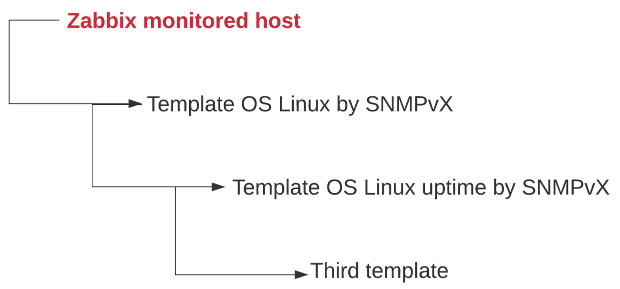

Zabbix monitored host
Template OS Linux by SNMPvX
Template OS Linux uptime by SNMPvX
Third template

Figure 4.40 – Template nesting tree structure

So, we have our Zabbix-monitored host, which in turn has **Template OS Linux by SNMPvX** linked as the only template. Now, because we have a nested template on Template OS Linux by `SNMPvX` (which is, of course, **Template OS Linux uptime by SNMPvX**), the items on that template will also be linked to our Zabbix-monitored host.

We can use this for a great deal of cases—one of my favorites is for networking equipment. If we have a Juniper EX (or Cisco Catalyst) and a Juniper QFX (or Cisco Nexus) series switch, both series of switches use the same SNMP discovery for interfaces. So, we can create a template for interfaces and nest this in the main template of the EX or QFX series, which do use different SNMP OIDs for other values.

This way, we don't have to write the same discovery rules, items, graphs, and everything else on a template a hundred times. We can simply do it once and nest the template neatly.

5
Visualizing Data, Inventory, and Reporting

When working with Zabbix, it is important that collected data is put to good use. After all, the data is of no use if we do not have a place to easily access it. Zabbix already puts our data to good use with the **Latest data** page and with problems created from triggers, but we can also put our data to good use by building some stuff ourselves, such as graphs, maps, and inventory, and by using reporting.

After working through these recipes, you'll be able to set up the most important parts of Zabbix data visualization. You'll also be able to make good use of your inventory and reporting systems, to get the most out of their useful features.

In this chapter, we'll go over these subjects to show you how to achieve good results in these recipes:

- Creating graphs for accessing visual data
- Creating maps to keep an eye on infrastructure

- Creating dashboards for getting the right overview
- Setting up Zabbix inventory
- Working through Zabbix reporting

Technical requirements

For this chapter, we will need our Zabbix server, our **System Network Management Protocol** (**SNMP**)-monitored host from *Chapter 4*, *Building Your Own Structured Templates*. We'll be doing most of our work in the frontend of Zabbix, so have your mouse at the ready.

The code files can also be accessed from the GitHub repository here:

`https://github.com/PacktPublishing/Zabbix-5-Network-Monitoring-Cookbook/tree/master/chapter5`

Creating graphs for accessing visual data

Graphs in Zabbix are a powerful tool, to show what's going on with your collected data. You might have already found some graphs by using the **Latest data** page, but we can also create our own predefined graphs. In this recipe, we will go over doing just that.

Getting ready

Make sure to get your Zabbix server ready, along with a Linux host that we can monitor with SNMP. If you have followed along with the recipes in *Chapter 4*, *Building Your Own Structured Templates*, you should already have your own personal created template. Alternatively, download the template here:

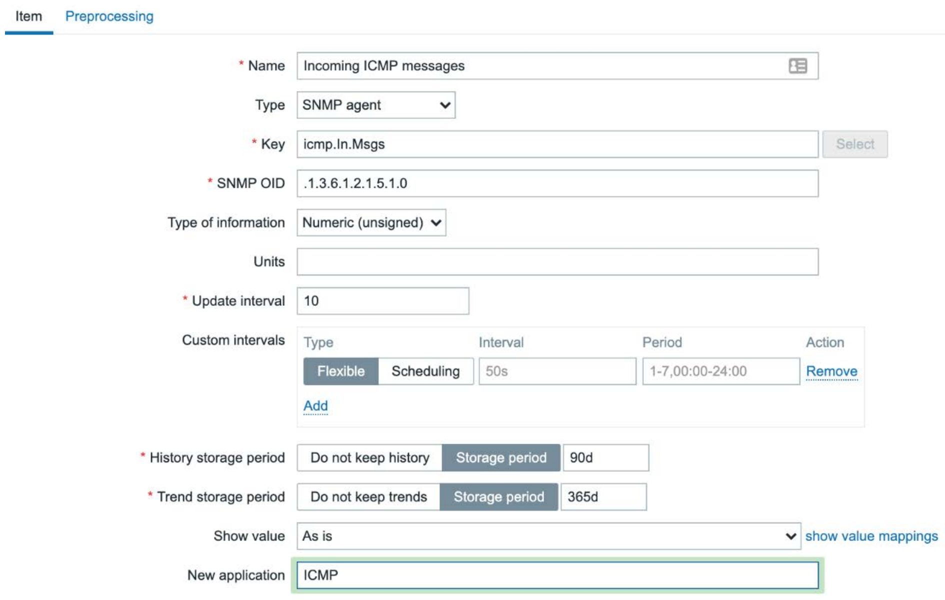
`https://github.com/PacktPublishing/Zabbix-5-Network-Monitoring-Cookbook`

If you are using the downloaded templates, download and import `Template OS Linux uptime by SNMPvX` first, and then `Template OS Linux by SNMPvX`. Importing a template can be done at **Configuration | Templates** by pressing the blue **Import** button in the top-right corner.

Make sure to put the template on a host and monitor it.

How to do it...

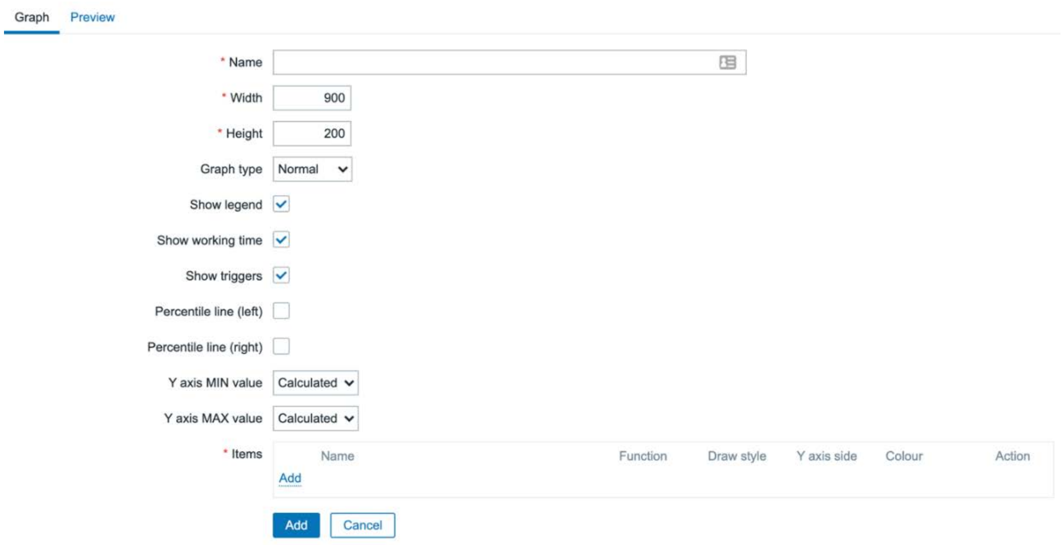
1. Let's start by navigating to our templates by going to **Configuration | Templates** and selecting the template. For me, it is still called **Template OS Linux by SNMPvX**.
2. Go to **Items** and create the following item on the template:



* Name	Incoming ICMP messages
Type	SNMP agent
* Key	icmp.In.Msgs
* SNMP OID	.1.3.6.1.2.1.5.1.0
Type of information	Numeric (unsigned)
Units	
* Update interval	10
Custom intervals	Type: Flexible; Interval: 50s; Period: 1-7,00:00-24:00
* History storage period	Storage period 90d
* Trend storage period	Storage period 365d
Show value	As is
New application	ICMP

Figure 5.1 – Item creation page

3. Now, back at the template configuration page, we go to **Graphs**. This is where we can see all of our configured graphs on this template; at the moment, there are none.
4. Click **Create graph** in the top-right corner. This will take you to the graph creation page:



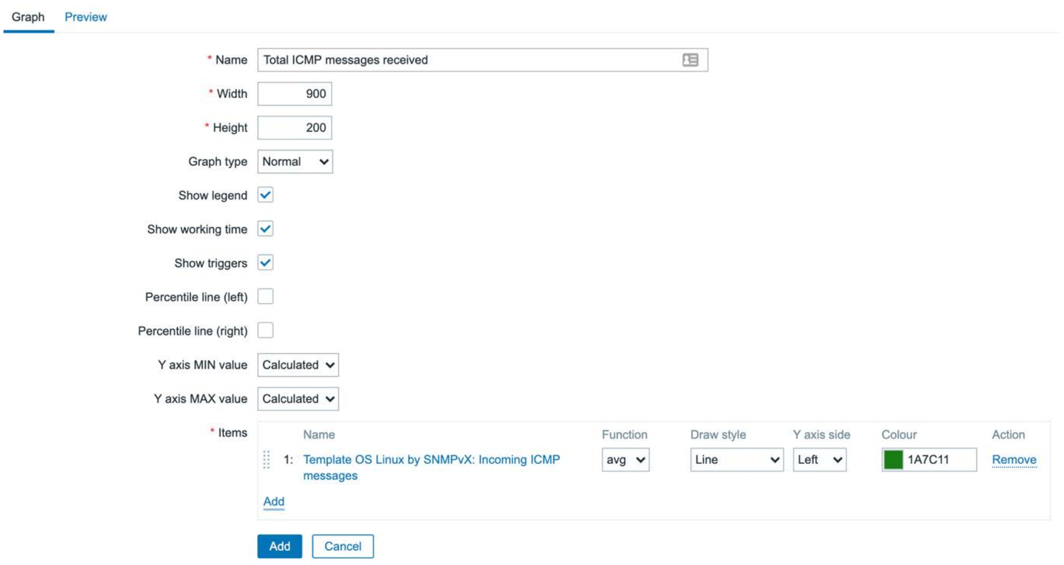
* Name	
* Width	900
* Height	200
Graph type	Normal
Show legend	✓
Show working time	✓
Show triggers	✓
Percentile line (left)	
Percentile line (right)	
Y axis MIN value	Calculated
Y axis MAX value	Calculated

Name	Function	Draw style	Y axis side	Colour	Action
Add					

Figure 5.2 – Graph creation page

This is where we can create graphs for standalone items. Let's create a graph to see our uptime.

5. Fill in the graph creation page with the following information:



* Name	Total ICMP messages received
* Width	900
* Height	200
Graph type	Normal
Show legend	✓
Show working time	✓
Show triggers	✓
Percentile line (left)	
Percentile line (right)	
Y axis MIN value	Calculated
Y axis MAX value	Calculated

* Items

	Name	Function	Draw style	Y axis side	Colour	Action
1:	Template OS Linux by SNMPvX: Incoming ICMP messages	avg	Line	Left	🟩 1A7C11	Remove

Figure 5.3 – Graph creation page filled with our information

Tip

When working with graphs, it's a good idea to keep colorblind people in mind. About 8% of all males and 0.5% of all females are affected by this condition. There are great sources online for reference of which colors to use for your production environment. You can find one such source here: `https://www.tableau.com/about/blog/2016/4/examining-data-viz-rules-dont-use-red-green-together-53463`.

6. Now, ping your SNMP-monitored host for a while. Do this from your Zabbix server **command-line interface** (**CLI**):

```
ping 10.16.16.153
```

7. Afterward, navigate to **Monitoring** | **Hosts** and click the **Graphs** button next to your host. In my case, the host is still called `lar-book-templated_snmp`.
8. Now, this will immediately take us to an overview of graphs for this host, where we
9. can see our new **Incoming ICMP messages** graph:

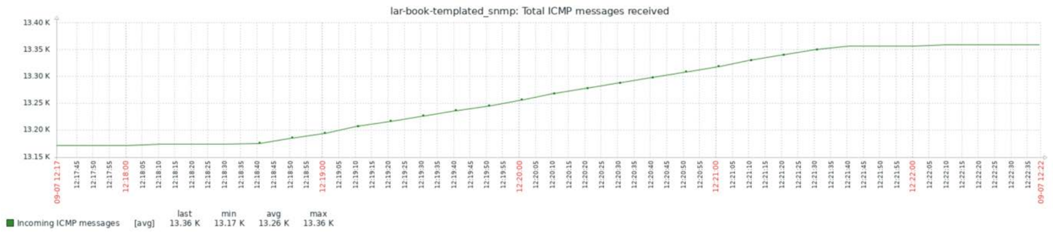
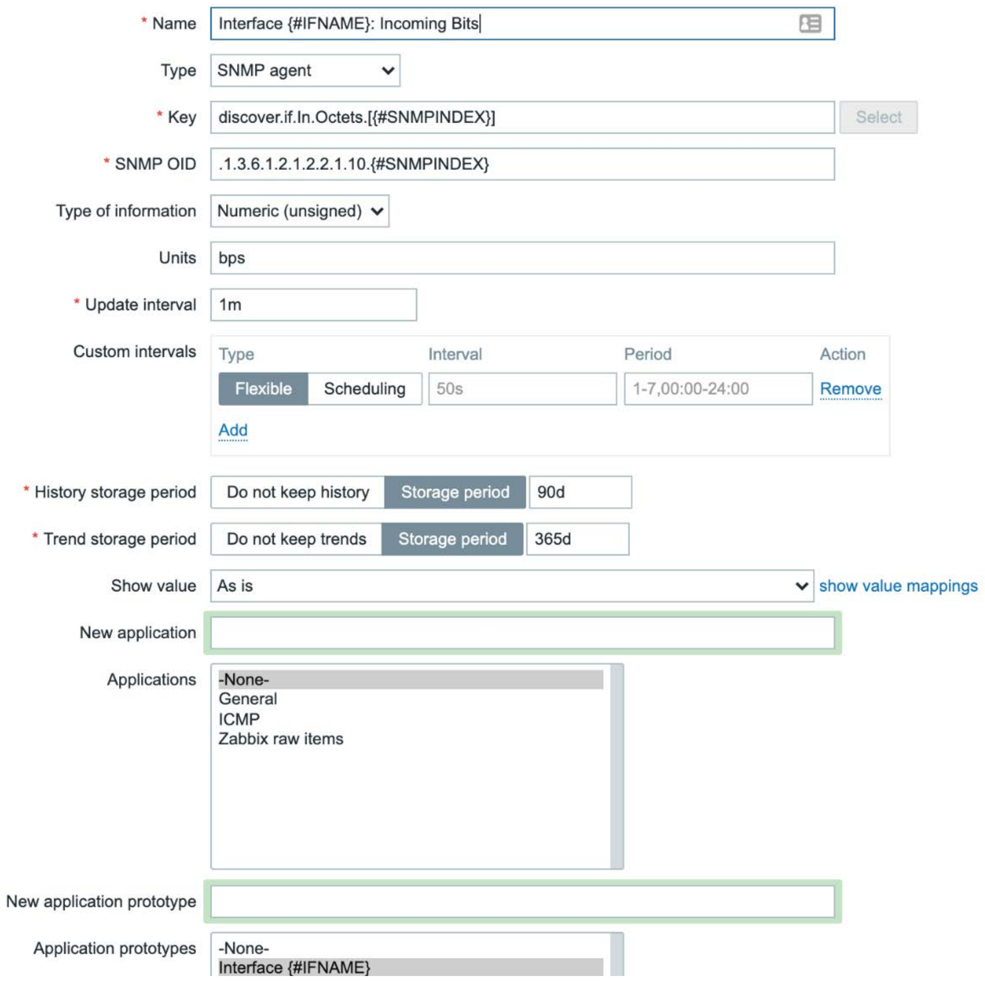


Figure 5.4 – Monitoring | Hosts graph page with our graph

We can also make graphs for discovery items; this is called a graph prototype. They work in about the same way as our item prototypes. Let's create one of these as well.

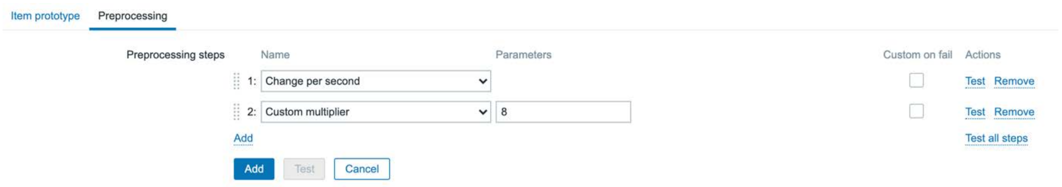
10. Navigate to **Configuration** | **Templates** and select our **Template OS Linux by SNMPvX** template.
11. Go to **Discovery rules**, and then, for the **Discover Network Interfaces** discovery rule, click on **Item prototypes**. In the top-right corner, click **Create item prototype** and create the following item prototype:



* Name	Interface {#IFNAME}: Incoming Bits
Type	SNMP agent
* Key	discover.if.In.Octets.[{#SNMPINDEX}]
* SNMP OID	.1.3.6.1.2.1.2.2.1.10.{#SNMPINDEX}
Type of information	Numeric (unsigned)
Units	bps
* Update interval	1m
Custom intervals	Type: Flexible; Interval: 50s; Period: 1-7,00:00-24:00; Action: Remove
* History storage period	Storage period: 90d
* Trend storage period	Storage period: 365d
Show value	As is
New application	
Applications	-None-, General, ICMP, Zabbix raw items
New application prototype	
Application prototypes	-None-, Interface {#IFNAME}

Figure 5.5 – Item prototype Incoming bits page filled with our information

12. Also, make sure to add the following to the **Preprocessing** page:



Item prototype | Preprocessing

Preprocessing steps

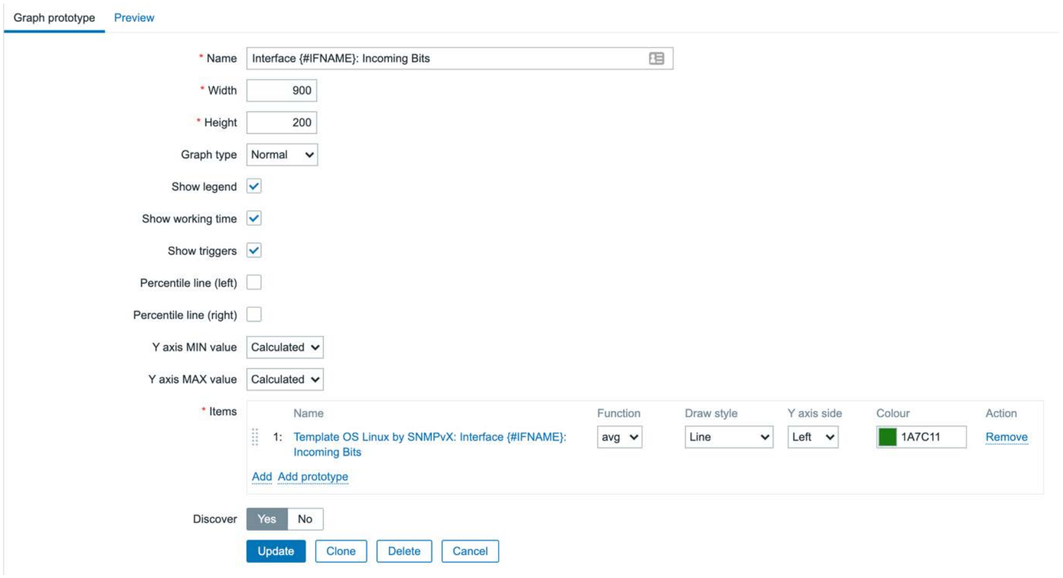
	Name	Parameters	Custom on fail	Actions
1:	Change per second		☐	Test Remove
2:	Custom multiplier	8	☐	Test Remove

Add — Test all steps

Add | Test | Cancel

Figure 5.6 – Item prototype Incoming bits Preprocessing page filled with our information

13. Now, back at **Discovery rules**, click the **Graph prototypes** button.
14. In the top-right corner, click **Create graph prototype** and fill the next page with the following information:



Graph prototype | Preview

* Name	Interface {#IFNAME}: Incoming Bits
* Width	900
* Height	200
Graph type	Normal
Show legend	✓
Show working time	✓
Show triggers	✓
Percentile line (left)	☐
Percentile line (right)	☐
Y axis MIN value	Calculated
Y axis MAX value	Calculated

* Items

	Name	Function	Draw style	Y axis side	Colour	Action
1:	Template OS Linux by SNMPvX: Interface {#IFNAME}: Incoming Bits	avg	Line	Left	🟩 1A7C11	Remove

Add Add prototype

Discover: [✓ Yes] No

Update | Clone | Delete | Cancel

Figure 5.7 – Graph prototype Incoming bits page filled with our information

15. Now, if we go back to **Monitoring | Hosts** and we click the **Graphs** button, we can see two new graphs:

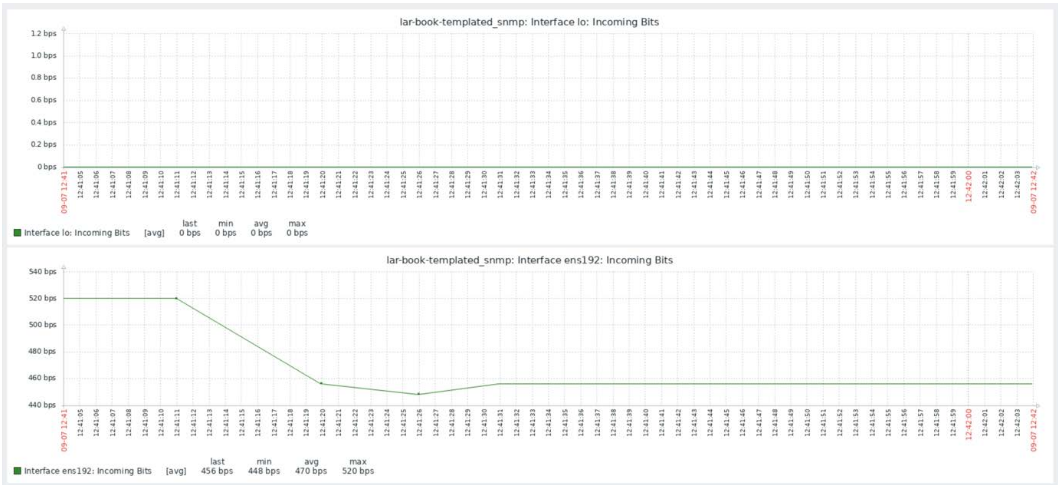


Figure 5.8 – Graphs page for our host

How it works...

Now, graphs work simply by putting your collected values in a visual form. We collect our data from our host—through SNMP, for example—and we put that data in our MySQL database. Our graphs then in turn collect this data from the MySQL database and put it in this visual form. For us, this is a lot better to read, and we can interpret the data easily.

The graph prototype works in almost the same way as our item prototype. For every single discovered interface, we create a graph using a name containing the `{#IFNAME}` **Low-Level Discovery** (**LLD**) macro. This way, we get a versatile structured environment because when a new interface is created (or deleted), a new graph is also created (or deleted).

Creating maps to keep an eye on infrastructure

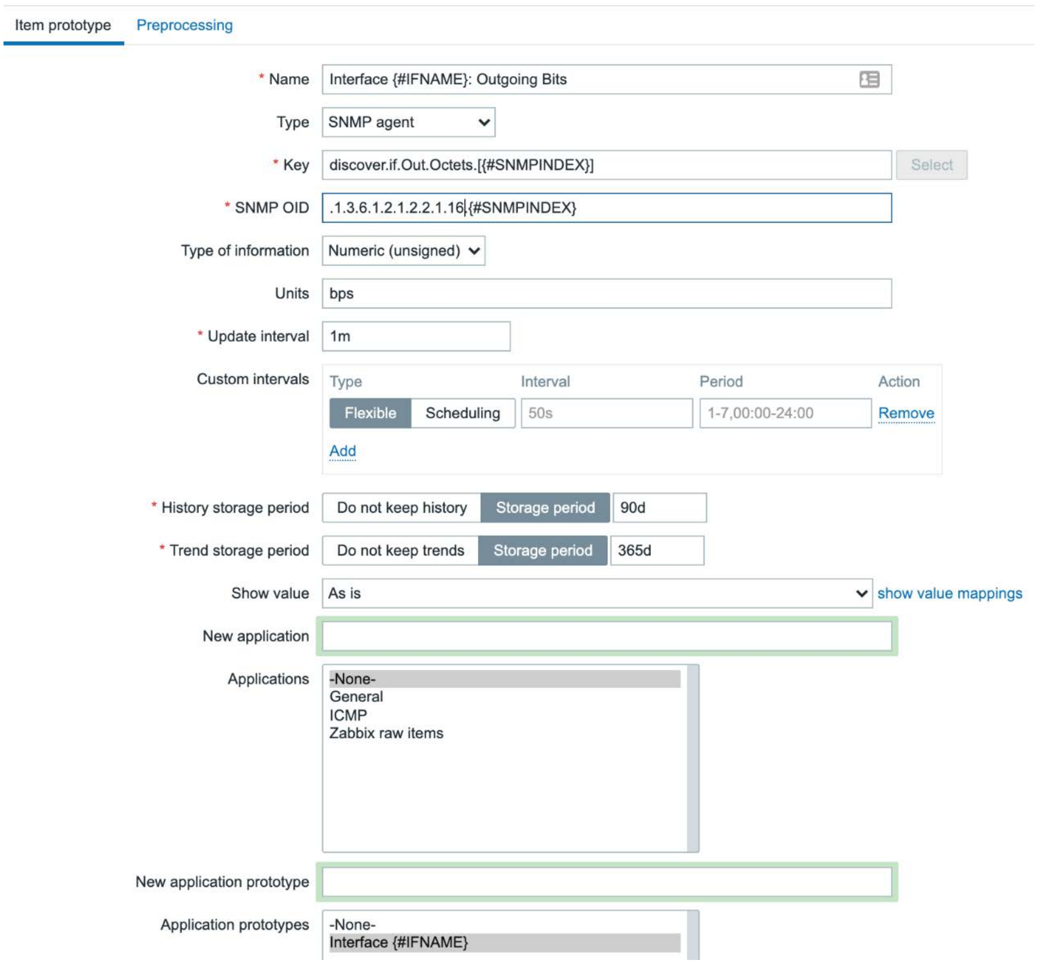
Maps in Zabbix are a great way to get an overview of infrastructure—for instance, they're amazing for following traffic flows or seeing where something is going off in your environment. They're not only super-useful for network overviews but also for server management overviews, and even for a lot of cool customization.

Getting ready

We will need our Zabbix server, our SNMP-monitored host, and the templates from the previous recipe.

How to do it...

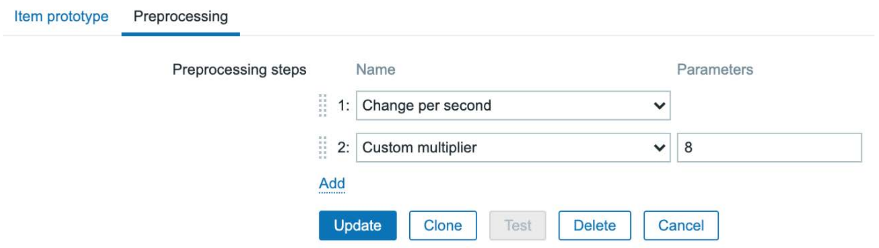
1. Let's start this recipe off by navigating to **Configuration | Template** and selecting our **Template OS Linux by SNMPvX** template.
2. Go to **Discovery rules** and then **Item prototypes**. Create the following item prototype by filling in the fields in the **Item prototype** creation page:



* Name	Interface {#IFNAME}: Outgoing Bits
Type	SNMP agent
* Key	discover.if.Out.Octets.[{#SNMPINDEX}]
* SNMP OID	.1.3.6.1.2.1.2.2.1.16{#SNMPINDEX}
Type of information	Numeric (unsigned)
Units	bps
* Update interval	1m
Custom intervals	Type: Flexible / Scheduling; Interval: 50s; Period: 1-7,00:00-24:00
* History storage period	Do not keep history / Storage period: 90d
* Trend storage period	Do not keep trends / Storage period: 365d
Show value	As is
New application	
Applications	-None-, General, ICMP, Zabbix raw items
New application prototype	
Application prototypes	-None-, Interface {#IFNAME}

Figure 5.9 – Item prototype creation page

3. Don't forget to add the **Preprocessing**:



Preprocessing steps	Name	Parameters
1:	Change per second	
2:	Custom multiplier	8

Figure 5.10 – Item prototype Preprocessing creation page

4. Click on the blue **Add** button to finish. Next up, navigate to **Monitoring | Maps**. There's already a default map here included in all Zabbix server installs, titled **Local network**. Feel free to check it out:

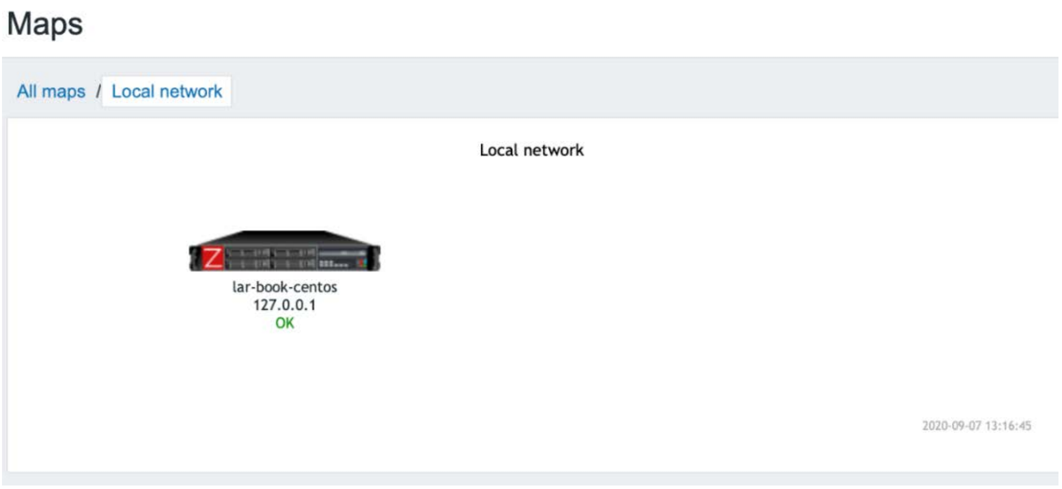
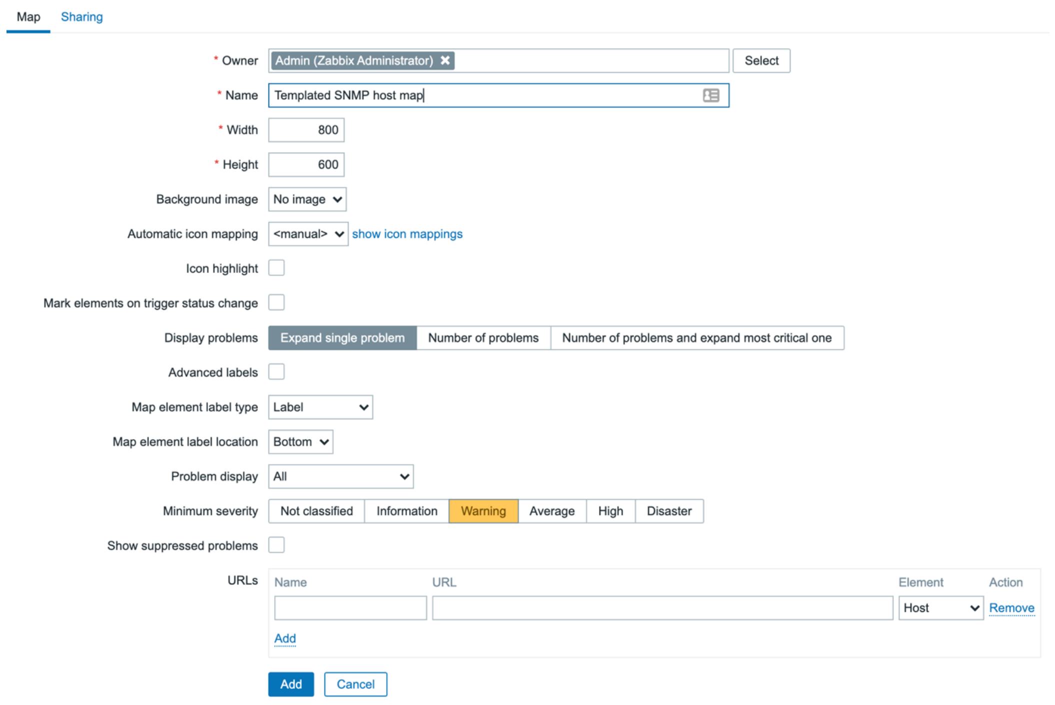


Figure 5.11 – The default Local network map

5. There's not much to see here, besides your local Zabbix server host and if it is in a problem state or not. So, let's click on **All maps**.

6. We are going to create our own map, so click the **Create map** button in the top-right corner. Create the map by filling in the following fields:



* Owner	Admin (Zabbix Administrator) ✖ Select
* Name	Templated SNMP host map
* Width	800
* Height	600
Background image	No image
Automatic icon mapping	<manual> show icon mappings
Icon highlight	☐
Mark elements on trigger status change	☐
Display problems	[✓ Expand single problem] Number of problems / Number of problems and expand most critical one
Advanced labels	☐
Map element label type	Label
Map element label location	Bottom
Problem display	All
Minimum severity	Not classified / Information / [✓ Warning] / Average / High / Disaster
Show suppressed problems	☐
URLs	Name / URL / Element: Host / Action: Remove / Add

Figure 5.12 – Map creation page

7. After clicking the blue **Add** button, the frontend will take you back to the **Map** overview page. Click the newly created `Templated SNMP host map` map here.
8. Click **Edit map** in the top-right corner to start editing the map.
9. Now, what we want to do here is select the **Add** button next to **Map element**. This will add the following element:

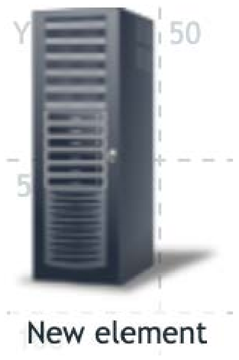
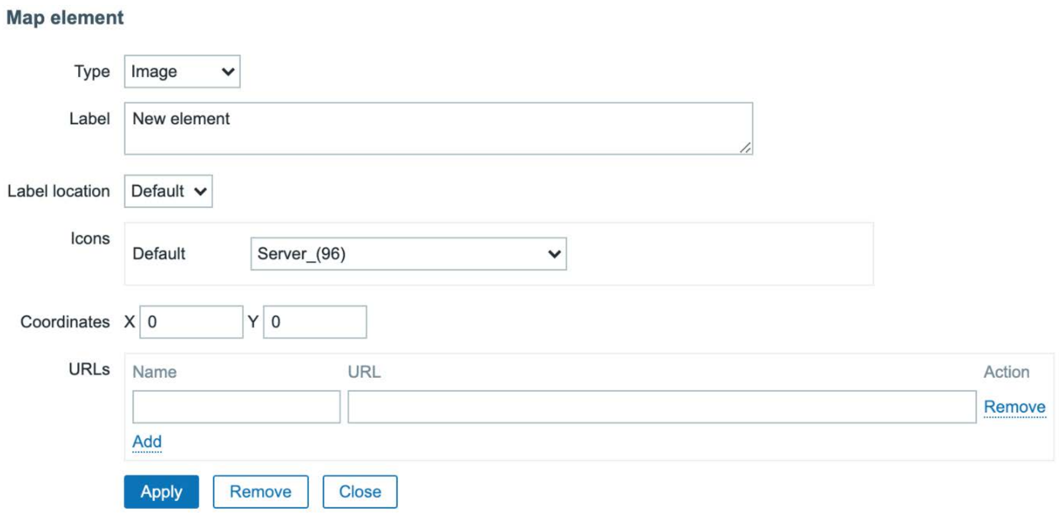


Figure 5.13 – The added element

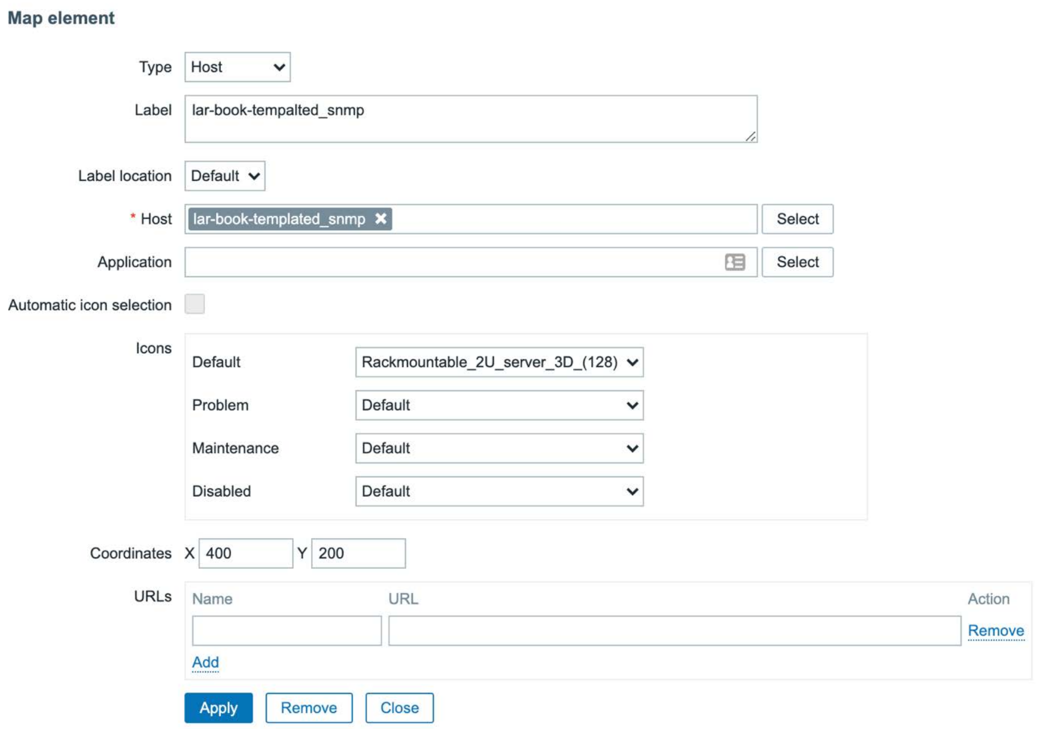
10. Click the newly added element—this will open the following screen:


Map element

Type	Image
Label	New element
Label location	Default
Icons	Default: Server_(96)
Coordinates	X 0 Y 0
URLs	Name / URL / Action: Remove

Figure 5.14 – New Map element edit window

11. Now, here we can fill out our host information. Let's add the following information to the fields:

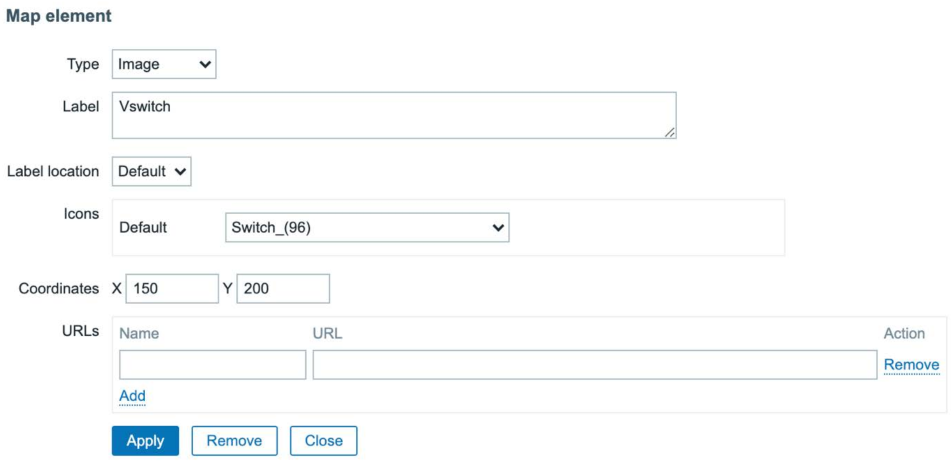


Map element

Type	Host
Label	lar-book-tempalted_snmp
Label location	Default
* Host	lar-book-templated_snmp
Application	
Automatic icon selection	☐
Icons	Default: Rackmountable_2U_server_3D_(128); Problem: Default; Maintenance: Default; Disabled: Default
Coordinates	X 400 Y 200
URLs	Name / URL / Action: Remove; Add

Figure 5.15 – Map element lar-book-templated_snmp edit window

12. Click **Apply** and move the element by dragging it to **X**:`400` and **Y**:`200` (see *Figure 5.16*). Now, add another element by clicking the **Add** button next to **Map element**. Edit the new element and add the following information:

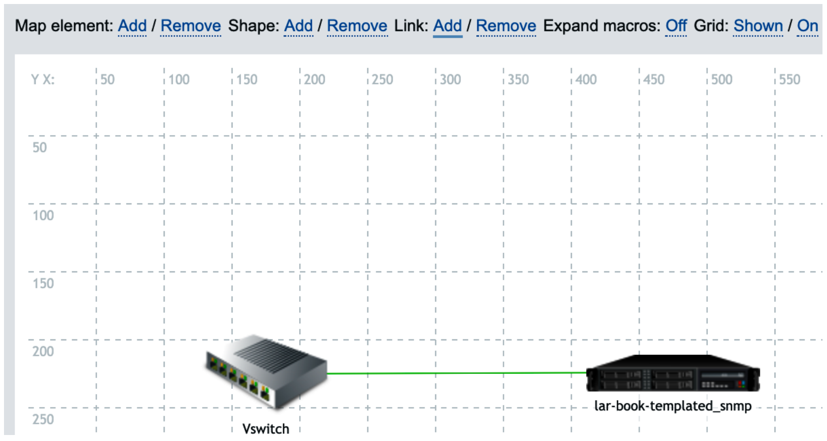


Map element

Type	Image
Label	Vswitch
Label location	Default
Icons	Default: Switch_(96)
Coordinates	X 150 Y 200
URLs	Name / URL / Action: Remove; Add

Figure 5.16 – Map element Vswitch edit window filled with information

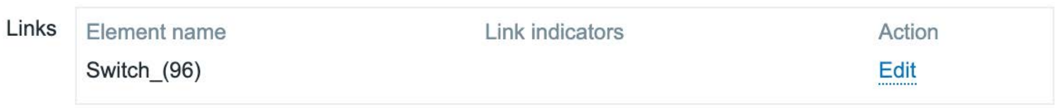
13. After creating both elements, let's move the new switch element to **X**:`150` and **Y**:`200`, as seen in *Figure 5.16*.
14. Now, select both elements by holding the *Ctrl* key (*Command* on a Mac) on your keyboard.
15. Then, click **Add** next to **Link** to add a link between the two elements. It should now look like this:



Map element: Add / Remove Shape: Add / Remove Link: Add / Remove Expand macros: Off Grid: Shown / On
Y X: 50 100 150 200 250 300 350 400 450 500 550
50
100
150
200
250
Vswitch
lar-book-templated_snmp

Figure 5.17 – Our newly created map

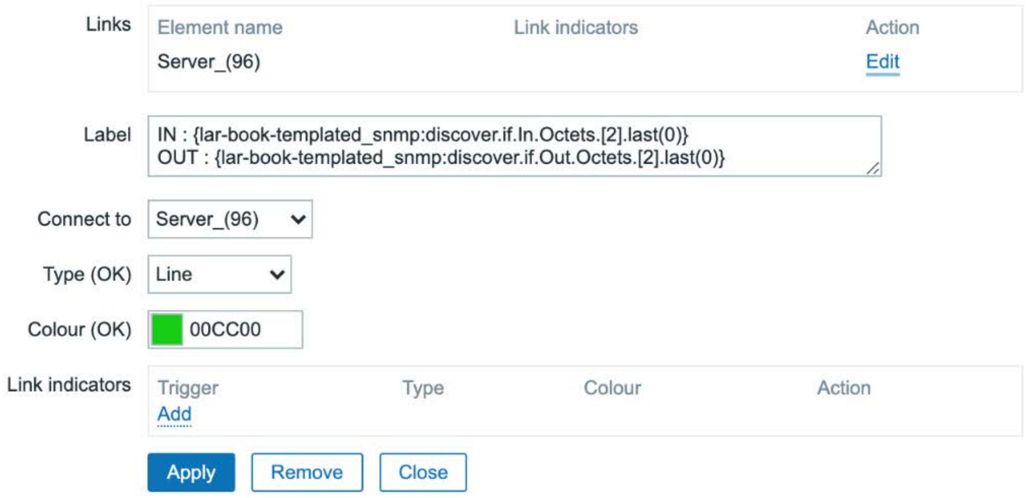
16. Edit the element for our server again after creating the link. Click on **Edit** next to the newly created link, as shown in the following screenshot:



Links	Element name	Link indicators	Action
	Switch_(96)		Edit

Figure 5.18 – Edit link in the Map element edit window

17. Add the following information to the window:



Links	Element name	Link indicators	Action
	Server_(96)		Edit

Label: IN : {lar-book-templated_snmp:discover.if.In.Octets.[2].last(0)}
OUT : {lar-book-templated_snmp:discover.if.Out.Octets.[2].last(0)}

Connect to: Server_(96)

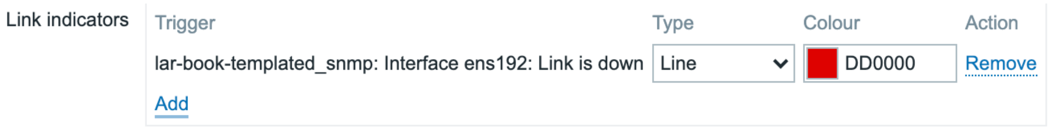
Type (OK): Line

Colour (OK): 🟩 00CC00

Link indicators	Trigger	Type	Colour	Action
	Add			

Figure 5.19 – Edit link in the Map element edit window with our information

18. Let's also click **Add** at the **Link indicators** section and add the following trigger with the color red:



Link indicators	Trigger	Type	Colour	Action
	lar-book-templated_snmp: Interface ens192: Link is down	Line	🟥 DD0000	Remove
	Add			

Figure 5.20 – Link indicator filled with a trigger

19. Now, click **Apply** at the bottom of the window and then **Update** in the top-right corner of the page. That's our first map created!

How it works...

Now, after creating and opening our map, we can see the following:

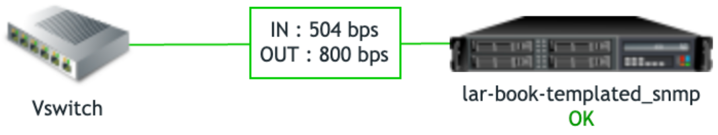
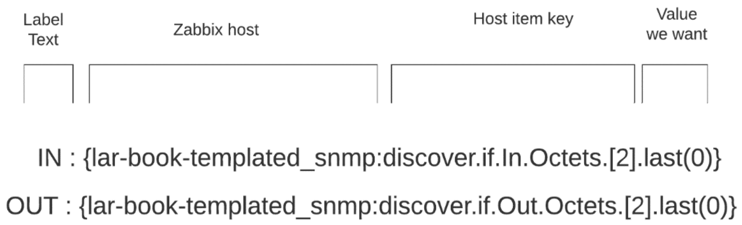


Figure 5.21 – Our newly created map

The map shows our switch (which is not a monitored host at the moment) and our server (which is a monitored host). This means that when something is wrong with our server, the **OK** status will turn into a **PROBLEM** status on the map.

We can also see our configured label (*see Figure 5.18*), which is showing us real-time information of traffic statistics. Now, when we break down the label, we get the following:



```
IN : {lar-book-templated_snmp:discover.if.In.Octets.[2].last(0)}
OUT : {lar-book-templated_snmp:discover.if.Out.Octets.[2].last(0)}
```

Figure 5.22 – Map label breakdown

We can pull real-time statistics into a label by defining which statistics we want to pull into the label between { }. In this case, we collect our values for interface traffic and put them directly in the label, creating a real-time traffic analyzation map.

We also put a trigger on this link. The cool thing about putting triggers such as this on our map is that when our link goes down, we can see the following happen:

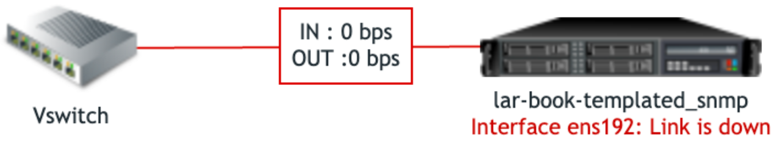


Figure 5.23 – Map showing problems

Traffic stopped flowing because the link is now down, and our line has turned red. Also, our host is now showing a **PROBLEM** state under the hostname.

We can even create orange lines with triggers that state 50% traffic utilization like this and trace **Distributed Denial of Service** (**DDoS**) traffic through our network.

Creating dashboards for getting the right overview

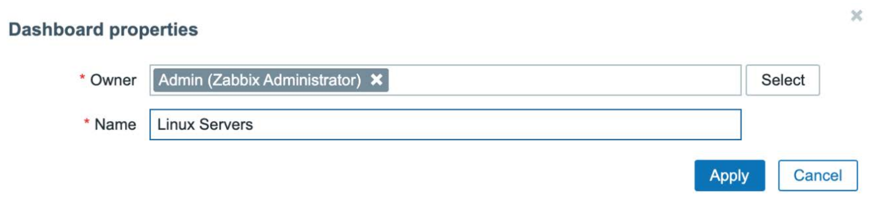
Now that we've created some graphs and a map, let's continue on by not only visualizing our data but also getting the visualization in an overview. We're going to create a dashboard for our Linux-monitored hosts.

Getting ready

Make sure you followed the previous two recipes and that you have your Zabbix server ready. We'll be using our SNMP-monitored host from the previous recipe, as well as our item, triggers, and map.

How to do it…

1. Navigate to **Monitoring | Dashboards** and click **All dashboards** in the left corner of the page.
2. Now, click the **Create dashboard** button in the top-right corner and fill in your dashboard name, like this:



Dashboard properties

* Owner	Admin (Zabbix Administrator)
* Name	Linux Servers

Figure 5.24 – Create dashboard window

Tip

Keeping Zabbix elements such as maps and dashboards owned under the Zabbix **Admin** user is a great idea. This way, they aren't dependent on a single user who might leave your environment at a later stage. The elements can be owned by a disabled user as well.

3. Now, click **Apply**, and you'll be taken to your dashboard immediately:

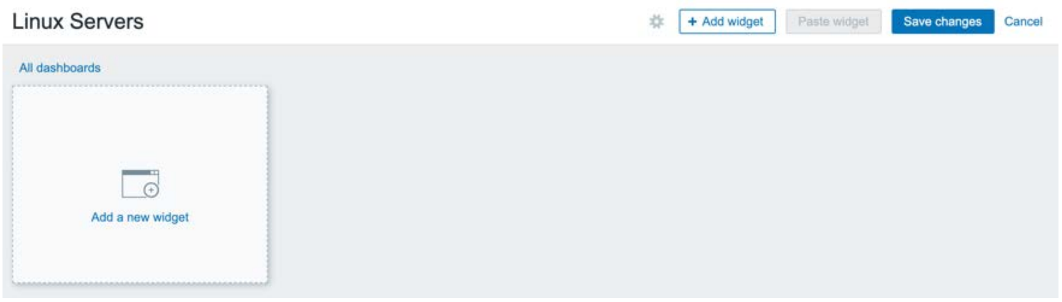
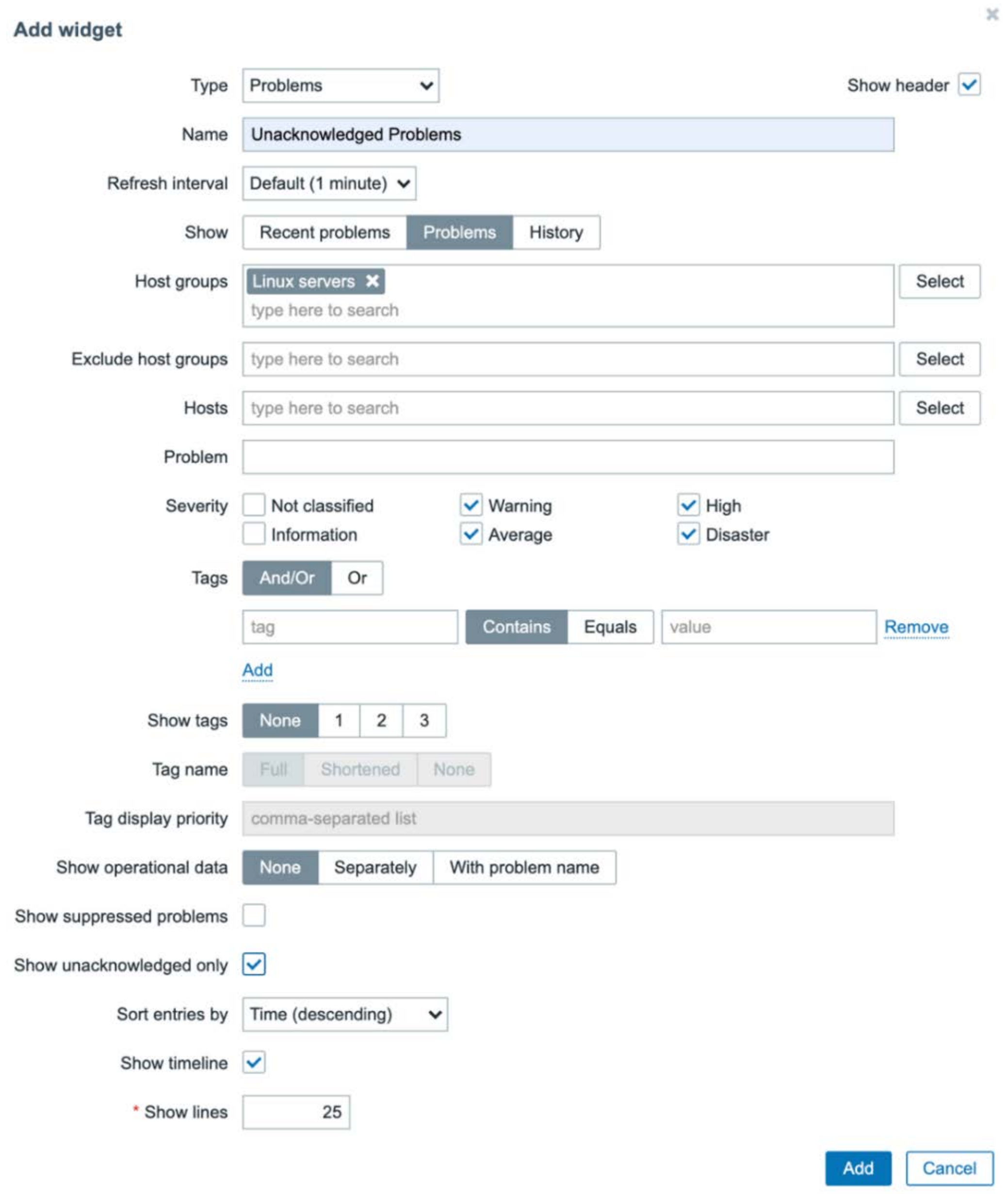


Figure 5.25 – New empty dashboard

After creating our dashboard, we will see that it is empty. We need to fill it with several widgets to create a good overview.

4. Let's start by adding a problem widget. Click **+ Add widget** in the top-right corner. Add the following widget by filling out all the fields:



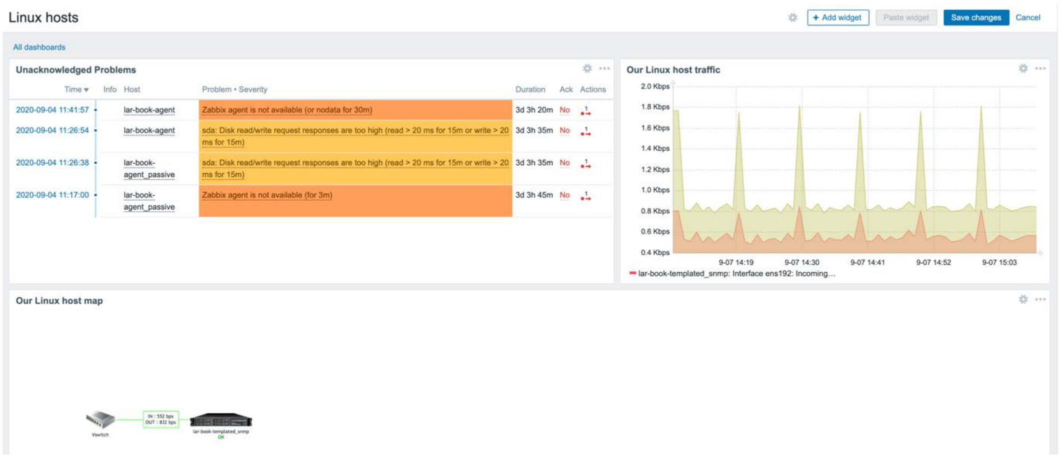
Add widget

Type	Problems
Show header	✓
Name	Unacknowledged Problems
Refresh interval	Default (1 minute)
Show	Recent problems [✓ Problems] History
Host groups	Linux servers
Exclude host groups	type here to search
Hosts	type here to search
Problem	
Severity	Not classified, Information, [✓ Warning], [✓ Average], [✓ High], [✓ Disaster]
Tags	[✓ And/Or] Or
	tag [✓ Contains] Equals value Remove
	Add
Show tags	[✓ None] 1 2 3
Tag name	Full Shortened None
Tag display priority	comma-separated list
Show operational data	[✓ None] Separately With problem name
Show suppressed problems	
Show unacknowledged only	✓
Sort entries by	Time (descending)
Show timeline	✓
* Show lines	25

Add Cancel

Figure 5.26 – New problem widget creation window

5. Click **Add**, and we have our first widget on our dashboard, displaying all **Unacknowledged Problems**. It will only show them for the **Severity** warning and higher on all `Linux servers`:

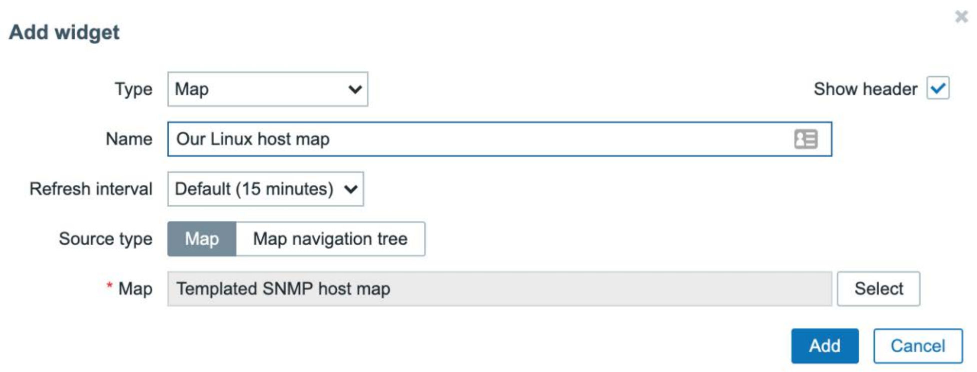


Unacknowledged Problems

Time ▾	Info	Host	Problem • Severity	Duration	Ack	Actions
2020-09-04 11:41:57		lar-book-agent	Zabbix agent is not available (or nodata for 30m)	3d 3h 20m	No	1
2020-09-04 11:26:54		lar-book-agent	sda: Disk read/write request responses are too high (read > 20 ms for 15m or write > 20 ms for 15m)	3d 3h 35m	No	1
2020-09-04 11:26:38		lar-book-agent_passive	sda: Disk read/write request responses are too high (read > 20 ms for 15m or write > 20 ms for 15m)	3d 3h 35m	No	1
2020-09-04 11:17:00		lar-book-agent_passive	Zabbix agent is not available (for 3m)	3d 3h 45m	No	1

Figure 5.27 – Our Unacknowledged Problems widget

6. Let's immediately add some more widgets, starting with our `Map` widget. Click **+ Add widget** in the top-right corner, and add the following widget:

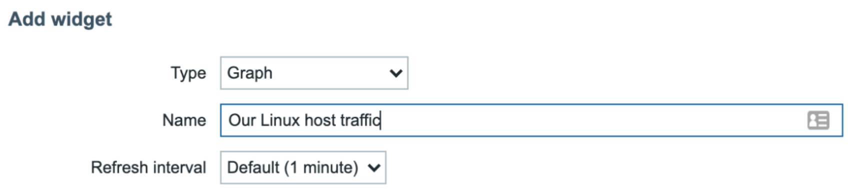


Add widget

Type	Map	Show header	✓
Name	Our Linux host map		
Refresh interval	Default (15 minutes)		
Source type	[✓ Map] Map navigation tree		
* Map	Templated SNMP host map	Select	

Figure 5.28 – New Map widget creation window

7. Also, add a Graph type widget by clicking **+ Add widget** in the top-right corner again. This one is a bit more difficult. Let's add our name first:

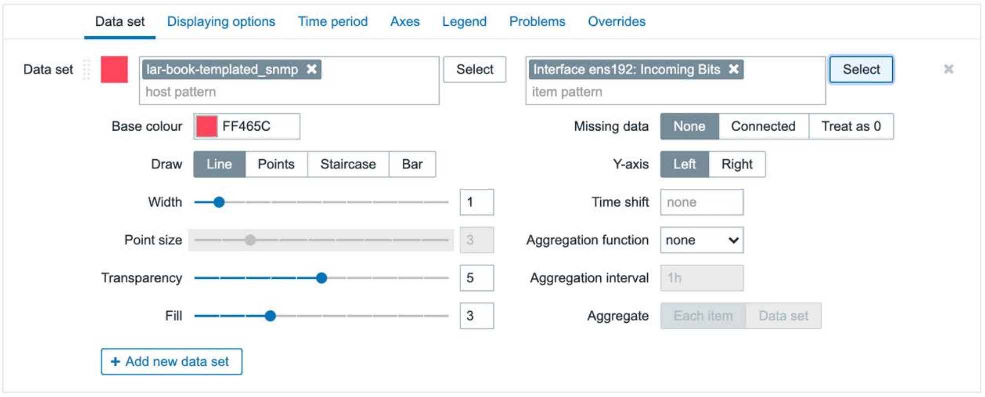


Add widget

Type	Graph
Name	Our Linux host traffic
Refresh interval	Default (1 minute)

Figure 5.29 – New Graph widget creation window

8. Then, we need to add our first **Data set**, like this:



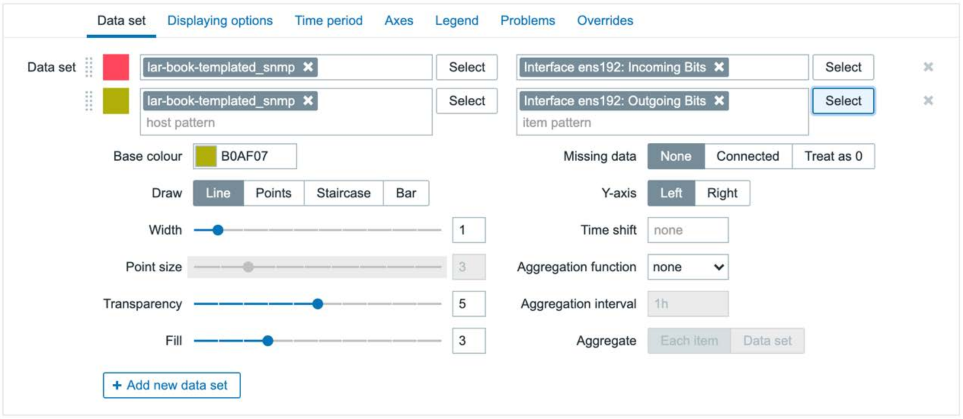
Data set | Displaying options | Time period | Axes | Legend | Problems | Overrides

Data set	lar-book-templated_snmp (host pattern)	Interface ens192: Incoming Bits (item pattern)	
Base colour	FF465C	Missing data	None
Draw	Line	Y-axis	Left
Width	1	Time shift	none
Point size	3	Aggregation function	none
Transparency	5	Aggregation interval	1h
Fill	3	Aggregate	Each item / Data set

+ Add new data set

Figure 5.30 – Adding dataset in New Graph widget creation window

9. Then, a second one, by clicking + **Add new data set** and adding the following:



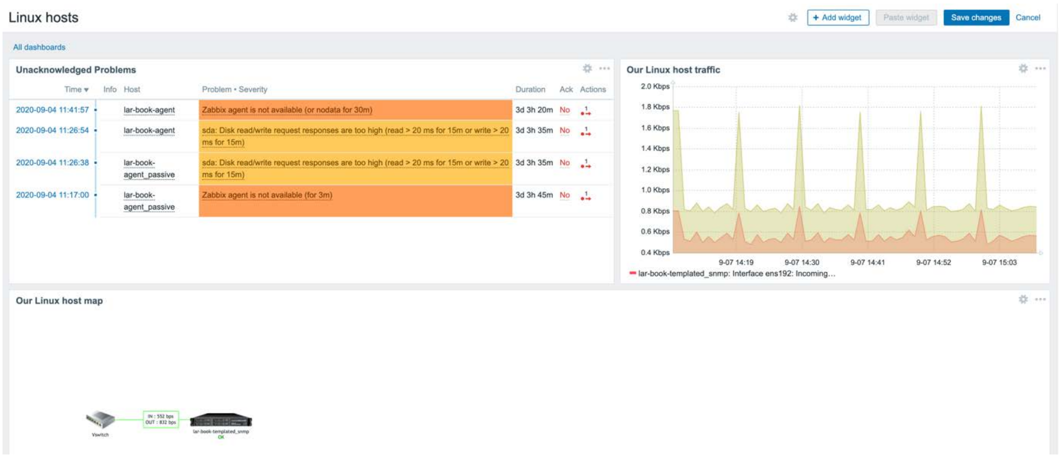
Data set | Displaying options | Time period | Axes | Legend | Problems | Overrides

Data set	lar-book-templated_snmp	Interface ens192: Incoming Bits	
	lar-book-templated_snmp (host pattern)	Interface ens192: Outgoing Bits (item pattern)	
Base colour	B0AF07	Missing data	None
Draw	Line	Y-axis	Left
Width	1	Time shift	none
Point size	3	Aggregation function	none
Transparency	5	Aggregation interval	1h
Fill	3	Aggregate	Each item / Data set

+ Add new data set

Figure 5.31 – Adding another dataset in New Graph widget creation window

10. We can then press **Add**, and our graph will be added to our dashboard. We can then freely move around the widgets until we get this, for example:



Unacknowledged Problems

Time ▾	Info	Host	Problem • Severity	Duration	Ack	Actions
2020-09-04 11:41:57		lar-book-agent	Zabbix agent is not available (or nodata for 30m)	3d 3h 20m	No	1
2020-09-04 11:26:54		lar-book-agent	sda: Disk read/write request responses are too high (read > 20 ms for 15m or write > 20 ms for 15m)	3d 3h 35m	No	1
2020-09-04 11:26:38		lar-book-agent_passive	sda: Disk read/write request responses are too high (read > 20 ms for 15m or write > 20 ms for 15m)	3d 3h 35m	No	1
2020-09-04 11:17:00		lar-book-agent_passive	Zabbix agent is not available (for 3m)	3d 3h 45m	No	1

Figure 5.32 – Our dashboard with information

11. Press **Save changes** in the top-right corner, and you are done.

How it works...

Creating dashboards is the best way to create overviews for quick access to data during troubleshooting, day-to-day problem monitoring, and—of course—for use with big TV walls. We've probably all seen the big operation centers with TVs displaying data. Zabbix is great for all these purposes and more, as you can see from this recipe.

Important note

Sometime in the future, Zabbix will remove the **Screens** functionality, which is why we do not cover it in this book. **Screens** will be replaced by **Dashboards**, and thus will focus building on dashboards only. **Screens** does have the additional **slideshow** functionality, but we can use external browser extensions to achieve the same effect with **Dashboards**.

Setting up Zabbix inventory

Zabbix Inventory is a feature I personally love, but it hasn't had a lot of love from the Zabbix development team lately. Sorry—I still love you, Zabbix developers, but if you're reading this, feel free to put some time into the feature!

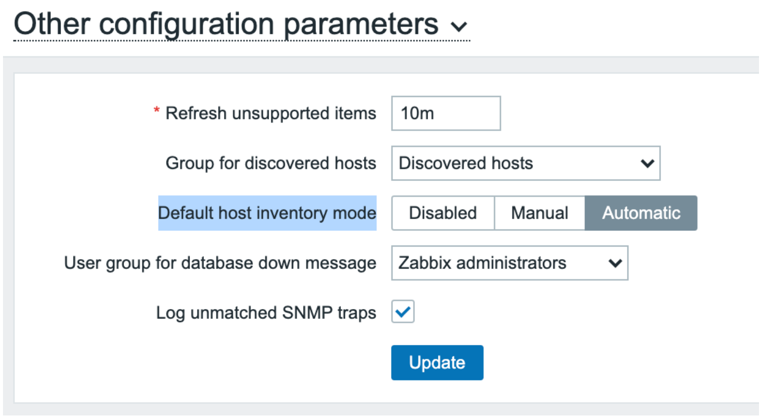
The feature makes it possible for us to automatically put collected data in a visual inventory in the Zabbix frontend. Let's get started.

Getting ready

Make sure to log in to the Zabbix frontend, and keep your SNMP-monitored host from the previous recipes ready.

How to do it...

1. Let's start by making sure our Zabbix server put all of our hosts' inventory information into the fields. I like to do this by going to **Administration** | **General** and then selecting **Other** from the dropdown in the top-left corner.
2. We can then set our **Default host inventory mode** parameter to **Automatic**. Don't forget to press **Update**:



Other configuration parameters

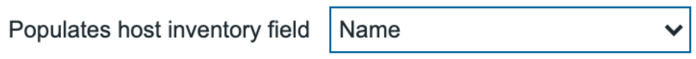
* Refresh unsupported items	10m
Group for discovered hosts	Discovered hosts
Default host inventory mode	Disabled / Manual / [✓ Automatic]
User group for database down message	Zabbix administrators
Log unmatched SNMP traps	✓

Figure 5.33 – Administration | General | Other configuration parameters page

3. Alternatively, we can do this at a host level. Go to **Configuration** | **Hosts** and select our `lar-book-templated_snmp` SNMP-monitored host.
4. Select **Inventory** and set it to **Automatic** here as well. As you may have noticed, the default applies only to newly created hosts from now on.

> **Important note**
> Changing the global setting does not apply it to all existing hosts but only to newly created hosts. It might be a good idea to run a **Mass update** before, or change the inventory mode manually, host by host.

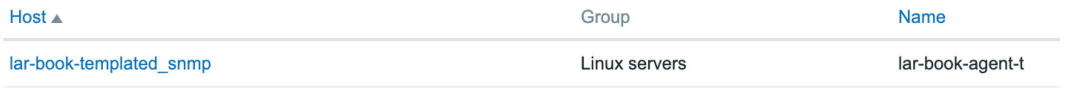
5. Clicking **Update** takes us back to the **Configuration** | **Hosts** page. Go to **Configuration** | **Templates** instead and select **Template OS Linux by SNMPvX**.
6. Go to **Items** and edit **System hostname**. We have to change the **Populates host inventory field** setting, like this:



Populates host inventory field: Name

Figure 5.34 – Edit item page

7. Click **Update** and navigate to **Inventory** | **Hosts**. You will see the following:



Host ▲	Group	Name
lar-book-templated_snmp	Linux servers	lar-book-agent-t

Figure 5.35 – Inventory | Hosts page

How it works...

Zabbix inventory is simple and underdeveloped at the moment. It's not amazing to filter to a point where it shows exactly what we want to see, but it can be useful nonetheless.

If you are working with a lot of equipment, such as in a **managed service provider** (**MSP**) environment, it can become overwhelming to log in to every device and get the serial number by hand. If you poll the serial number and you populate the **inventory** field, suddenly you have an active list of up-to-date serial numbers.

Use Zabbix inventory wisely when creating items, and put the population to **Automatic**, like we did in this chapter—you'll then never even have to think too much about the feature. You configure it almost automatically this way, and have nice lists waiting for you when you are in need of them.

Working through Zabbix reporting

Zabbix reporting is another feature that could use some love, especially on the part of actually getting reports out of the system. But it's a powerful feature to show you exactly what's going on with your statistics—something your customers might love, for instance.

Getting ready

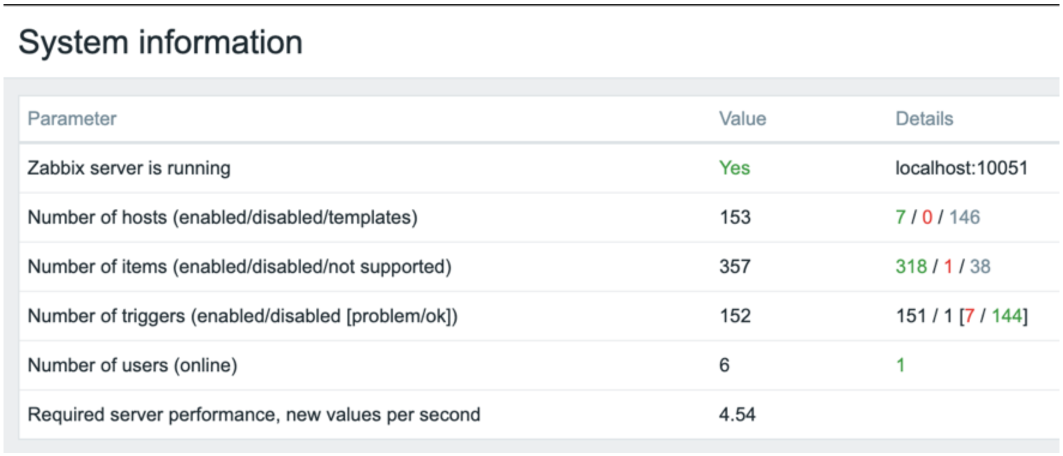
For this recipe, all you'll need is the Zabbix frontend and the SNMP-monitored host from the previous recipes.

How to do it…

There isn't anything to configure really, as reporting is present in Zabbix from the start. So, let's dive into what each page of reporting offers us.

System information

If you navigate to **Reports** | **System information**, you will find the following table:



System information

Parameter	Value	Details
Zabbix server is running	Yes	localhost:10051
Number of hosts (enabled/disabled/templates)	153	7 / 0 / 146
Number of items (enabled/disabled/not supported)	357	318 / 1 / 38
Number of triggers (enabled/disabled [problem/ok])	152	151 / 1 [7 / 144]
Number of users (online)	6	1
Required server performance, new values per second	4.54	

Figure 5.36 – Reports | System information page

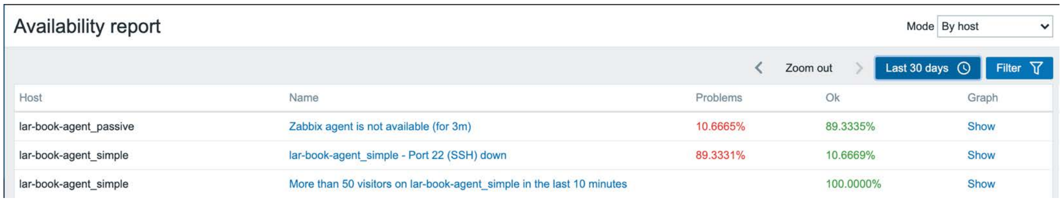
You might have seen this table before, as it is also configurable as a dashboard widget. This page gives us all of the quick information we need about our Zabbix server, such as the following:

- **Zabbix server is running**: Informs us whether the Zabbix server backend is actually running and where it is running. In this case it's running, and it's running on `localhost:10051`.
- **Number of hosts**: This will detail to us the number of hosts **enabled** (`7`), the number of hosts **disabled** (`0`), and the number of **templates** we have (`146`). It gives us a quick overview of our Zabbix server host information.
- **Number of items**: Here, we can see details of our Zabbix server's items—in this case, **enabled** (`318`), **disabled** (`1`), and **not supported** (`38`).
- **Number of triggers**: This details the number of triggers to us. We can see how many are **enabled** (`151`) and **disabled** (`1`), but also how many are in a **problem** state (`7`) and how many are in an **ok** state (`144`).

- **Number of users (online)**: The first value details the total number of users. The second value details the numbers of users currently logged in to the Zabbix frontend.
- **Required server performance, new values per second**: Perhaps I'm introducing you to a completely new concept here, which is **New Values Per Second**, or **NVPS**. A Zabbix server receives or requests values through items and writes this to our MariaDB database (or another database). The NVPS information detailed here shows the average number of NVPS received by the Zabbix server. Keep a close eye on this as your Zabbix server grows, as it's a good indicator to see how quickly you should scale up.

Availability report

Navigating to **Reports** | **Availability report** will give us some useful information about how long a trigger has been in a **Problems** state versus an **OK** state for a certain time period:


Availability report

Mode: By host

Zoom out · Last 30 days · Filter

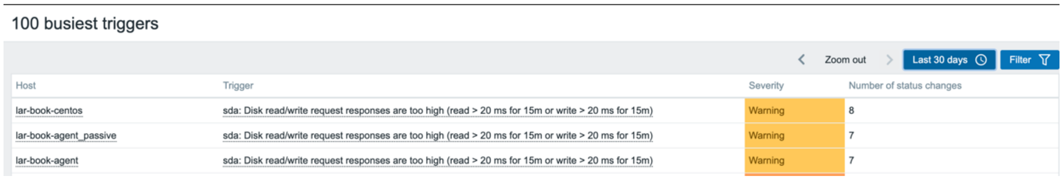
Host	Name	Problems	Ok	Graph
lar-book-agent_passive	Zabbix agent is not available (for 3m)	10.6665%	89.3335%	Show
lar-book-agent_simple	lar-book-agent_simple - Port 22 (SSH) down	89.3331%	10.6669%	Show
lar-book-agent_simple	More than 50 visitors on lar-book-agent_simple in the last 10 minutes		100.0000%	Show

Figure 5.37 – Reports | Availability report page

Looking at one of our hosts, we can see that in the last `30` days, the **Zabbix agent is not available (3m)** trigger has been in a **Problem** state for `10,66665%` of the time. This might be useful for us to know so that we can define how often a certain problem arises.

Trigger top 100

Navigating to **Reports** | **Trigger top 100**, we will find the top 100 triggers that have been firing in a certain amount of time:


100 busiest triggers

Zoom out · Last 30 days · Filter

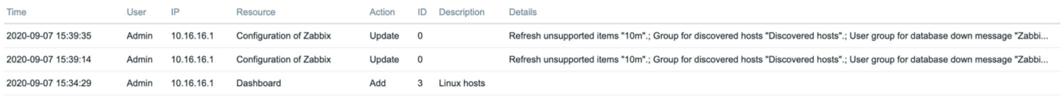
Host	Trigger	Severity	Number of status changes
lar-book-centos	sda: Disk read/write request responses are too high (read > 20 ms for 15m or write > 20 ms for 15m)	Warning	8
lar-book-agent_passive	sda: Disk read/write request responses are too high (read > 20 ms for 15m or write > 20 ms for 15m)	Warning	7
lar-book-agent	sda: Disk read/write request responses are too high (read > 20 ms for 15m or write > 20 ms for 15m)	Warning	7

Figure 5.38 – Reports | Trigger top 100 page

For my Zabbix server, the busiest trigger was a **Disk read/write** trigger on a server. A useful page to see what we are putting the most of our time into, problem-wise.

Audit

The audit log, which is a small addition to Zabbix, is found at **Reports | Audit**:



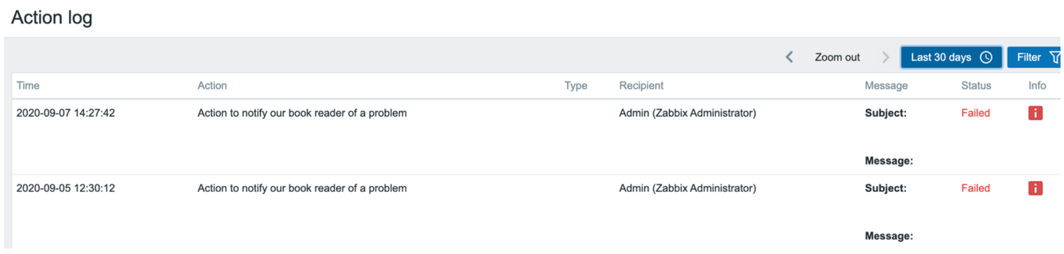
Time	User	IP	Resource	Action	ID	Description	Details
2020-09-07 15:39:35	Admin	10.16.16.1	Configuration of Zabbix	Update	0		Refresh unsupported items "10m".; Group for discovered hosts "Discovered hosts".; User group for database down message "Zabbi...
2020-09-07 15:39:14	Admin	10.16.16.1	Configuration of Zabbix	Update	0		Refresh unsupported items "10m".; Group for discovered hosts "Discovered hosts".; User group for database down message "Zabbi...
2020-09-07 15:34:29	Admin	10.16.16.1	Dashboard	Add	3	Linux hosts	

Figure 5.39 – Reports | Audit log page

We can see which user has done what in our Zabbix frontend—identifying a culprit for something that shouldn't have been done, for instance.

Action log

When we go to **Reports | Action log**, we land on a page that shows which actions have been fired. If you've configured **Actions**, then you can get a list here, like this:

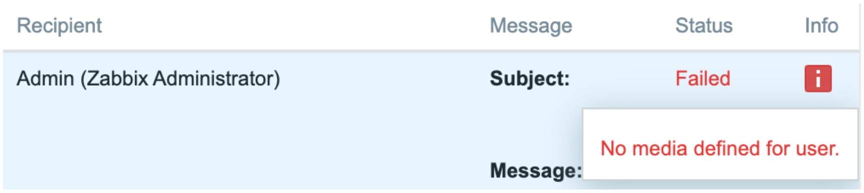


Time	Action	Type	Recipient	Message	Status	Info
2020-09-07 14:27:42	Action to notify our book reader of a problem		Admin (Zabbix Administrator)	**Subject:** **Message:**	Failed	🟥
2020-09-05 12:30:12	Action to notify our book reader of a problem		Admin (Zabbix Administrator)	**Subject:** **Message:**	Failed	🟥

Figure 5.40 – Reports | Action log page

If you're not sure if your action succeeded, then look at this list. It is very useful to troubleshoot your actions to a point where you get them up and running as you want.

When you hover over the **info** box, you also get to see what went wrong. For example, for the **Failed** items on my Zabbix instance, I should define the appropriate media type for the **Admin** user:



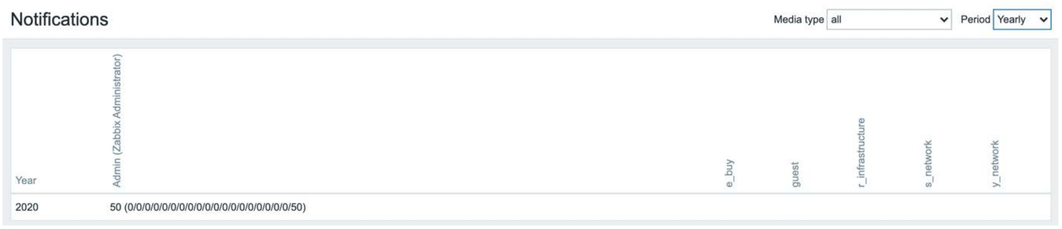
Recipient	Message	Status	Info
Admin (Zabbix Administrator)	**Subject:** **Message:**	Failed	i

No media defined for user.

Figure 5.41 – Reports | Action log info box

Notifications

Last, but not least, navigating to **Reports** | **Notifications** will show us the amount of notifications sent to a certain user for a period of time:



Notifications

Media type: all Period: Yearly

Year	Admin (Zabbix Administrator)	e_buy	guest	r_infrastructure	s_network	y_network
2020	50 (0/0/0/0/0/0/0/0/0/0/0/0/0/0/0/0/0/0/0/50)					

Figure 5.42 – Reports | Notifications page

In my case, `50` notifications have been sent to the **Admin** user this year, and `0` to other users.

6
Using Discovery for Automatic Creation

This chapter is going to be all about making sure that you as a Zabbix Administrator are doing as little work as possible on the host creation part. We are going to learn how to perform (or perfect, maybe) automatic host creation. Check out the recipes featured here to see just what we are going to discover.

In this chapter, we will first learn how to set up Zabbix discovery with Zabbix agent and **Simple Network Management Protocol** (**SNMP**). We will then set up active agent autoregistration. Later, we will also cover Windows performance counters and **Java Managements Extension** (**JMX**) object discovery.

In this chapter, we will cover the following recipes:

- Setting up Zabbix Agent discovery
- Setting up Zabbix SNMP discovery
- Working with Active agent autoregistration
- Using the new Windows performance counter discovery
- Discovering JMX objects

Technical requirements

As this chapter is all about host discovery, besides our Zabbix server we will need one new Linux host and a Windows host. Both these hosts will need Zabbix agent 2 installed, but not configured just yet.

Furthermore, we are going to need our JMX host, as configured in *Chapter 2*, *Setting up Zabbix Monitoring*, and a new host with SNMP set up. To learn more about setting up an SNMP-monitored host, check out the *Working with SNMP monitoring* recipe in *Chapter 2*, *Setting up Zabbix Monitoring*.

Setting up Zabbix Agent discovery

A lot of Zabbix administrators use Zabbix Agent extensively and thus spend a lot of time creating Zabbix Agent hosts by hand. They do this simply because they have no knowledge of Zabbix Agent discovery or have had no time to set it up yet. No longer will this be the case—in this recipe, we will learn just how easy it is to set up Zabbix Agent discovery, and neither knowledge nor time will be an issue anymore.

Getting ready

Besides our Zabbix server, I mentioned in this chapter's introduction that we will need two clean hosts with Zabbix Agent installed: one Windows host and one Linux host. If you don't know how to install Zabbix Agent, check out *Chapter 2*, *Setting up Zabbix Monitoring*, or see the Zabbix documentation at `https://www.zabbix.com/documentation/current/manual/concepts/agent2`.

Don't forget to install **Zabbix Agent** on both servers! Let's give the servers the following hostnames:

- `lar-book-disc-lnx`: For the Linux host (use Zabbix Agent 2)
- `lar-book-disc-win`: For the Windows host (Use Zabbix Agent 5.x)

How to do it…

1. Let's get started by logging in to our `lar-book-disc-nx` Linux host and editing the following file:

```
vim /etc/zabbix/zabbix_agent2.conf
```

2. Now, make sure your zabbix agent configuration file contains at least the following line:

```
Hostname=lar-book-disc-lnx
```

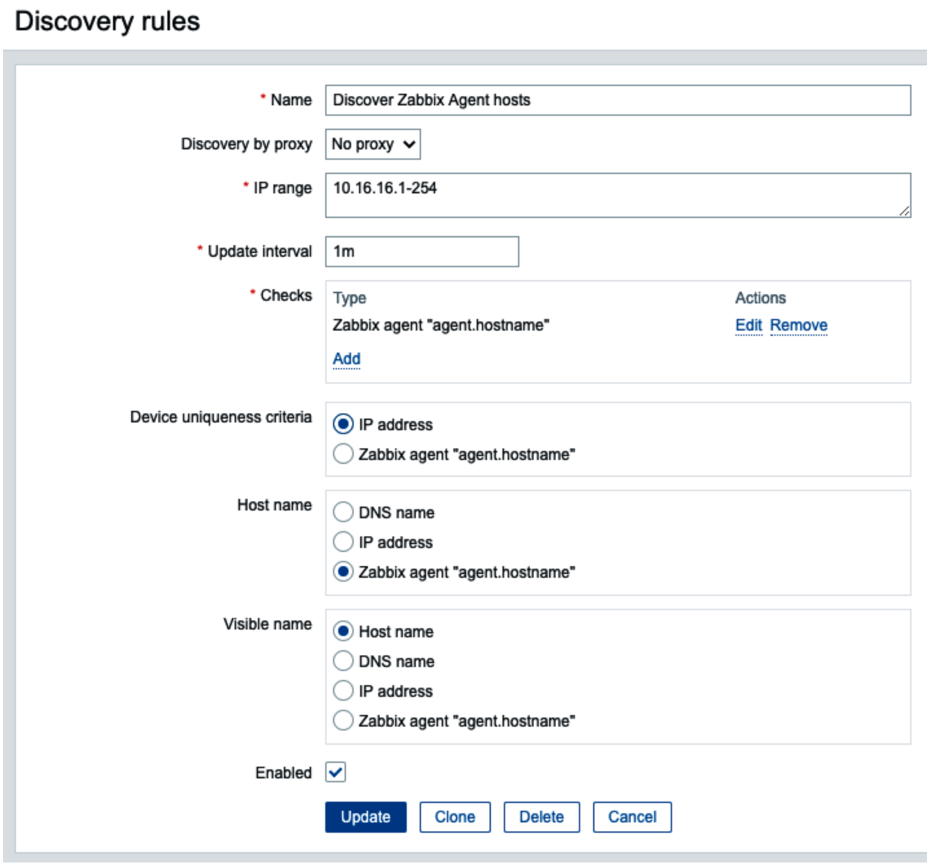
3. For your Windows Zabbix agent, it's important to do the same. Edit the following file:

```
C:\Program Files\Zabbix Agent\zabbix_agentd.conf
```

4. Now, change the hostname by editing the following line:

```
Hostname=lar-book-disc-win
```

5. Next up, in our Zabbix frontend, navigate to **Configuration** | **Discovery**, and on this page, we are going to create a rule with the following settings:



Discovery rules

* Name	Discover Zabbix Agent hosts
Discovery by proxy	No proxy
* IP range	10.16.16.1-254
* Update interval	1m
* Checks	Type: Zabbix agent "agent.hostname" — Actions: Edit Remove; Add
Device uniqueness criteria	[✓ IP address] / Zabbix agent "agent.hostname"
Host name	DNS name / IP address / [✓ Zabbix agent "agent.hostname"]
Visible name	[✓ Host name] / DNS name / IP address / Zabbix agent "agent.hostname"
Enabled	✓

Figure 6.1 – Discovery rules page for Zabbix agent hosts

6. Click the blue **Add** button to move on.
7. After setting up the discovery rule, we will also need to set up an action to actually create the host with the right template. Navigate to **Configuration** | **Actions** and use the dropdown in the top-left corner to configure **Discovery actions**:

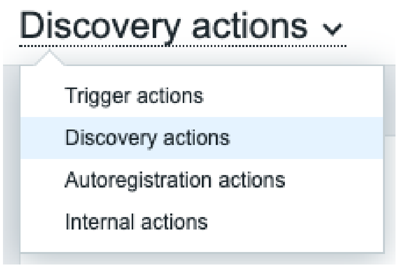
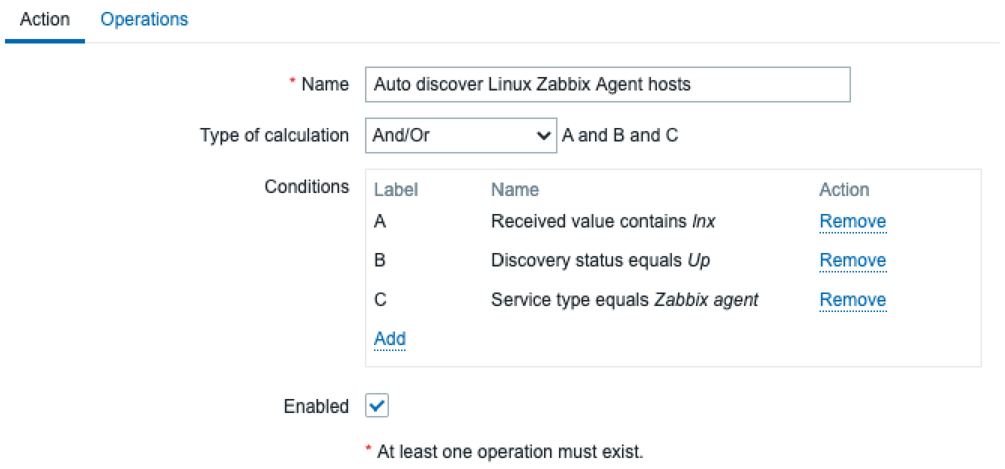


Figure 6.2 – Dropdown at Configuration | Actions

8. Here, we will click the **Create action** button in the top-right corner and fill out the next page with the following information:



Action | Operations

* Name	Auto discover Linux Zabbix Agent hosts
Type of calculation	And/Or — A and B and C
Conditions	
Enabled	✓

Label	Name	Action
A	Received value contains *lnx*	Remove
B	Discovery status equals *Up*	Remove
C	Service type equals *Zabbix agent*	Remove
Add		

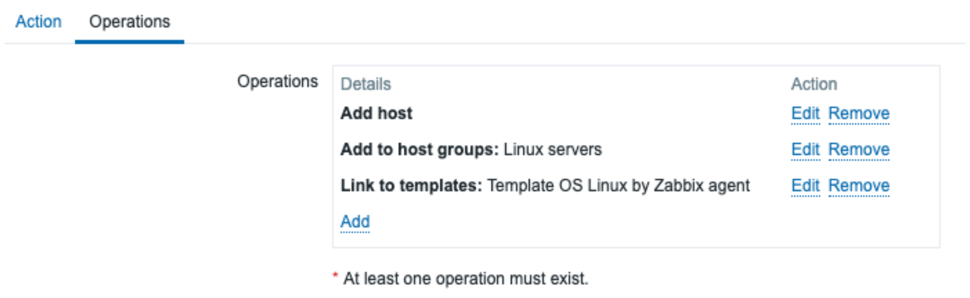
* At least one operation must exist.

Figure 6.3 – Create Discovery Action page for Zabbix agent hosts

Tip

When creating Zabbix actions, it's important to keep the order of creation in mind. The labels seen in the preceding screenshot will be added in order of creation. This means that it's easier to keep track of your Zabbix actions if you keep the order of creation the same for all actions.

9. Next up, click the **Operations** tab. This is where we will add the following:


Action Operations

Operations	Details	Action
	Add host	Edit Remove
	Add to host groups: Linux servers	Edit Remove
	Link to templates: Template OS Linux by Zabbix agent	Edit Remove
	Add	

* At least one operation must exist.

Figure 6.4 – Create Discovery action Operations page for Zabbix agent hosts

10. That's it for the Linux agent. Click the blue **Add** button, and let's continue to our Windows discovery rule.
11. Navigate to **Configuration** | **Host groups**. Create a host group for our Windows hosts by clicking **Create host group** in the top-right corner and filling out the group name:

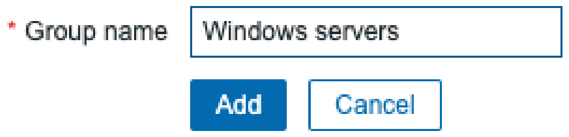
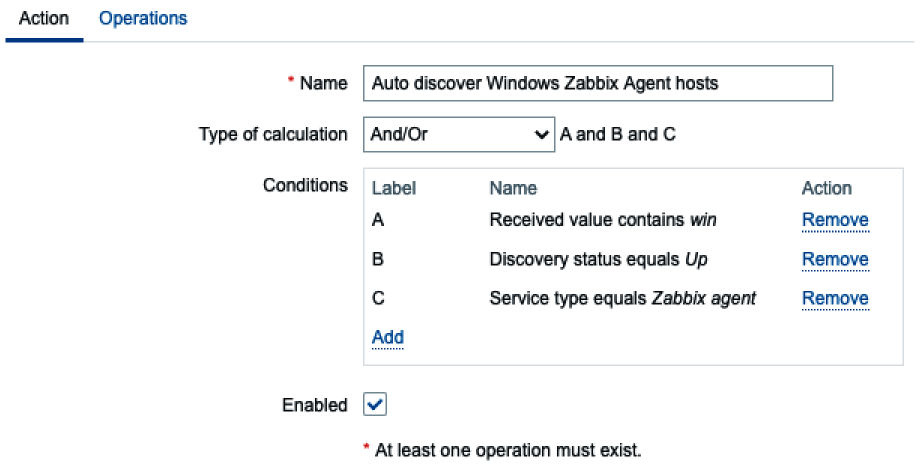

* Group name: Windows servers

Figure 6.5 – Create host group page for Windows server hosts

12. Click the blue **Add** button and navigate to **Configuration** | **Actions**.
13. Go to **Discovery actions** again and click **Create action**. We will fill out the same thing, but for our Windows hosts this time:


Action Operations

* Name: Auto discover Windows Zabbix Agent hosts

Type of calculation: And/Or A and B and C

Conditions:

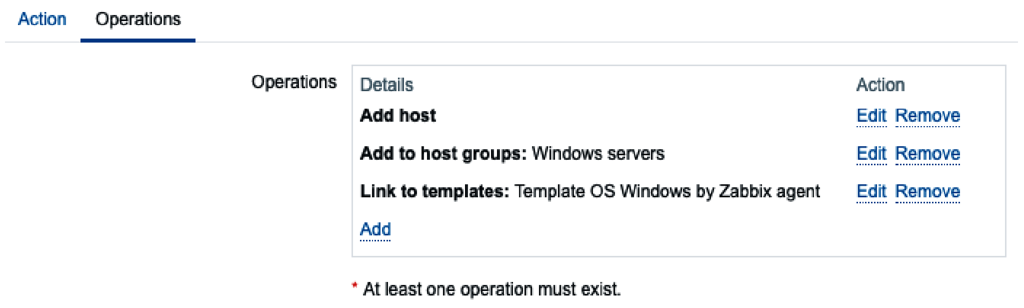
Label	Name	Action
A	Received value contains *win*	Remove
B	Discovery status equals *Up*	Remove
C	Service type equals *Zabbix agent*	Remove
Add		

Enabled: ✓

* At least one operation must exist.

Figure 6.6 – Create Discovery Action page for Windows Zabbix agents

14. Before pressing **Add**, let's also fill out the **Operations** page with the **Operations** shown in the image:

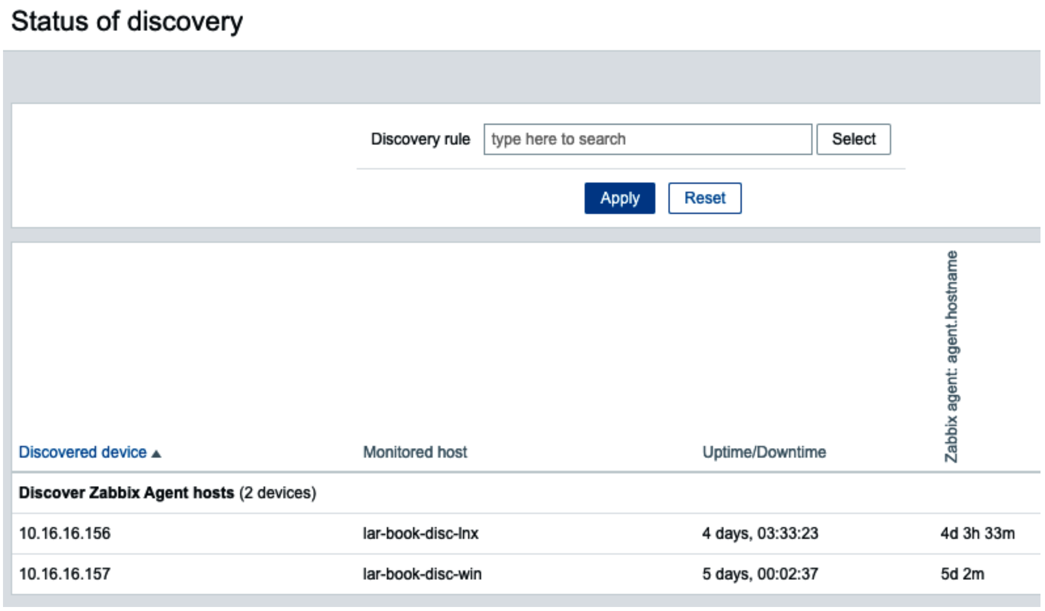


Operations	Details	Action
	Add host	Edit Remove
	Add to host groups: Windows servers	Edit Remove
	Link to templates: Template OS Windows by Zabbix agent	Edit Remove
	Add	

* At least one operation must exist.

Figure 6.7 – Create Discovery action Operations page for Windows Zabbix agents

15. Now, we can press the blue **Add** button, and our second discovery action is present.
16. Move on to **Monitoring | Discovery**. This is where we can see if and when our hosts are discovered:



Status of discovery

Discovered device ▲	Monitored host	Uptime/Downtime	Zabbix agent: agent.hostname
Discover Zabbix Agent hosts (2 devices)			
10.16.16.156	lar-book-disc-lnx	4 days, 03:33:23	4d 3h 33m
10.16.16.157	lar-book-disc-win	5 days, 00:02:37	5d 2m

Figure 6.8 – Monitoring | Discovery page

Tip

Use the **Monitoring | Discovery** page to keep a close eye on the hosts you expect to show up in your Zabbix setup. It's very useful to track new hosts coming in and see which zabbix discovery rule was used to create the host.

How it works...

Network discovery might not be very hard to set up initially, but there are loads of options to configure. For this example, we made the choice to use the `agent.hostname` key as our check. We create the Zabbix hostname based on what's configured in the Zabbix Agent config file.

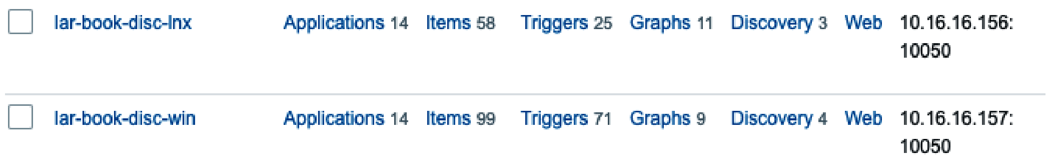
What happens is that Zabbix network discovery finds our hosts and performs our check. In this case, the check is: *What is the agent hostname in the config file?* This information, plus our **Internet Protocol** (**IP**), is then triggering the action. Our action then performs our configured checks:

- Does the hostname contain `lnx` or `win`?
- Is the discovery status `UP`?
- Is the service type `Zabbix Agent`?

If all of those checks are true, our action will then create our new discovered host with the following:

- Our configured host group plus the default `Discovered hosts` host group
- Our template as configured in our action

We will end up with two newly created hosts, with all the right settings:



☐	lar-book-disc-lnx	Applications 14	Items 58	Triggers 25	Graphs 11	Discovery 3	Web	10.16.16.156: 10050
☐	lar-book-disc-win	Applications 14	Items 99	Triggers 71	Graphs 9	Discovery 4	Web	10.16.16.157: 10050

Figure 6.9 – Configuration | Hosts page with our new hosts windows and linux

There's more...

Creating the host by using the configuration file settings isn't always the right way to go, but it's a solid start to working with network discovery.

If you want a more flexible environment where you don't have to even touch the Zabbix Agent configuration file, then you might want to use different checks on the discovery rule. Check out which keys we can use to build different discovery rules in the Zabbix documentation at `https://www.zabbix.com/documentation/current/manual/config/items/itemtypes/zabbix_agent`.

Setting up Zabbix SNMP discovery

If you work with a lot of SNMP devices but don't always want to set up monitoring manually, network discovery is the way to go. Zabbix network discovery uses the same functionality as Zabbix Agent discovery, but with a different configuration approach.

Getting ready

To get started with network discovery, we are going to need a host that we can monitor with SNMP. If you don't know how to set up a host such as this, check out the *Working with SNMP monitoring* recipe in *Chapter 2*, *Setting up Zabbix Monitoring*. We'll also need our Zabbix server.

How to do it...

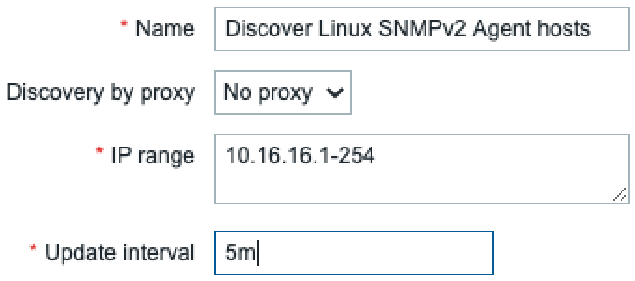
1. First, log in to your new SNMP-monitored host and change the hostname to the following:

```
hostnamectl set-hostname lar-book-disc-snmp
```

2. Then, reboot using the following command:

```
shutdown -r now
```

3. Now, navigate to **Configuration** | **Discovery** and click on **Create discovery rule** in the top-right corner.
4. We are going to create a new SNMP discovery rule, with an SNMP **Object Identifier** (**OID**) check. Fill out the **Name** and **IP range** fields first, like this:

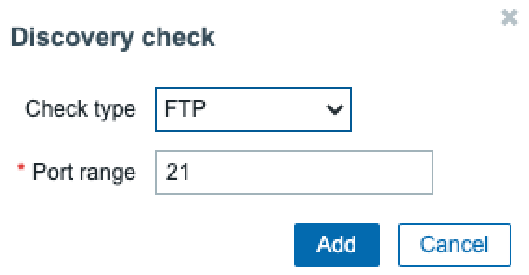


* Name	Discover Linux SNMPv2 Agent hosts
Discovery by proxy	No proxy
* IP range	10.16.16.1-254
* Update interval	5m

Figure 6.10 – Configuration | Discovery, discovery rule creation page for SNMPv2 agents

5. Make sure to fill out your own IP range in the **IP range** field.

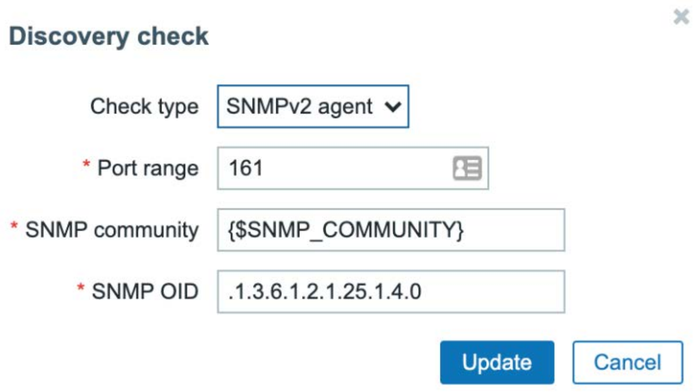
6. Now, we are going to create our SNMP check. Click on **Add** next to **Checks**, and you'll see the following pop-up screen:


Discovery check

Check type	FTP
* Port range	21

Figure 6.11 – Configuration | Discovery, discovery check creation pop-up window

7. We want our **Check type** to be `SNMPv2 agent` and we want to fill it with our community and a useful OID. Fill it out, like this:


Discovery check

Check type	SNMPv2 agent
* Port range	161
* SNMP community	{$SNMP_COMMUNITY}
* SNMP OID	.1.3.6.1.2.1.25.1.4.0

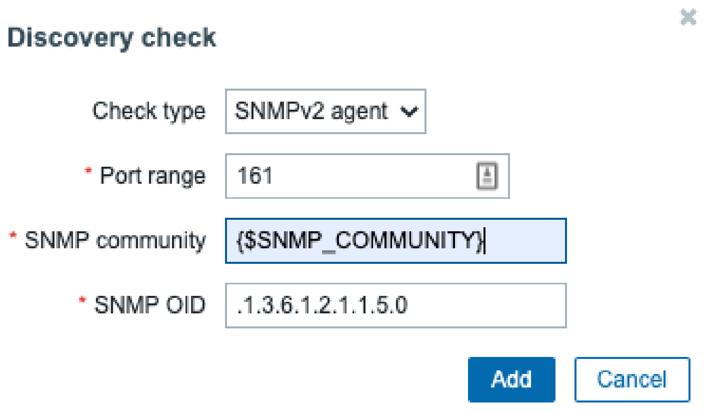
Figure 6.12 – Configuration | Discovery, discovery check creation pop-up window filled for SNMPv2 agents

Important note

Please note that our check type is **NOT** SNMP version-independent. We have three SNMP versions and thus three different check types to choose from, unlike with our new SNMP interface selection on the Zabbix 5 host screen.

8. Click **Add** and move on.

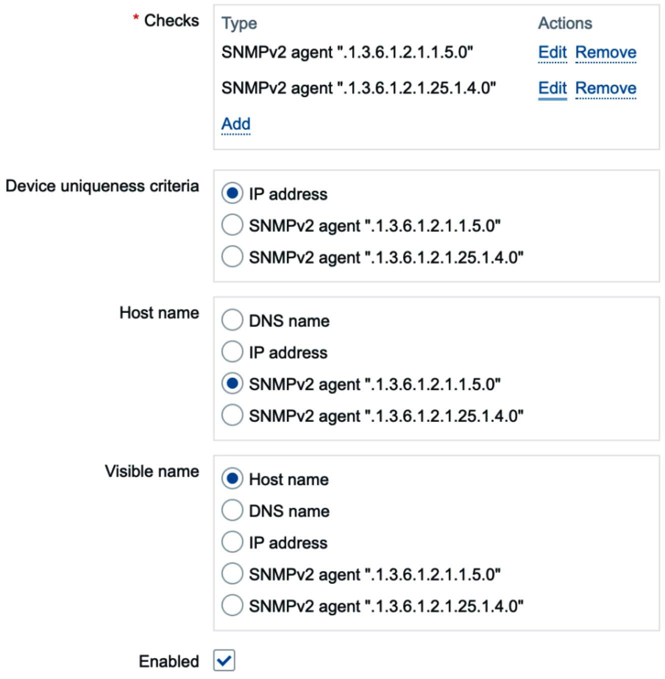
9. We will create another `SNMPv2 agent` check like this, but with the following settings:



Check type	SNMPv2 agent
* Port range	161
* SNMP community	{$SNMP_COMMUNITY}
* SNMP OID	.1.3.6.1.2.1.1.5.0

Figure 6.13 – Configuration | Discovery, discovery check creation pop-up window filled for SNMPv2 agents

10. After clicking **Add** again, fill out the rest of the page, as follows:



* Checks

Type	Actions
SNMPv2 agent ".1.3.6.1.2.1.1.5.0"	Edit Remove
SNMPv2 agent ".1.3.6.1.2.1.25.1.4.0"	Edit Remove
Add	

Device uniqueness criteria: [✓ IP address] SNMPv2 agent ".1.3.6.1.2.1.1.5.0" SNMPv2 agent ".1.3.6.1.2.1.25.1.4.0"

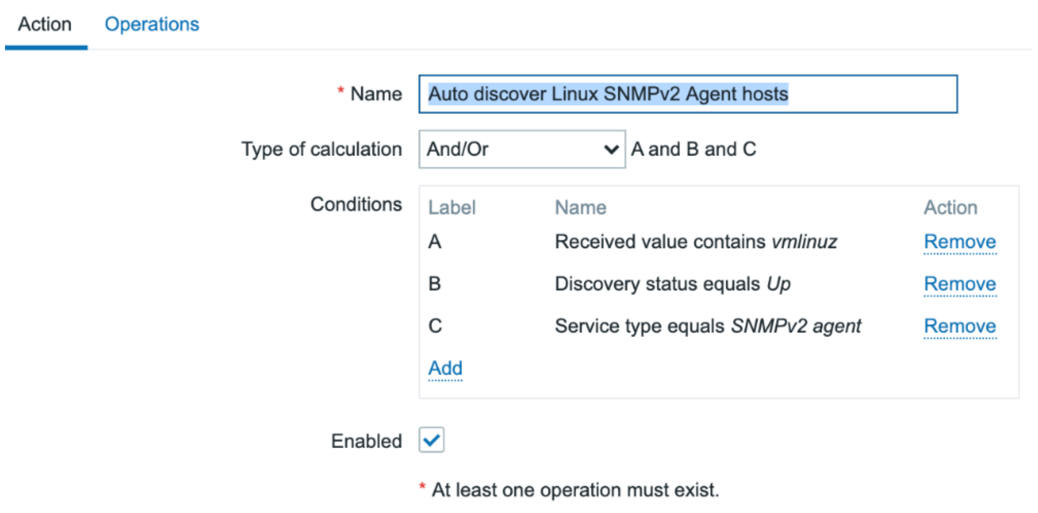
Host name: DNS name IP address [✓ SNMPv2 agent ".1.3.6.1.2.1.1.5.0"] SNMPv2 agent ".1.3.6.1.2.1.25.1.4.0"

Visible name: [✓ Host name] DNS name IP address SNMPv2 agent ".1.3.6.1.2.1.1.5.0" SNMPv2 agent ".1.3.6.1.2.1.25.1.4.0"

Enabled ✓

Figure 6.14 – Configuration | Discovery page for SNMPv2 agents

11. Last, but not least, click the **Add** button at the bottom of the page. This concludes creating our Zabbix Discovery rule.
12. We will also need an action for creating our hosts from the discovery rule. Navigate to **Configuration** | **Actions**, and after using the dropdown to select **Discovery actions**, click on **Create action**.
13. We will fill out the page with the following information:



* Name: Auto discover Linux SNMPv2 Agent hosts

Type of calculation: And/Or A and B and C

Conditions

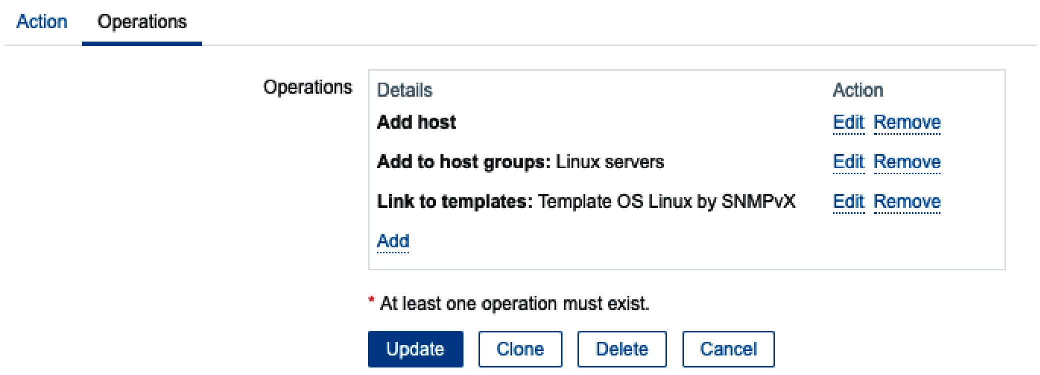
Label	Name	Action
A	Received value contains *vmlinuz*	Remove
B	Discovery status equals *Up*	Remove
C	Service type equals *SNMPv2 agent*	Remove

Enabled: ✓

* At least one operation must exist.

Figure 6.15 – Configuration | Actions, action creation page for SNMPv2 agents

14. Before clicking **Add**, navigate to **Operations** and fill out this page with the following details:



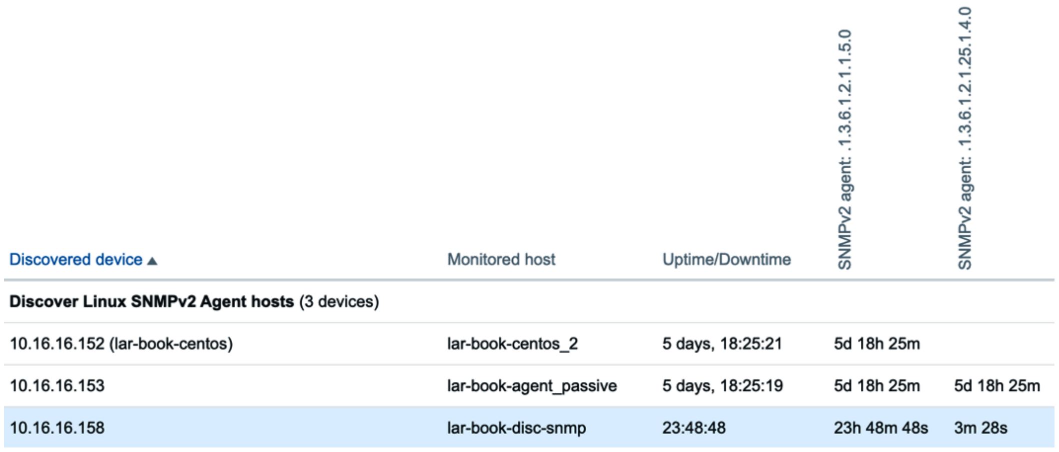
Operations

Details	Action
Add host	Edit Remove
Add to host groups: Linux servers	Edit Remove
Link to templates: Template OS Linux by SNMPvX	Edit Remove

* At least one operation must exist.

Figure 6.16 – Configuration | Actions, action creation Operations page for SNMPv2 agents

15. Now, click **Add** and navigate to **Monitoring | Discovery** to see if our host gets created:



Discovered device ▲	Monitored host	Uptime/Downtime	SNMPv2 agent: .1.3.6.1.2.1.1.5.0	SNMPv2 agent: .1.3.6.1.2.1.25.1.4.0
Discover Linux SNMPv2 Agent hosts (3 devices)				
10.16.16.152 (lar-book-centos)	lar-book-centos_2	5 days, 18:25:21	5d 18h 25m	
10.16.16.153	lar-book-agent_passive	5 days, 18:25:19	5d 18h 25m	5d 18h 25m
10.16.16.158	lar-book-disc-snmp	23:48:48	23h 48m 48s	3m 28s

Figure 6.17 – Monitoring | Discovery page for SNMPv2 agents

How it works…

In this recipe, we've created another discovery rule, but this time for SNMP. As you've noticed, the principle remains the same, but the application is a bit different.

When we created this Zabbix Discovery rule, we gave it two checks instead of the one check we did in the previous recipe. We created one check on SNMP OID `.1.3.6.1.2.1.1.5.0` to retrieve the hostname of the device through SNMP. We then put the hostname retrieved from the system into Zabbix as the Zabbix hostname of the system.

We also created a check on SNMP OID `.1.3.6.1.2.1.25.1.4.0`. This check will retrieve the following string if present:

```
"BOOT_IMAGE=(hd0,gpt2)/vmlinuz-4.18.0-193.6.3.el8_2.x86_64
root=/dev/mapper/cl-root ro crashkernel=auto resume=/dev/
mapper/cl-swa"
```

If the string is present, it means that the boot image is Linux on this host. This is a perfect example of how we can retrieve multiple OIDs to do multiple checks in our Zabbix Discovery rules. If we'd been monitoring a networking device, for instance, we could have picked an OID to see if it was a Cisco or a Juniper device. We would replace `.1.3.6.1.2.1.25.1.4.0` with any OID and poll it. Then, we would create our action based on what we received (Juniper or Cisco) and add our templates accordingly.

Important note

General knowledge of SNMP structure is very important when creating Zabbix discovery rules. We want to make sure we use the right SNMP OIDs as checks. Make sure to do your research well, utilize SNMP walks, and make a plan to discover SNMP agents. This way, you'll end up with a solid monitoring infrastructure.

Working with Active agent autoregistration

Using discovery to set up your Zabbix agents is a very useful method to automate your host creation. But what if we want to be even more up front with our environment and automate further? That's when we use a Zabbix feature called Active agent autoregistration.

Getting ready

For this recipe, we will need a new Linux (CentOS 8) host. We will call this host `lar-book-lnx-agent-auto`. Make sure to install the Zabbix Agent 2 to this host. Besides this new host, we'll also need our Zabbix server.

How to do it...

1. Let's start by logging in to our new `lar-book-lnx-agent-auto` host and change the following file:

```
vim /etc/zabbix/zabbix_agent2.conf
```

2. We will then edit the following line in the file. Make sure to enter your Zabbix server IP on this line:

```
ServerActive=10.16.16.152
```

3. We can also change the following line in the file if we want to set our hostname in the file manually:

```
Hostname=lar-book—lnx-agent-auto
```

This is not a requirement though, as Zabbix Agent will send the system hostname to our Zabbix server to use.

4. Next up, we will navigate to our Zabbix frontend, where we'll go to **Configuration** | **Actions**.
5. Use the drop-down menu to go to **Autoregistration actions**, as in the following screenshot:

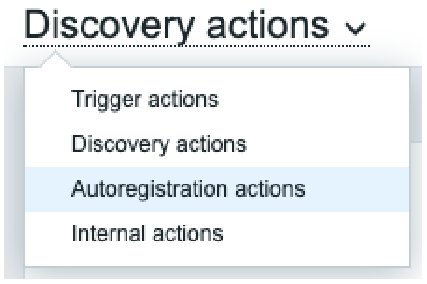
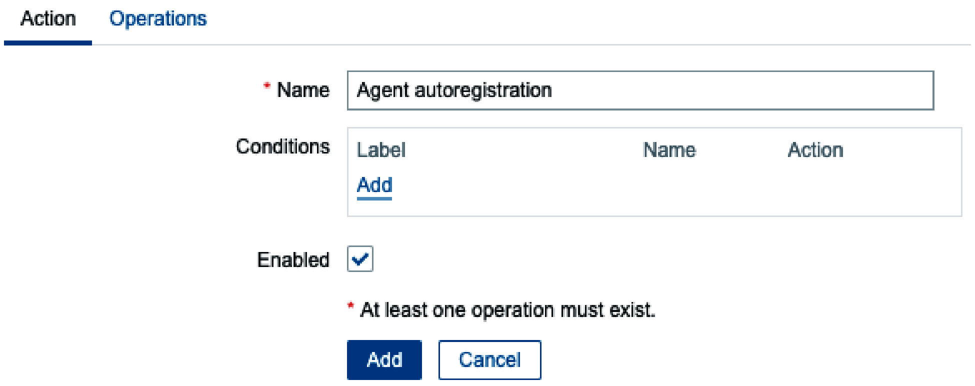


Figure 6.18 – Configuration | Actions page drop-down menu

6. Now, we will click the blue **Create action** button in the top-right corner to create a new action.
7. Fill out the **Name** and then click on the **Add** text link:



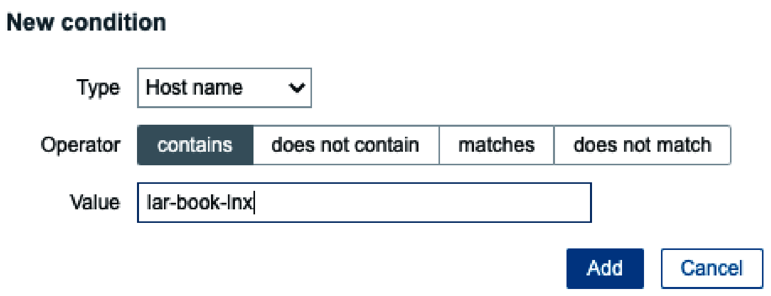
* Name: Agent autoregistration

Enabled: ✓

* At least one operation must exist.

Figure 6.19 – Configuration | Actions, create new action page

8. We can set up a condition here to only register hosts with a certain hostname. Let's do this by filling out the window like this:


New condition

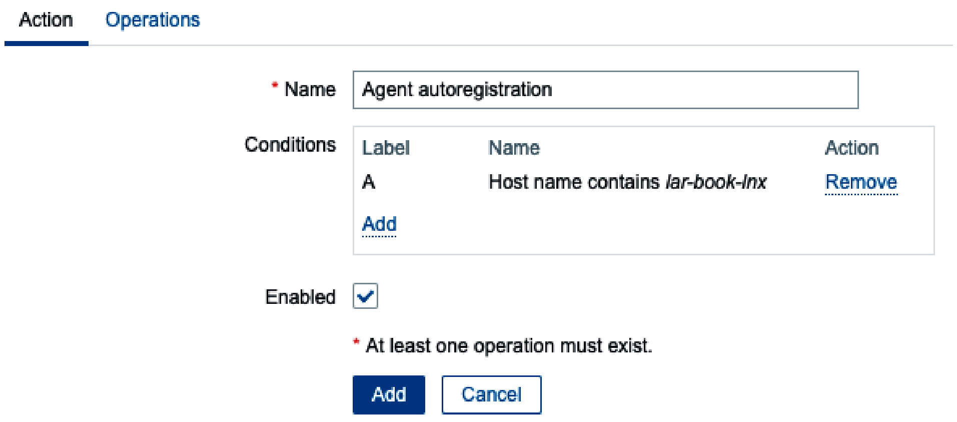
Type: Host name

Operator: [✓ contains] does not contain | matches | does not match

Value: lar-book-lnx

Figure 6.20 – Create action page, New condition window for host lar-book-lnx

Your page should now look like this:



* Name: Agent autoregistration

Conditions:

Label	Name	Action
A	Host name contains *lar-book-lnx*	Remove

Enabled: ✓

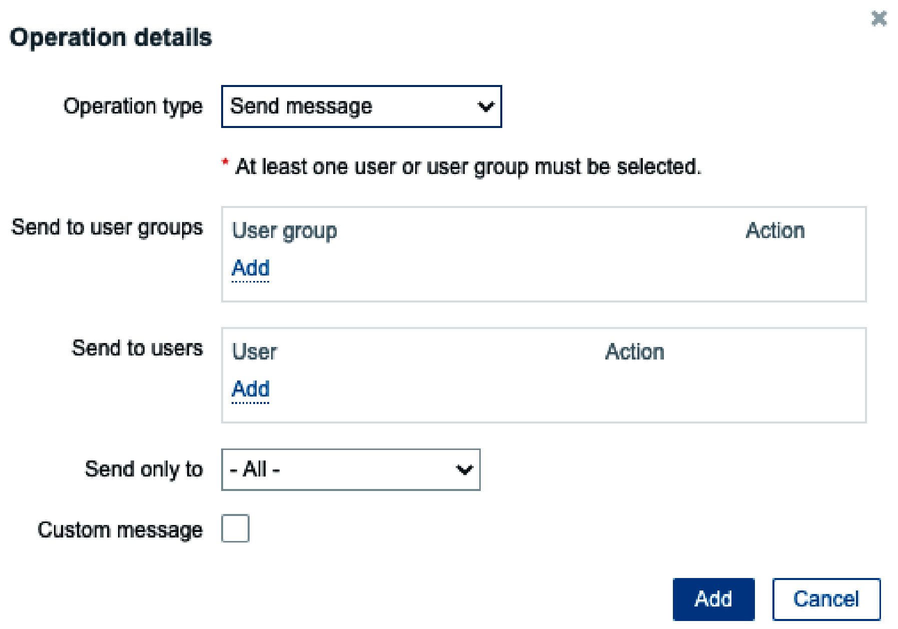
* At least one operation must exist.

Figure 6.21 – Create action page, filled with our information for host lar-book-lnx

Tip

We can set up conditions for different types of hosts. For instance, if we want to add Windows hosts, we set up a new action with a different hostname filter. This way, it is easy to maintain the right groups and templates, even with autoregistration.

9. Before clicking the blue **Add** button, let's go to the **Operations** tab.
10. Click on the **Add** text link, and you will see the following window:


Operation details

Operation type: Send message

* At least one user or user group must be selected.

Send to user groups: User group | Action — Add

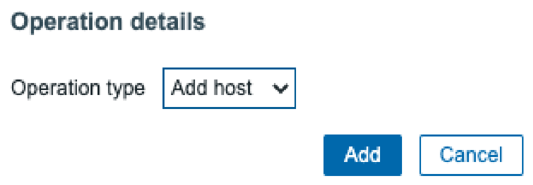
Send to users: User | Action — Add

Send only to: - All -

Custom message: ☐

Figure 6.22 – Create action operations page, new operation window, Send message for host lar-book-lnx

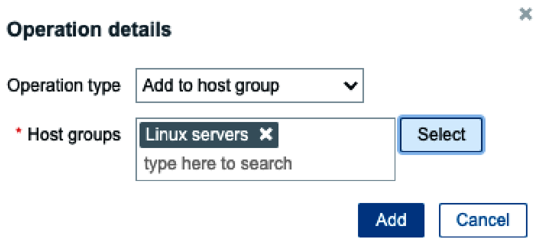
11. Create an action to add the host by changing the **Operation type** field to **Add host**:


Operation details

Operation type: Add host

Figure 6.23 – Create action operations page, new operation window, Add host, lar-book-lnx

12. Create an action to add the host to the **Linux servers** host group:


Operation details

Operation type: Add to host group

* Host groups: Linux servers

Figure 6.24 – Create action operations page, new operation window, Add to host group, lar-book-lnx

13. Create an action to add the host to the `Template OS Linux by Zabbix agent active` template:

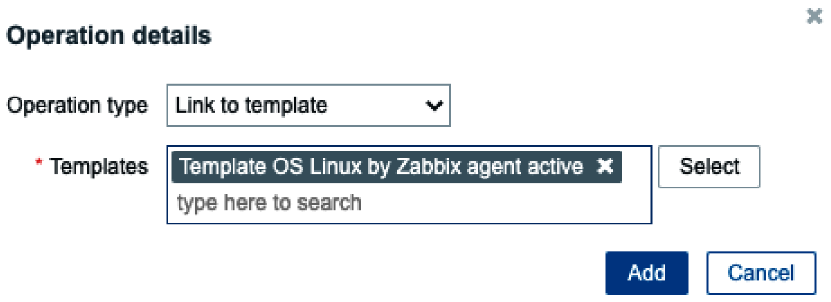
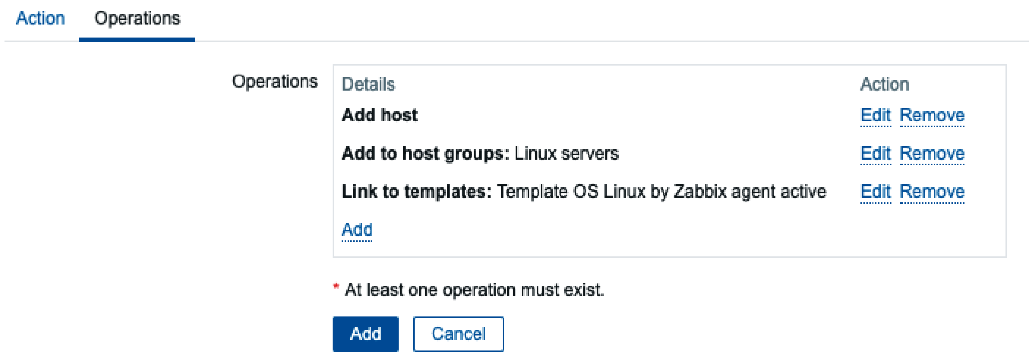


Figure 6.25 – Create action operations page, new operation window, Link to template, lar-book-lnx

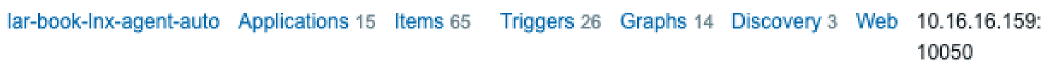
14. Your finalized **Operations** page should now look like this, and we can click the blue **Add** button:



Operations	Details	Action
	Add host	Edit Remove
	Add to host groups: Linux servers	Edit Remove
	Link to templates: Template OS Linux by Zabbix agent active	Edit Remove
	Add	

Figure 6.26 – Create action operations page, filled with our information, lar-book-lnx

15. Navigate to **Configuration** | **Hosts**, and we can see our new active autoregistered host:



lar-book-lnx-agent-auto	Applications 15	Items 65	Triggers 26	Graphs 14	Discovery 3	Web	10.16.16.159: 10050

Figure 6.27 – Configuration | Hosts page with host lar-book-lnx-agent-auto

How it works...

Active agent autoregistration is a solid method to let a host register itself. Once the `ServerActive=` line is set up with the Zabbix server IP, the host agent will start sending messages to the Zabbix server. Zabbix Server will receive these requests, and if there is an action set up in Zabbix (as we just did in this recipe), the host autoregisters to Zabbix:

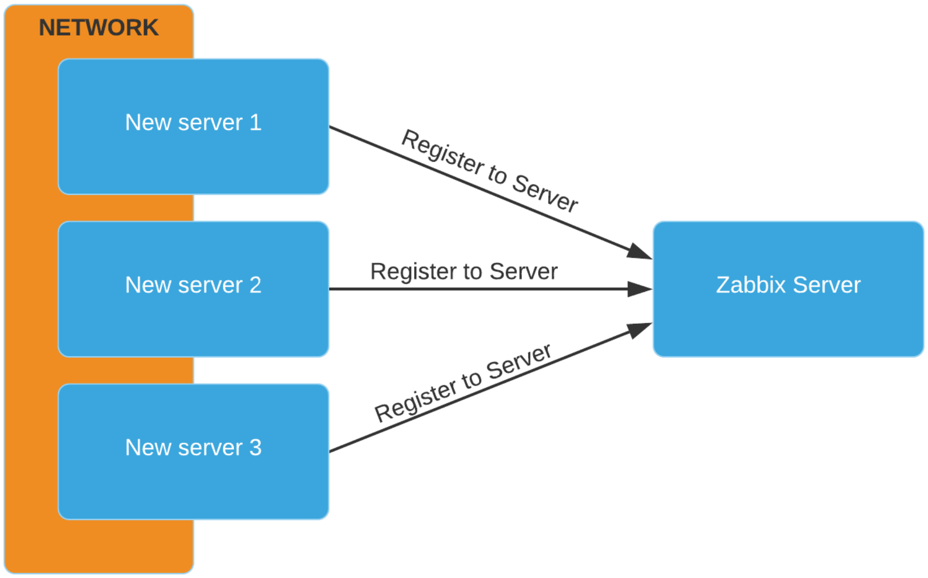


Figure 6.28 – Host autoregistration process

We can do a bunch of cool automations with this functionality. We could create a script to fill up our Zabbix Agent configuration file with the right `ServerActive=` line on our hosts in a certain IP pool.

It would also be super-easy to set up new hosts with Ansible. We make Zabbix Agent install itself with Ansible and we add the `ServerActive=` line in the `/etc/zabbix/zabbix_agent2.conf` file with Ansible. Our Zabbix Server action will take care of the rest from here.

Zabbix Agent autoregistration is a perfect way to get a zero-touch monitoring environment that's always up to date with our latest new hosts.

There's more...

Not every company uses hostnames that reflect the machine's OS or other attributes. This is when Zabbix `HostMetadata` can come in very useful. We can add this field to the active Zabbix Agent configuration to reflect attributes of the machine.

Afterward, we can use this `HostMetadata` in our Zabbix discovery action to do the same kind of filtering as we did on the hostname. Check out this link for more information:

`https://www.zabbix.com/documentation/current/manual/discovery/auto_registration#using_host_metadata`

Using the new Windows performance counter discovery

Since Zabbix 5.0.1, it is possible to discover Windows performance counters. In this recipe, we will go over the process of discovering Windows performance counters to use in our environments.

Discovering Windows performance counters is one of the new features in Zabbix 5. It might still be a little tricky as you probably haven't seen it before, but once we finish this recipe you'll know exactly how to set it up.

Getting ready

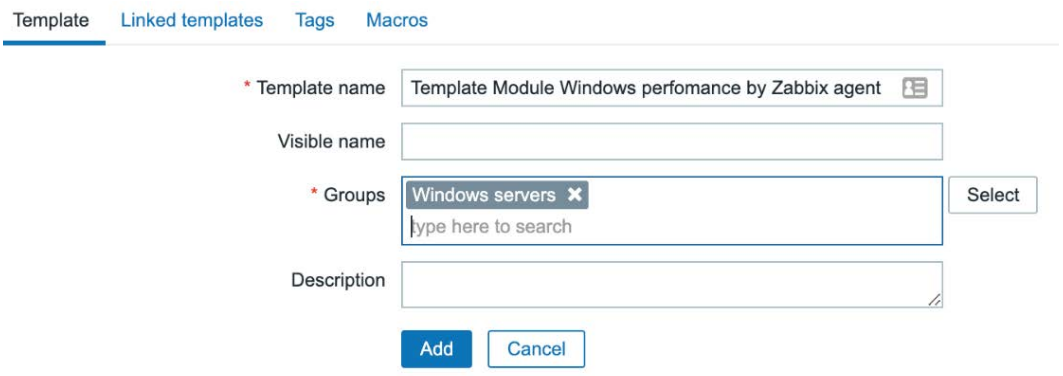
In this chapter, we added the `lar-book-disc-win` host to our setup, which is the host used in our Zabbix Agent discovery process. We can reuse this host for discovering Windows performance counters easily.

Of course, we'll also need our Zabbix server. Window performance counter discovery is supported starting from Zabbix 5.0.1, so if you are still on Zabbix 5.0.0 you will need to upgrade. To do this, use the following command:

```
yum update
```

How to do it…

1. Let's start by navigating to **Configuration** | **Templates** and creating a new template by clicking **Create template** in the top-right corner.
2. Create the following template:



* Template name	Template Module Windows perfomance by Zabbix agent
Visible name	
* Groups	Windows servers ✖ type here to search
Description	

Figure 6.29 – Configuration | Templates, Create template page, Windows performance

3. Click on the blue **Add** button, which will bring you back to **Configuration** | **Templates**. Select the new template.
4. Now, before continuing with our template, navigate to your Windows frontend and open `perfmon.exe`:

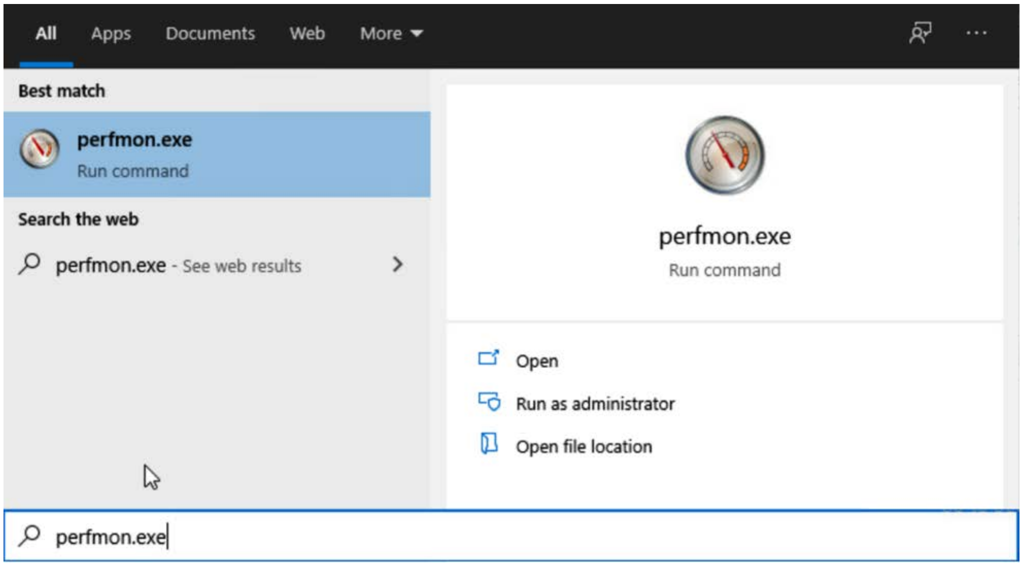
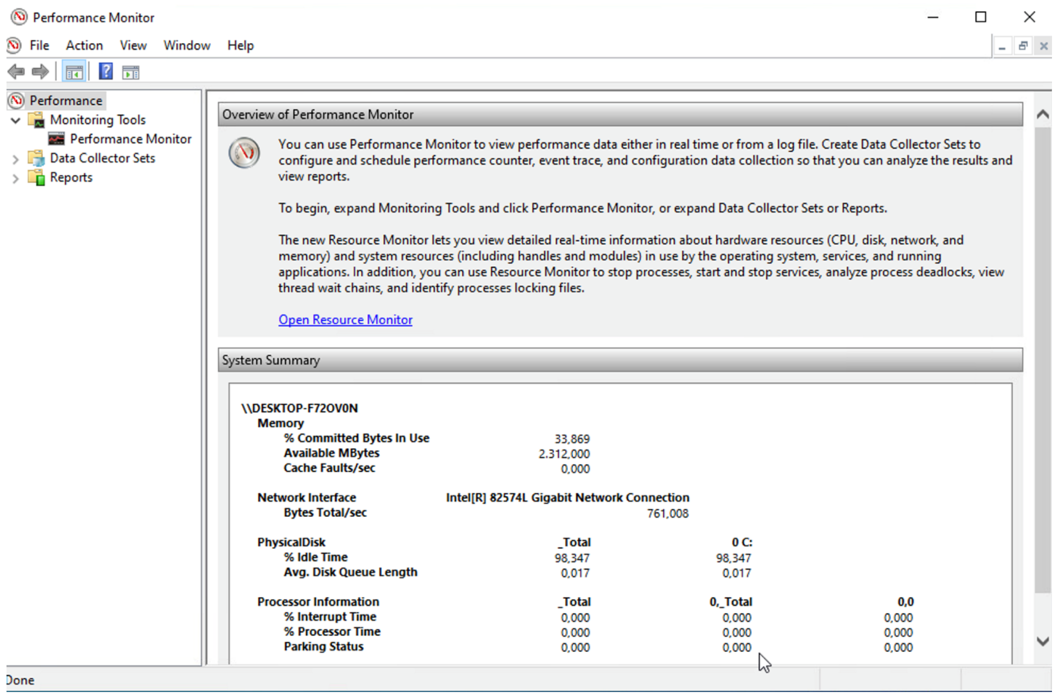


Figure 6.30 – Windows search bar, perfmon.exe

5. This will open the following window:



You can use Performance Monitor to view performance data either in real time or from a log file. Create Data Collector Sets to configure and schedule performance counter, event trace, and configuration data collection so that you can analyze the results and view reports.

To begin, expand Monitoring Tools and click Performance Monitor, or expand Data Collector Sets or Reports.

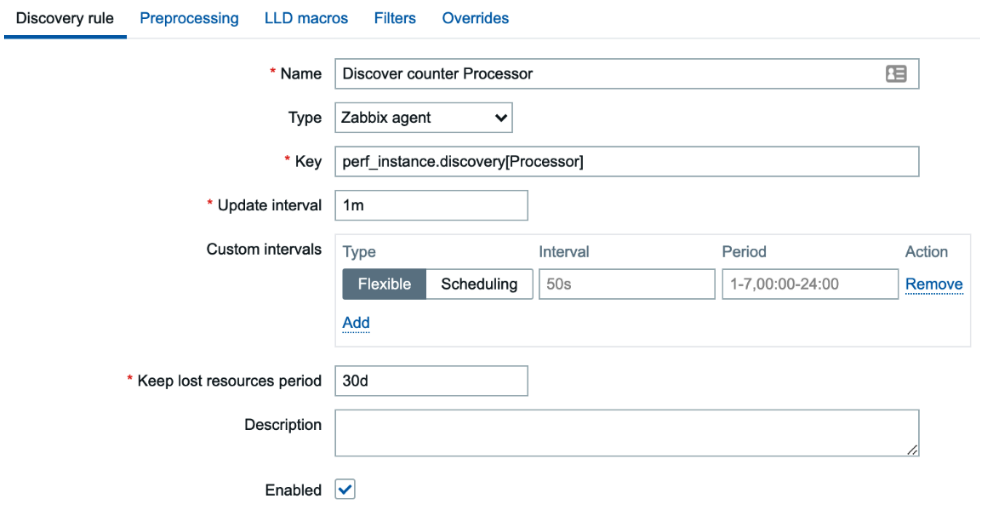
The new Resource Monitor lets you view detailed real-time information about hardware resources (CPU, disk, network, and memory) and system resources (including handles and modules) in use by the operating system, services, and running applications. In addition, you can use Resource Monitor to stop processes, start and stop services, analyze process deadlocks, view thread wait chains, and identify processes locking files.

```
\\DESKTOP-F72OV0N
  Memory
      % Committed Bytes In Use                33,869
      Available MBytes                     2.312,000
      Cache Faults/sec                         0,000

  Network Interface              Intel[R] 82574L Gigabit Network Connection
      Bytes Total/sec                                              761,008

  PhysicalDisk                               _Total                  0 C:
      % Idle Time                            98,347                98,347
      Avg. Disk Queue Length                  0,017                 0,017

  Processor Information                      _Total              0,_Total               0,0
      % Interrupt Time                        0,000                 0,000             0,000
      % Processor Time                        0,000                 0,000             0,000
      Parking Status                          0,000                 0,000             0,000
```

Figure 6.31 – Windows Perfmon.exe

6. Let's click on **Performance Monitor** and then on the green + icon. This will show you all the available Windows performance counters.
7. Let's start by using the **Processor** counter.
8. Go back to the **Configuration** | **Template** page in Zabbix and edit our new **Template Module Windows performance by Zabbix agent** template.
9. When you are at the **Edit template** page, click on **Discovery rules** in the bar next to your template name.

10. Click on **Create new discovery rule** in the top-right corner and add the following rule:

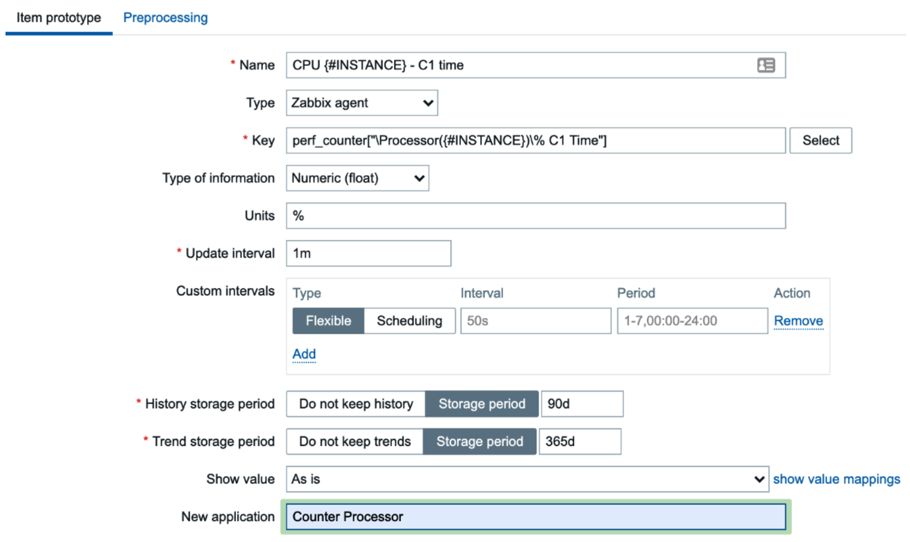


* Name	Discover counter Processor
Type	Zabbix agent
* Key	perf_instance.discovery[Processor]
* Update interval	1m
Custom intervals	Type: Flexible / Scheduling; Interval: 50s; Period: 1-7,00:00-24:00; Action: Remove; Add
* Keep lost resources period	30d
Description	
Enabled	✓

Figure 6.32 – Create Low-Level Discovery (LLD) rule page, Discover counter Processor

11. Click the blue **Add** button at the bottom and click our new `Discover counter Processor` discovery rule.

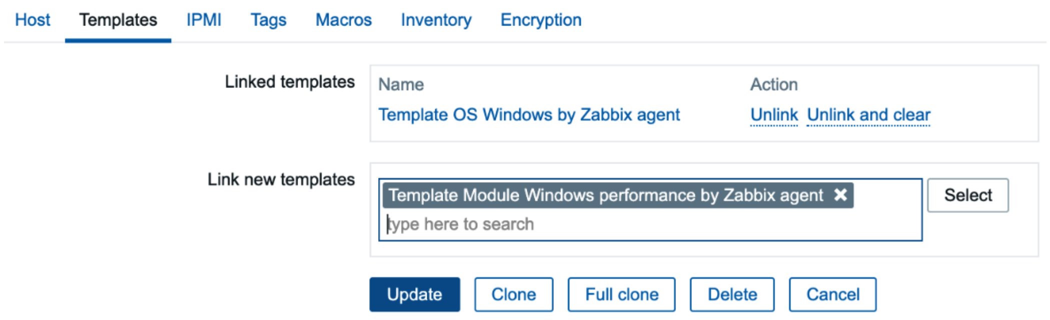
12. Click on **Item prototypes**, and in the top-right corner click on **Create item prototype**. We will then create the following item prototype:



* Name	CPU {#INSTANCE} - C1 time
Type	Zabbix agent
* Key	perf_counter["\Processor({#INSTANCE})\% C1 Time"]
Type of information	Numeric (float)
Units	%
* Update interval	1m
Custom intervals	Type: Flexible / Scheduling; Interval: 50s; Period: 1-7,00:00-24:00; Action: Remove; Add
* History storage period	Do not keep history / Storage period: 90d
* Trend storage period	Do not keep trends / Storage period: 365d
Show value	As is
New application	Counter Processor

Figure 6.33 – Create LLD item prototype page, CPU instance C1 time

13. Now, go to **Configuration** | **Hosts** and click on `lar-book-disc-win`.
14. Go to **Templates** and add our **Template Module Windows performance by Zabbix agent** template:



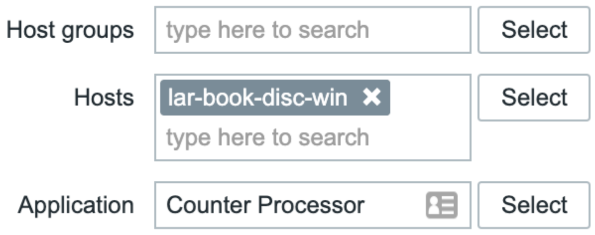
Linked templates

Name	Action
Template OS Windows by Zabbix agent	Unlink Unlink and clear

Link new templates: Template Module Windows performance by Zabbix agent ✖ type here to search

Figure 6.34 – Create LLD item prototype page, add Templates for lar-book-disc-win

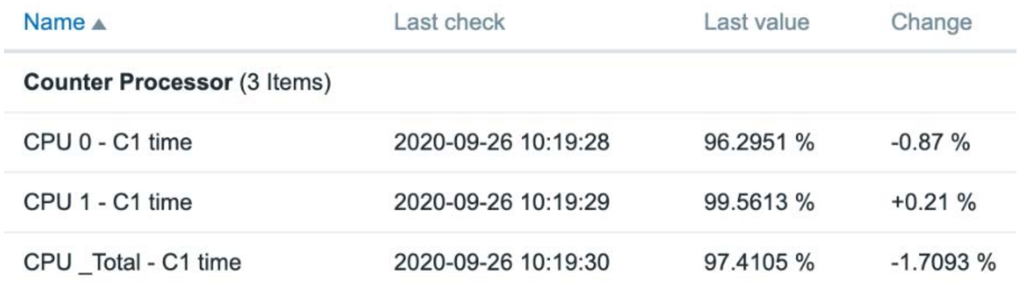
15. After clicking on the blue **Update** button, we can navigate to **Monitoring** | **Latest data**. Add the following filters:



Host groups	type here to search
Hosts	lar-book-disc-win ✖ type here to search
Application	Counter Processor

Figure 6.35 – Latest data filter on host lar-book-disc-win

16. We can now see our three newly created items:



Name ▲	Last check	Last value	Change
Counter Processor (3 Items)			
CPU 0 - C1 time	2020-09-26 10:19:28	96.2951 %	-0.87 %
CPU 1 - C1 time	2020-09-26 10:19:29	99.5613 %	+0.21 %
CPU _Total - C1 time	2020-09-26 10:19:30	97.4105 %	-1.7093 %

Figure 6.36 – Monitoring | Latest data page for our host lar-book-disc-win

How it works…

Windows performance counters have been around for a long time and we were already able to monitor them with Zabbix. Unfortunately, we couldn't yet monitor them using **Low-Level Discovery** (**LLD**).

But since Zabbix 5.0.1 that has now changed, with the addition of discovering Windows performance counters.

We created a very simple but effective Windows performance counter discovery rule by adding the discovery rule with the `perf_instance.discovery[Processor]` item key. The `[Processor]` part of this item key directly correlates to the `perfmon.exe` window we saw. If we look at the following screenshot, we already see **Processor** listed:

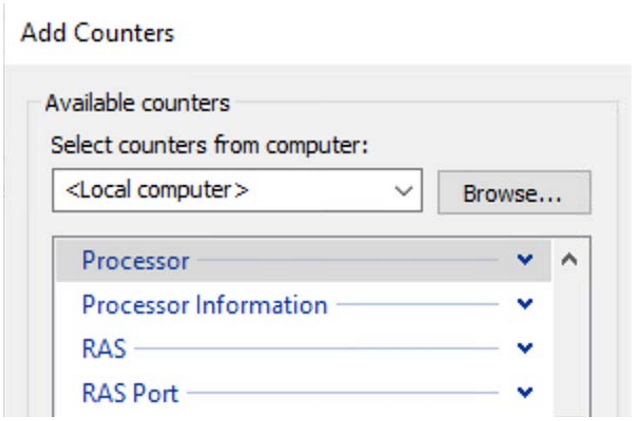


Figure 6.37 – Perfmon.exe | Add Counters, Processor

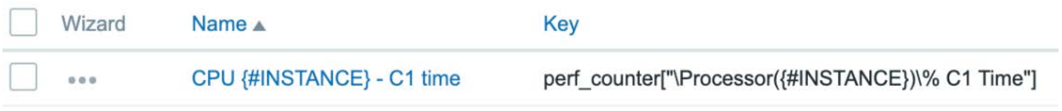
When our Discovery rule polls this item key, Zabbix Agent will return the following value for our host:

```
[{"{#INSTANCE}":"0"},{"{#INSTANCE}":"1"},{"{#INSTANCE}":"_
Total"}]
```

This value means that Zabbix will fill the `{#INSTANCE}` macro with three values:

- `0`
- `1`
- `_Total`

We can then use these three values by using the `{#INSTANCE}` macro in our item prototype, like we did here:



	Wizard	Name ▲	Key
	•••	CPU {#INSTANCE} - C1 time	perf_counter["\Processor({#INSTANCE})\% C1 Time"]

Figure 6.38 – Our created item prototype, CPU C1 time

It will then create three items with our macro values, with the right keys to monitor the second part of our counter—`% C1 time`. If you expand the window in your `perfmon.exe` file, you can see all the different counters we could add in our item prototypes to monitor more Windows performance counters:

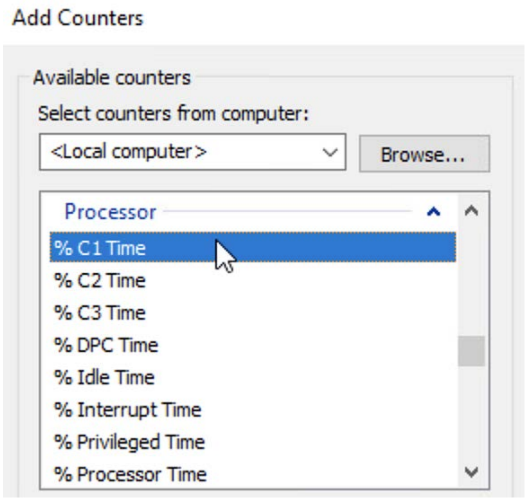


Figure 6.39 – Perfmon.exe | Add Counters, Processor expanded

Discovering JMX objects

In *Chapter 2*, *Setting up Zabbix Monitoring*, we went over setting up JMX monitoring in the recipe titled *Setting up JMX monitoring*. What we didn't cover yet though was discovering JMX objects.

In this recipe, we will go over how to set up JMX objects with LLD, and after you've finished this recipe, you'll know just how to set it up.

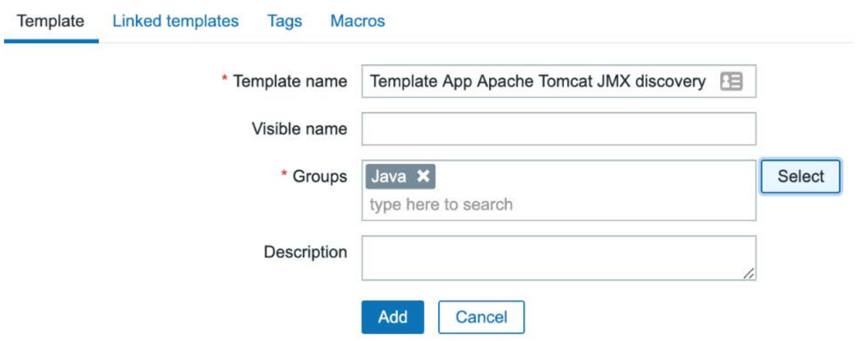
Getting ready

For this recipe, we will need the JMX host that you set up for the *Setting up JMX monitoring* recipe in *Chapter 2*, *Setting up Zabbix Monitoring*. Make sure to follow that recipe before working on this one.

We will also need our Zabbix server with our Zabbix JMX host titled `lar-book-jmx`.

How to do it…

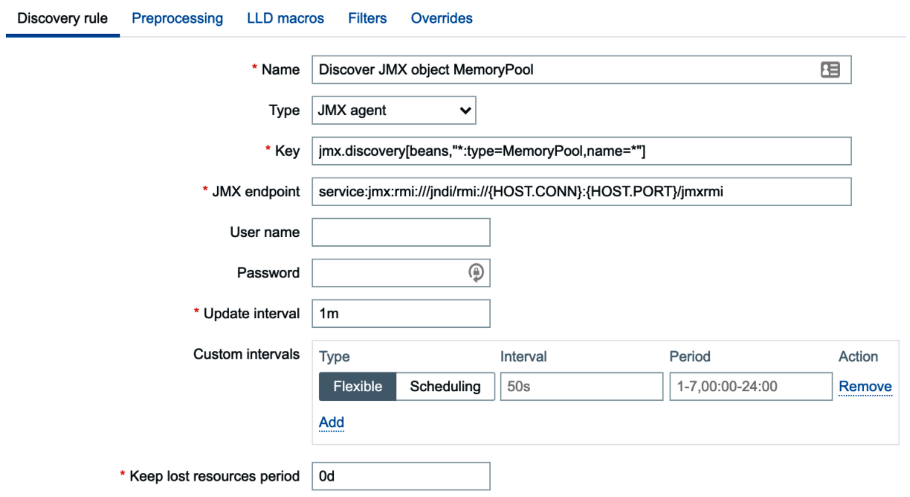
1. Let's start this recipe off by logging in to our Zabbix frontend and navigating to **Configuration** | **Templates**.
2. Create a new template by clicking on **Create template** in the top-right corner. Fill in the following fields:



* Template name	Template App Apache Tomcat JMX discovery
Visible name	
* Groups	Java ✖ type here to search
Description	

Figure 6.40 – Configuration | Templates, Create new template page, Apache Tomcat JMX

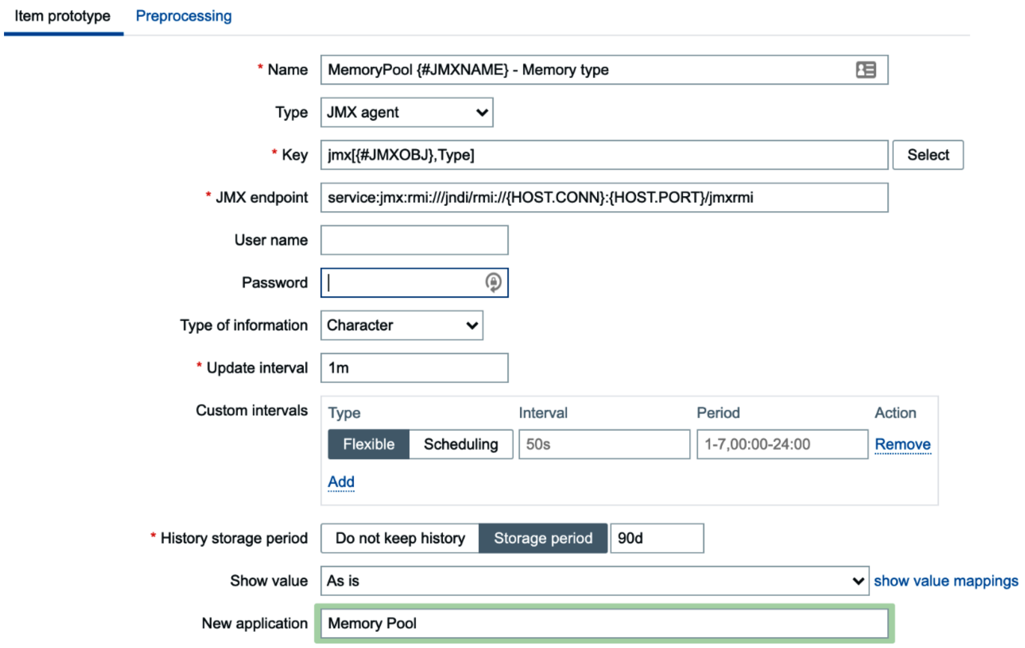
3. After clicking the blue **Add** button, you will be taken back to **Configuration** | **Templates**. Click on your new **Template App Apache Tomcat JMX discovery** template.
4. We will now add our JMX discovery rule. Click on **Discovery rules** next to our template name.
5. Now, click on **Create discovery rule** and fill in the following fields:



* Name	Discover JMX object MemoryPool
Type	JMX agent
* Key	jmx.discovery[beans,"*:type=MemoryPool,name=*"]
* JMX endpoint	service:jmx:rmi:///jndi/rmi://{HOST.CONN}:{HOST.PORT}/jmxrmi
User name	
Password	
* Update interval	1m
Custom intervals	Type: Flexible / Scheduling; Interval: 50s; Period: 1-7,00:00-24:00; Action: Remove
* Keep lost resources period	0d

Figure 6.41 – Configuration | Templates, create new template page, Discover JMX object MemoryPool

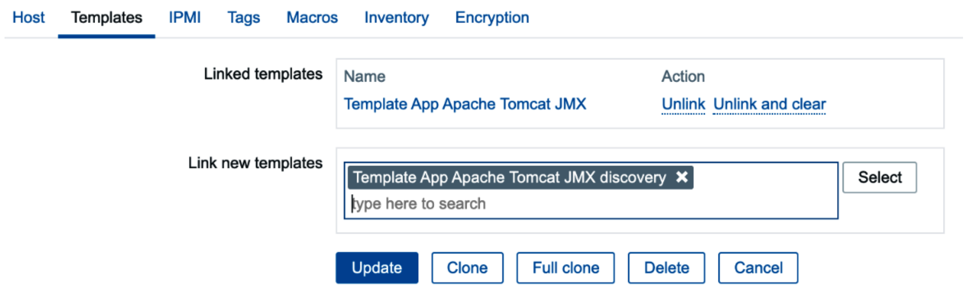
6. Click on the blue **Add** button at the bottom of the page. Then, click on **Item prototypes** next to your newly created **Discover JMX object MemoryPool** discovery rule.
7. We will now click on the **Create item protype** button in the top-right corner and create the following item prototype:



* Name	MemoryPool {#JMXNAME} - Memory type
Type	JMX agent
* Key	jmx[{#JMXOBJ},Type]
* JMX endpoint	service:jmx:rmi:///jndi/rmi://{HOST.CONN}:{HOST.PORT}/jmxrmi
User name	
Password	
Type of information	Character
* Update interval	1m
Custom intervals	Type: Flexible; Interval: 50s; Period: 1-7,00:00-24:00
* History storage period	Storage period 90d
Show value	As is
New application	Memory Pool

Figure 6.42 – Item prototype creation page, MemoryPool Memory type

8. Let's click on the blue **Add** button and move on.
9. Go to **Configuration** | **Hosts** and click on `lar-book-jmx`. We will add our template to this host.
10. Click on **Templates** and add the template, like this:



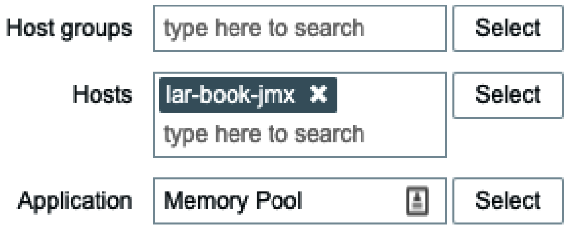
Linked templates:

Name	Action
Template App Apache Tomcat JMX	Unlink Unlink and clear

Link new templates: Template App Apache Tomcat JMX discovery

Figure 6.43 – Configuration | Hosts, add template to host lar-book-jmx

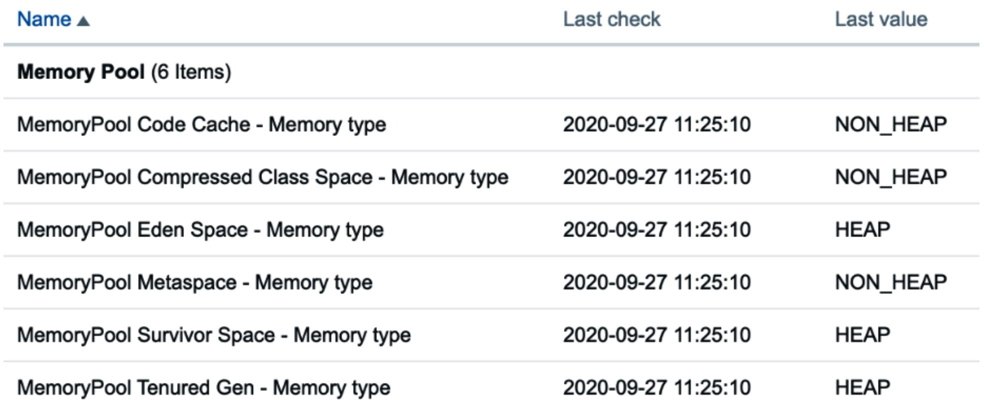
11. Click on the blue **Update** button.
12. When we navigate to **Monitoring | Latest data** now, we will filter on **Hosts**: `lar-book-jmx` and **Application**: `Memory pool`, like this:



Host groups	type here to search	Select
Hosts	lar-book-jmx ✖ type here to search	Select
Application	Memory Pool	Select

Figure 6.44 – Monitoring | Latest data page filters, host lar-book-jmx

13. We will then see the following results:



Name ▲	Last check	Last value
Memory Pool (6 Items)		
MemoryPool Code Cache - Memory type	2020-09-27 11:25:10	NON_HEAP
MemoryPool Compressed Class Space - Memory type	2020-09-27 11:25:10	NON_HEAP
MemoryPool Eden Space - Memory type	2020-09-27 11:25:10	HEAP
MemoryPool Metaspace - Memory type	2020-09-27 11:25:10	NON_HEAP
MemoryPool Survivor Space - Memory type	2020-09-27 11:25:10	HEAP
MemoryPool Tenured Gen - Memory type	2020-09-27 11:25:10	HEAP

Figure 6.45 – Monitoring | Latest data page for host lar-book-jmx with our results

How it works...

Monitoring JMX applications can be quite daunting at first, as there is a lot of work to figure out while building your own LLD rules. But now that you've built your first LLD rule for JMX, there is a clear structure in it.

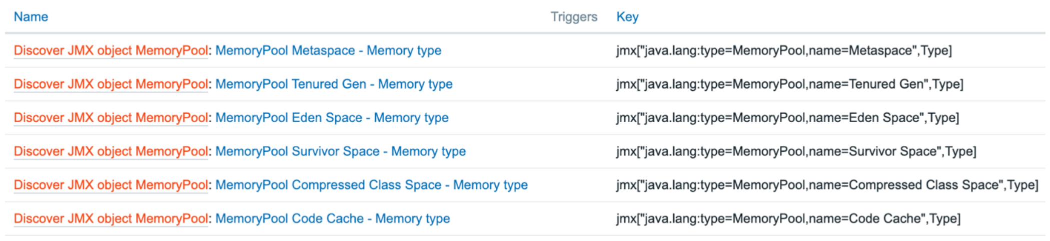
First, for our Discovery rule, we've picked the item key:

```
jmx.discovery[beans,"*:type=MemoryPool,name=*"]
```

`MemoryPool` is what we call an Mbean object in Java. We poll this Mbean object for several JMX objects and fill the macros accordingly.

We picked the `name=*` object to fill the `{#JMXNAME}` macro in this discovery rule. Our macro is then used in our item prototype to create our items.

Our items are then created, like this:



Name	Triggers	Key
Discover JMX object MemoryPool: MemoryPool Metaspace - Memory type		jmx["java.lang:type=MemoryPool,name=Metaspace",Type]
Discover JMX object MemoryPool: MemoryPool Tenured Gen - Memory type		jmx["java.lang:type=MemoryPool,name=Tenured Gen",Type]
Discover JMX object MemoryPool: MemoryPool Eden Space - Memory type		jmx["java.lang:type=MemoryPool,name=Eden Space",Type]
Discover JMX object MemoryPool: MemoryPool Survivor Space - Memory type		jmx["java.lang:type=MemoryPool,name=Survivor Space",Type]
Discover JMX object MemoryPool: MemoryPool Compressed Class Space - Memory type		jmx["java.lang:type=MemoryPool,name=Compressed Class Space",Type]
Discover JMX object MemoryPool: MemoryPool Code Cache - Memory type		jmx["java.lang:type=MemoryPool,name=Code Cache",Type]

Figure 6.46 – Items on our JMX-monitored host

If we look at the keys of the items, we can see that we poll the **Type** JMX attribute on every `MemoryPool` with different names.

That's how we create JMX LLD rules with ease.

There's more...

If you are not familiar with Mbeans objects, then make sure to check out the Java documentation. This will explain to you a lot about what Mbeans are and how they can be used for monitoring JMX attributes: `https://docs.oracle.com/javase/tutorial/jmx/mbeans/index.html`.

Tip

Before diving deeper into using JMX object discovery, dive deeper into the preceding JMX object documentation. There's a lot of information in it and it will greatly improve your skills in creating these LLD rules.

7 Setting Up Zabbix Proxies

You can't preach about Zabbix without actually preaching about the use of Zabbix proxies—a nice addition at first, but a no-brainer by now. Anyone that's expecting to set up a Zabbix environment of a medium/larger size will need proxies.

In this chapter, we will first learn how to set up a Zabbix proxy. We will then learn how to work with passive and active Zabbix proxies, and also how to monitor hosts with either form of Zabbix proxy. We will also cover Zabbix discovery using the proxies, and we'll learn how to monitor Zabbix proxies to keep them healthy. After these recipes, you'll have no more excuses for not setting up the proxies, as we'll cover most of the possible forms of proxy use in this chapter.

So, let's go through the following recipes and check out how to work with Zabbix proxies:

- Setting up a Zabbix proxy
- Working with passive Zabbix proxies
- Working with active Zabbix proxies
- Monitoring hosts with a Zabbix proxy
- Using discovery with Zabbix proxies
- Monitoring your Zabbix proxies

Technical requirements

We are going to need several new Linux hosts for this chapter, as we'll be building them as Zabbix proxies.

Set up two proxies by installing CentOS (or your preferred Linux distribution) to the following two new hosts:

- `lar-book-proxy-passive`
- `lar-book-proxy-active`

You'll also need the Zabbix server, with at least one monitored host. We'll be using the following new host with Zabbix agent installed:

- `lar-book-agent-by-proxy`

Setting up a Zabbix proxy

Setting up a Zabbix proxy can be quite daunting if you don't have a lot of experience with Linux, but the task is quite simple once you get the hang of it. We will install Zabbix proxy to our `lar-book-proxy-passive` server; you can repeat the task on `lar-book-proxy-active`.

Getting ready

Make sure to have your new Linux (CentOS 8) host ready and installed. We won't need our Zabbix server in this recipe yet.

You'll need a Zabbix repository for this recipe as well. Check out the following link to find the latest version: `https://www.zabbix.com/download`.

How to do it...

1. Let's start by logging in to the **command-line interface** (**CLI**) of our new `lar-book-proxy-passive` host.
2. Now, execute the following command to add the Zabbix repository for CentOS 8:

```
rpm -Uvh https://repo.zabbix.com/zabbix/5.0/rhel/8/
x86_64/zabbix-release-5.0-1.el8.noarch.rpm
dnf clean all
```

For Ubuntu, execute this command:

```
wget https://repo.zabbix.com/zabbix/5.0/ubuntu/pool/
main/z/zabbix-release/zabbix-release_5.0-1+focal_all.deb
```

```
dpkg -i zabbix-release_5.0-1+focal_all.deb
```

```
apt update
```

3. Now, install Zabbix proxy for CentOS 8 by executing the following command:

```
dnf install zabbix-proxy-sqlite3
```

For Ubuntu, execute this command:

```
apt install zabbix-proxy-sqlite3
```

Tip

On CentOS8 servers, don't forget to disable Security-Enhanced Linux (SELinux) or allow Zabbix proxy in SELinux for production. For lab environments it is fine to disable SELinux, but in production I would recommend leaving it enabled.

4. Now, edit the Zabbix proxy configuration by executing the following command:

```
vim /etc/zabbix/zabbix_proxy.conf
```

5. Let's start by setting the proxy mode on the `passive` proxy. The mode will be `1` on this proxy. On the `active` proxy, this will be `0`:

```
ProxyMode=1
```

6. Change the following line to your Zabbix server address:

```
Server=10.16.16.152
```

7. Change the following line to your proxy hostname:

```
Hostname=lar-book-proxy-passive
```

8. As we'll be using the `sqlite` version of the proxy for the example, change the `DBName` parameter to the following:

```
DBName=/tmp/zabbix_proxy.db
```

9. You could create the initial `sqlite3` database by executing the following command, but this is also done automatically on startup of `zabbix-proxy-sqlite3`:

```
zcat /usr/share/doc/zabbix-proxy-sqlite3/schema.sql.gz |
sqlite3 /tmp/zabbix_proxy.db
```

10. You can now enable Zabbix proxy and start it with the following two commands:

```
systemctl enable zabbix-proxy
```

```
systemctl start zabbix-proxy
```

11. You might want to check that the Zabbix proxy logs will restart, with the following command:

```
tail -f /var/log/zabbix/zabbix_proxy.log
```

How it works...

There are three versions of Zabbix proxy to work with:

- `zabbix-proxy-mysql`
- `zabbix-proxy-pgsql`
- `zabbix-proxy-sqlite3`

We've just done the setup for the `zabbix-proxy-sqlite3` package, which is the easiest method, if you ask me. The `sqlite3` version of Zabbix proxy makes it possible for us to set up a Zabbix proxy with great ease as we don't actually need to worry too much about database setup.

Please do note that the `sqlite3` versions might not be suited to Zabbix proxies with a lot of hosts. You get more options to scale a `mysql` or `pgsql` version of Zabbix proxy, by the fine-tuning mechanisms installed with those database types.

Tip

Always pick the right type of Zabbix proxy for what you expect to need in the future. Although it is easy to switch proxies later, don't go too easy on this choice as you might save yourselves hours in the future.

The amazing part about the `sqlite3` version is that, if we run into database issues, it's very easy to just run the following:

```
rm -rf /tmp/zabbix_proxy.db
```

Zabbix proxy then creates a new database on startup, and we're all ready to start collecting again. Do note that we might lose some information that is in the proxy database and not yet in the Zabbix database.

There's more...

More information about installing Zabbix proxies can be found here:

`https://www.zabbix.com/documentation/current/manual/installation/install_from_packages`

Choose your respected distribution, and you can find the guides for all the different variants of proxy installations.

Working with passive Zabbix proxies

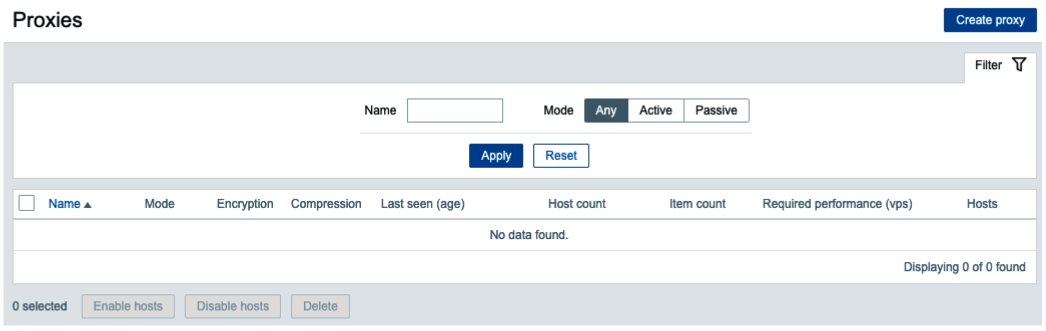
Now that we have installed our Zabbix proxy in the previous recipe, we can start working with it. Let's start by setting up our passive Zabbix proxy in the frontend and see what we can do with it from the start.

Getting ready

You will need the `lar-book-proxy-passive` host for this recipe, ready and installed with Zabbix proxy. We will also be using our Zabbix server in this recipe again.

How to do it...

1. Let's start by logging in to our Zabbix frontend and navigating to **Administration** | **Proxies**:



Proxies

Name: (empty) Mode: [✓ Any] Active Passive

Name ▲	Mode	Encryption	Compression	Last seen (age)	Host count	Item count	Required performance (vps)	Hosts
No data found.								

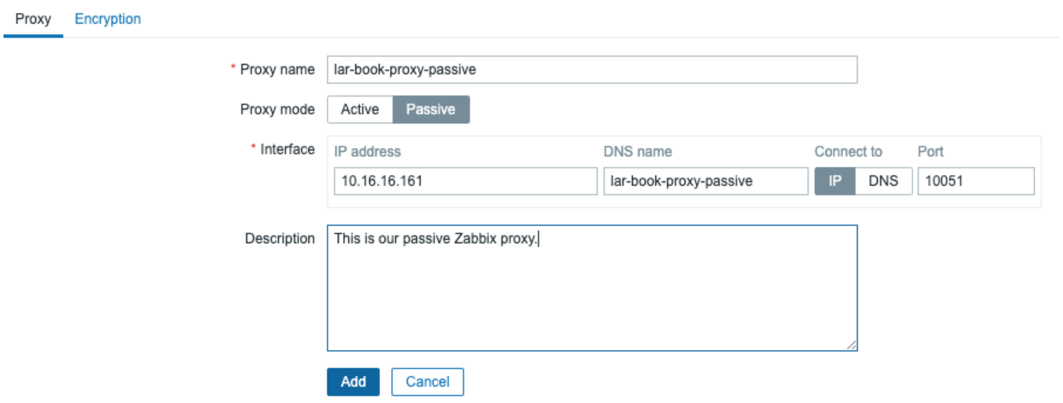
Displaying 0 of 0 found

0 selected

Figure 7.1 – Administration | proxies page, no passive proxies

Our **Proxies** page is where we do all configuration that's proxy-related.

2. Let's add a new proxy with the blue **Create proxy** button in the top-right corner.
3. This will take us to the **Create proxy** page, where we will fill out the following information:



Proxy | Encryption

* Proxy name: lar-book-proxy-passive

Proxy mode: Active [✓ Passive]

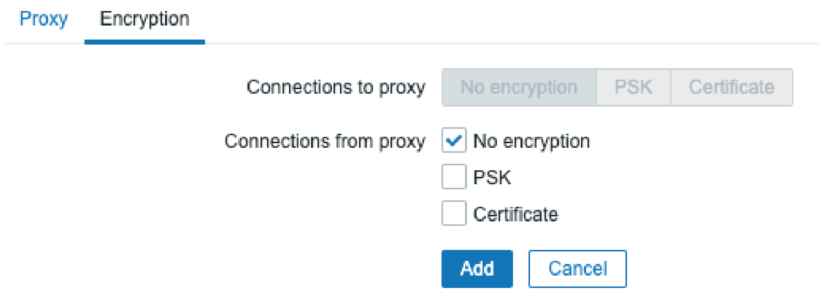
* Interface:

IP address	DNS name	Connect to	Port
10.16.16.161	lar-book-proxy-passive	[✓ IP] DNS	10051

Description: This is our passive Zabbix proxy.

Figure 7.2 – Administration | proxies, Create proxy page, lar-book-proxy-passive

4. Before clicking the blue **Add** button, let's take a look at the **Encryption** tab:



Connections to proxy: [✓ No encryption] PSK Certificate

Connections from proxy: [✓] No encryption, [ ] PSK, [ ] Certificate

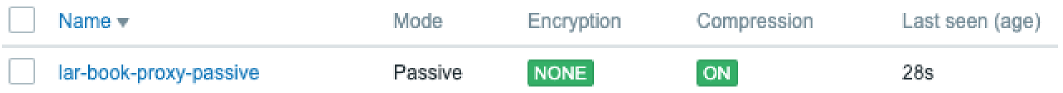
Figure 7.3 – Administration | proxies, Create proxy Encryption page, lar-book-proxy-passive

5. By default, **No encryption** is checked here, which we'll leave be for this recipe.

Important note

A lot of valuable information is exchanged by Zabbix server and Zabbix proxy. If you are working with insecure networks or just need an extra layer of security, use Zabbix proxy encryption. It's easy to set up and doesn't really impact performance. You can find more information on Zabbix encryption here: `https://www.zabbix.com/documentation/current/manual/encryption`.

6. Now, click the blue **Add** button, which will take us back to our proxy overview page.
7. The **Last seen (age)** part of your newly added proxy should now show a time value, instead of **Never**:



Name ▾	Mode	Encryption	Compression	Last seen (age)
lar-book-proxy-passive	Passive	NONE	ON	28s

Figure 7.4 – Administration | proxies page, Last seen (age)

How it works…

Adding proxies isn't the hardest task after we've already done the installation part. After the steps we took in this recipe, we are 100% ready to start monitoring with this proxy.

The proxy we just added is a Passive proxy. These proxies work by receiving configuration from Zabbix server, which the Zabbix server sends to the Zabbix proxy on port `10051`:

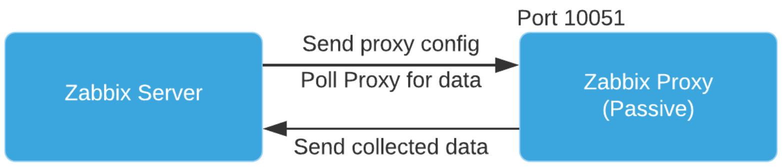


Figure 7.5 – Diagram showing active proxy connection

The Zabbix server then keeps sending config changes and it keeps polling the Zabbix proxy. The proxy then in turn returns the collected data to our Zabbix server.

Working with active Zabbix proxies

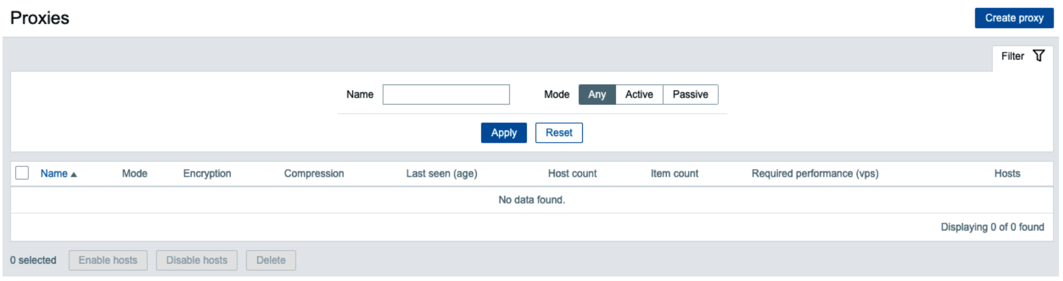
We now know how to install and add proxies. Let's set up our active proxy, like we did with the passive proxy in the previous recipe, and see how it works.

Getting ready

You will need the `lar-book-proxy-active` host for this recipe, ready and installed with Zabbix proxy. We will also be using our Zabbix server in this recipe.

How to do it...

1. Let's start by logging in to our Zabbix frontend and navigating to **Administration** | **Proxies**:



Name ▲	Mode	Encryption	Compression	Last seen (age)	Host count	Item count	Required performance (vps)	Hosts

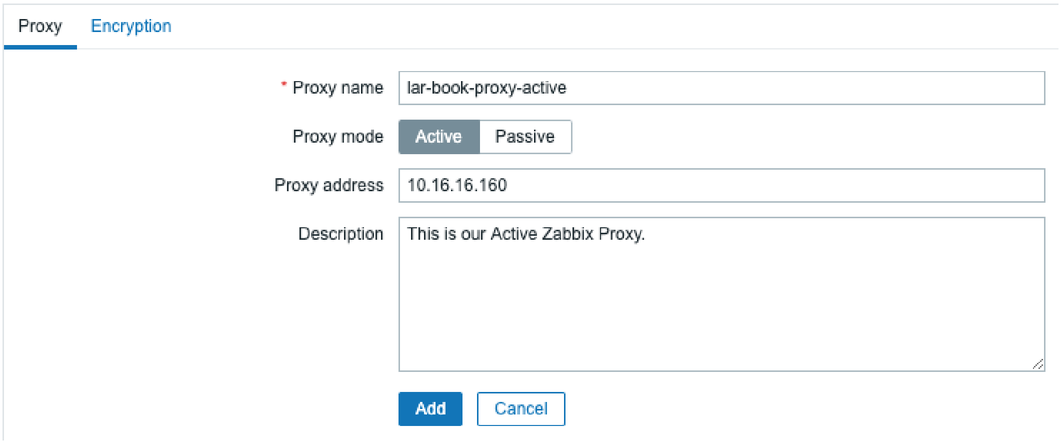
No data found.

Displaying 0 of 0 found

Figure 7.6 – Administration | proxies page, no active proxies

2. Our **Proxies** page is where we do all configuration that's proxy-related.
3. Let's add a new proxy by clicking the blue **Create proxy** button in the top-right corner.

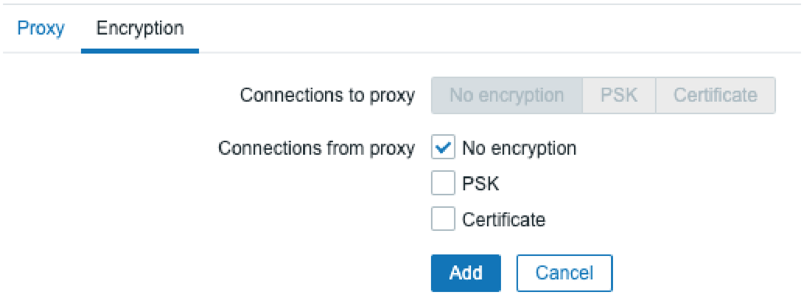
4. This will take us to the Create proxy page, where we will fill out the following information:



* Proxy name	lar-book-proxy-active
Proxy mode	[✓ Active] Passive
Proxy address	10.16.16.160
Description	This is our Active Zabbix Proxy.

Figure 7.7 – Administration | proxies, Create proxy page, lar-book-proxy-active

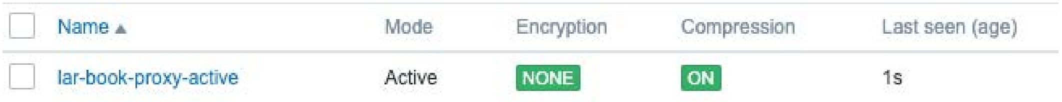
5. Before clicking the blue **Add** button, let's take a look at the **Encryption** tab:



Connections to proxy	No encryption PSK Certificate
Connections from proxy	✓ No encryption ☐ PSK ☐ Certificate

Figure 7.8 – Administration | proxies, Create proxy Encryption page, lar-book-proxy-active

6. By default, **No encryption** is checked here, which we'll leave be for this recipe.
7. Now, click the blue **Add** button, which will take us back to the proxy overview page.
8. The **Last seen (age)** part of your newly added proxy should now show a time value, instead of **Never**:



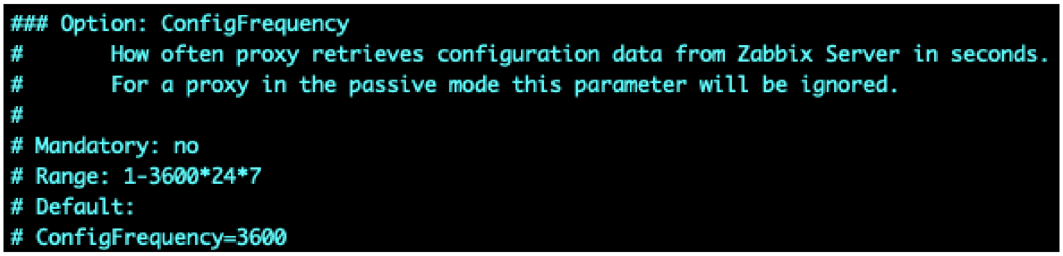
☐	Name ▲	Mode	Encryption	Compression	Last seen (age)
☐	lar-book-proxy-active	Active	NONE	ON	1s

Figure 7.9 – Administration | proxies page, Last seen (age)

9. Depending on your setting in the proxy configuration file, the **Last seen (age)** part may take a while to change. Log in to the CLI and check the configuration with the following command:

```
vim /etc/zabbix/zabbix_proxy.conf
```

10. By default, the frequency at which the proxy requests configuration changes is `3600` seconds (that is, 1 hour):



```
### Option: ConfigFrequency
#       How often proxy retrieves configuration data from Zabbix Server in seconds.
#       For a proxy in the passive mode this parameter will be ignored.
#
# Mandatory: no
# Range: 1-3600*24*7
# Default:
# ConfigFrequency=3600
```

Figure 7.10 – Zabbix proxy configuration file, ConfigFrequency

> **Important note**
> Polling the Zabbix server for configuration too often might increase load, but polling too infrequently will leave your Zabbix proxy waiting for a new configuration. Choose a frequency that is best suited to your environment.

How it works...

If you followed the *Working with passive Zabbix proxies* recipe from this chapter, the steps are about the same, except for the part where we add the proxy mode and the part where we checked the `ConfigFrequency` value.

The proxy we just added is an active proxy that works by requesting a configuration from Zabbix server on port `10051`. The server then sends the configuration data back to the Zabbix proxy:

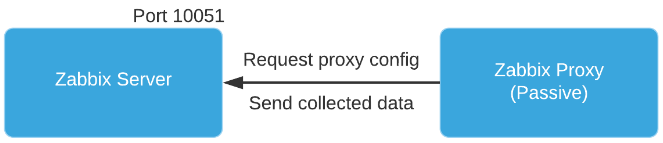


Figure 7.11 – Diagram showing passive proxy connection

The Zabbix proxy keeps requesting configuration changes, and it keeps sending any new collected data to the Zabbix server when available.

Important note

It is recommended to use active Zabbix proxies, as they are faster in sending new data to the server.

Monitoring hosts with Zabbix proxy

We have our active and passive Zabbix proxies ready to use, so it's now time to add some hosts to them. Setting up the Zabbix frontend to monitor hosts with Zabbix proxies works in about the same way as monitoring directly from the Zabbix server. The backend and design change completely though, which I'll explain in the *How it works…* section of this recipe.

Getting ready

Make sure you have your `lar-book-proxy-passive` passive proxy and your `lar-book-proxy-active` active proxy ready by following all of the previous recipes in this chapter.

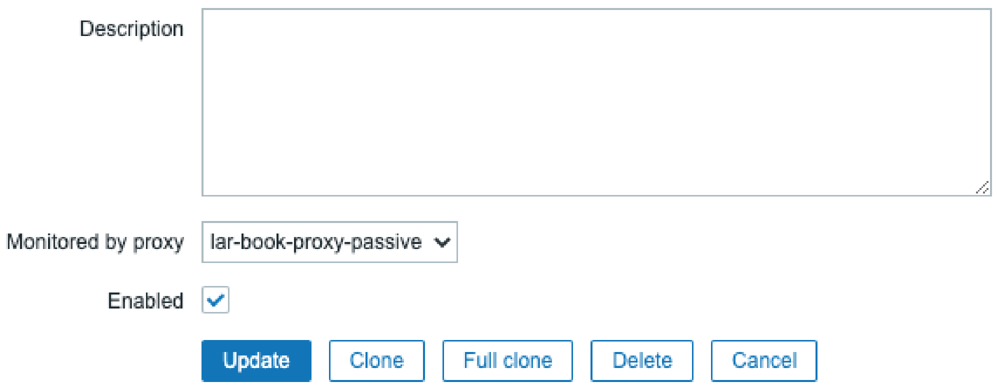
You will also need your Zabbix server and at least two hosts to monitor. We will be using `lar-book-agent_snmp` and `lar-book-agent` in the example, but any host with an active and passive Zabbix agent will work.

How to do it…

We'll configure a host on both our active and our passive proxies to show you what the difference is between these two. Let's start with the passive proxy.

Passive proxy

1. Let's start off this recipe by logging in to our Zabbix frontend and navigating to **Configuration** | **Hosts**.
2. Let's add the host with the passive agent to our passive proxy. In my case, this is the `lar-book-agent_snmp` host.
3. Click on the `lar-book-agent_snmp` host and change the **Monitored by proxy** field to `lar-book-proxy-passive`, as in the following screenshot:


Description

Monitored by proxy: lar-book-proxy-passive

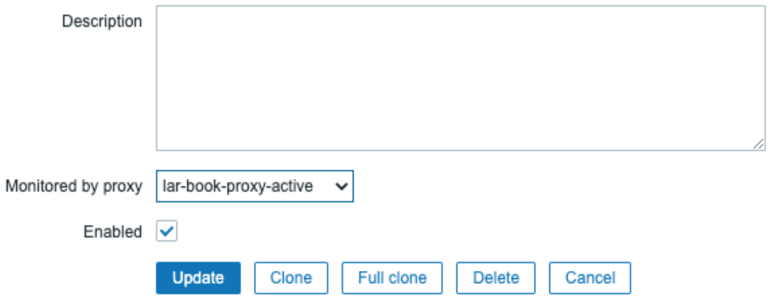
Enabled: ✓

Figure 7.12– Configuration | Hosts, Edit host page for host lar-book-agent_snmp

4. Now, click on the blue **Update** button. Our host will now be monitored by the Zabbix proxy.

Active proxy

1. Let's do the same for our other `lar-book-agent` host by navigating back to **Configuration** | **Hosts**.
2. Click on the `lar-book-agent` host and edit the **Monitored by proxy** field to `lar-book-proxy-active`, as in the following screenshot:


Description

Monitored by proxy: lar-book-proxy-active

Enabled: ✓

Figure 7.13 – Configuration | Hosts, Edit host page for host lar-book-agent

3. Now, click on the blue **Update** button. Our host will now be monitored by the Zabbix proxy.

How it works...

Monitoring hosts with a Zabbix proxy in passive or active mode works in the same way from the frontend. We merely configure in our frontend which host is monitored by which proxy, and it will be done.

Let's take a look at how our **Simple Network Management Protocol** (**SNMP**) agent is now monitored by the passive proxy:

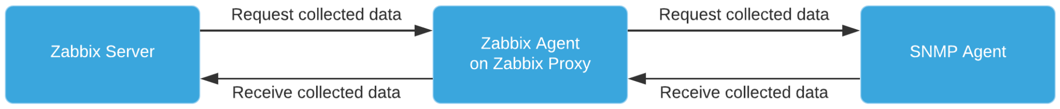


Figure 7.14 – A completely passive Zabbix setup with proxy

Our passive Zabbix proxy now collects data from our SNMP agent, and after this is collected, Zabbix server collects this data from our Zabbix proxy. Sounds like a whole process already, right?

Let's look at our active Zabbix proxy setup:

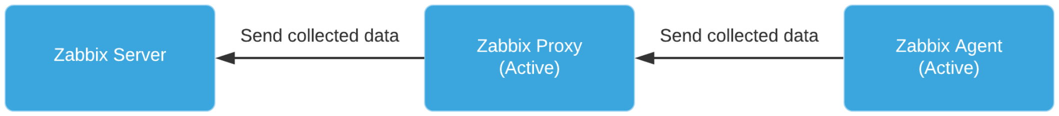


Figure 7.15 – A completely active Zabbix setup with proxy

Our active Zabbix proxy receives data from our active Zabbix agent and then sends this data to our Zabbix server. We've eliminated all the timers in this proxy setup altogether and are now receiving all of our data at the Zabbix server as soon as it's available.

This is why I would always recommend working with active proxies—and even active agents—as much as possible. If we look at the following screenshot, we can see a setup that you might see at a company:

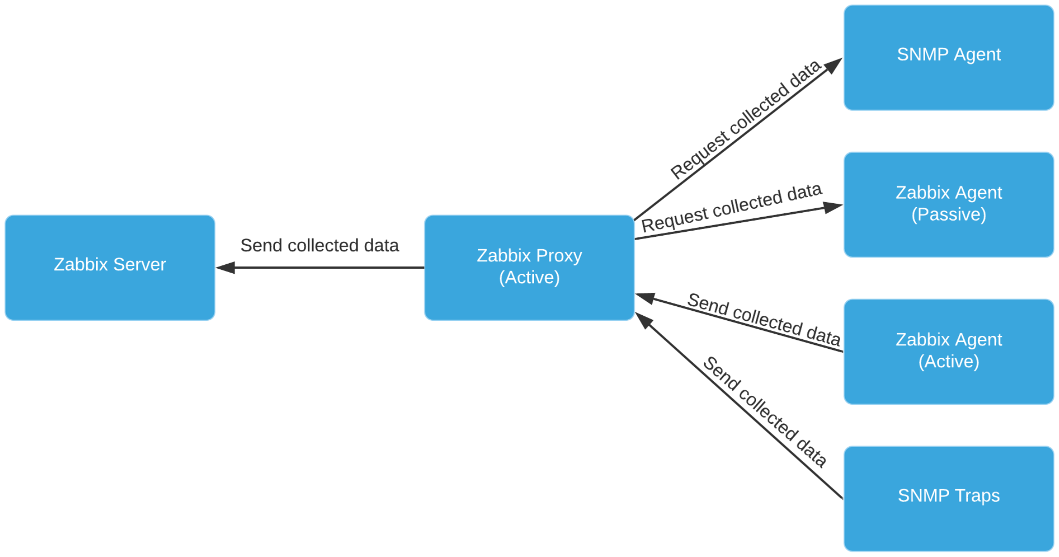


Figure 7.16 – An active Zabbix proxy setup with different monitored types

Fortunately, we have the option of using a lot of different combinations of setups. It is perfectly possible—and even logical—to combine your checks from a proxy, just as much as it would be with Zabbix server. We can monitor all types from our proxy, whether it's a Zabbix agent, SNMP, or even **Java Management Extensions** (**JMX**) and the **Intelligent Platform Management Interface** (**IPMI**).

Tip

When designing a new Zabbix hosting infrastructure, start with adding proxies if possible. This way, you don't have to change a lot later. It's easy to add and change proxies, but it's harder to go from just using Zabbix server to using Zabbix proxies in your design.

There's more...

We now have a solid setup with some proxies up and running. We've figured out the difference between active and passive proxies and how they affect monitoring. But why would we build a setup like this? Well, Zabbix proxies are great for many environments—not just the big ones, but even sometimes in the smallest ones.

We can use Zabbix proxies to offload polling and preprocessing from our main Zabbix server, thus keeping the server clear for showing data and handling our main Zabbix database connection.

We can use Zabbix proxies to monitor offsite locations, such as when you're a **Managed Service Provider** (**MSP**) and want to monitor a big customer network. We simply place a proxy on site and monitor this through a **virtual private network** (**VPN**), for example. When the VPN goes down, our proxy will keep collecting data on site and send this to our Zabbix server when the VPN comes back up.

We can also use the Zabbix proxy to bypass firewall complications. When we place a proxy behind a firewall in a monitored network, we only need one firewall rule between the Zabbix server and Zabbix proxy. Our Zabbix proxy then monitors the different hosts and sends the collected data in one stream to the Zabbix server.

With this, you have loads of options to use Zabbix proxies already.

See also

Check out this interesting blog post by Dmitry Lambert about some more cool hidden benefits of Zabbix proxies: `https://blog.zabbix.com/hidden-benefits-of-zabbix-proxy/9359/`.

Dmitry is an experienced Zabbix engineer and head of Zabbix Customer Support. His blog posts are easy to understand and bring in some new angles to look at Zabbix from.

Using Discovery with Zabbix proxies

In *Chapter 6*, *Using discovery for Automatic Creation*, we talked about Zabbix and discovery. It is a very good idea to edit your discovery rules if you followed along with that chapter. Let's see how this would work in this recipe.

Getting ready

You'll need to have finished *Chapter 6*, *Using discovery for Automatic Creation,* or have some discovery rules and **active agent autoregistration** setup.

I'll be using `lar-book-lnx-agent-auto`, `lar-book-disc-lnx`, and `lar-book-disc-win` hosts in this example. We will also need our Zabbix server.

How to do it…

Let's start with editing our discovery rule and then move on to editing our active agent to autoregister to the proxy.

Discovery rules

Starting with Zabbix discovery rules, let's look at how to make sure we do this from the Zabbix proxy:

1. Log in to the CLI of `lar-book-disc-lnx` and edit the `/etc/zabbix/zabbix_agent2.conf` file. Edit the following lines to include our Zabbix proxy address:

```
Server=127.0.0.1,10.16.16.152,10.16.16.160,10.16.16.161
```

```
ServerActive=127.0.0.1,10.16.16.152,10.16.16.160,
10.16.16.161
```

2. Restart your Zabbix agent by executing the following command:

```
systemctl restart zabbix_agent2
```

3. Now, make sure to log in to `lar-book-disc-win` and edit the `C:\Program Files\Zabbix agent\zabbix_agentd` file. Edit the following lines to include our Zabbix proxy address:

```
Server=127.0.0.1,10.16.16.152,10.16.16.160,10.16.16.161
```

```
ServerActive=127.0.0.1,10.16.16.152,10.16.16.160,
10.16.16.161
```

4. Restart your Zabbix agent by executing the following commands in the Windows command line:

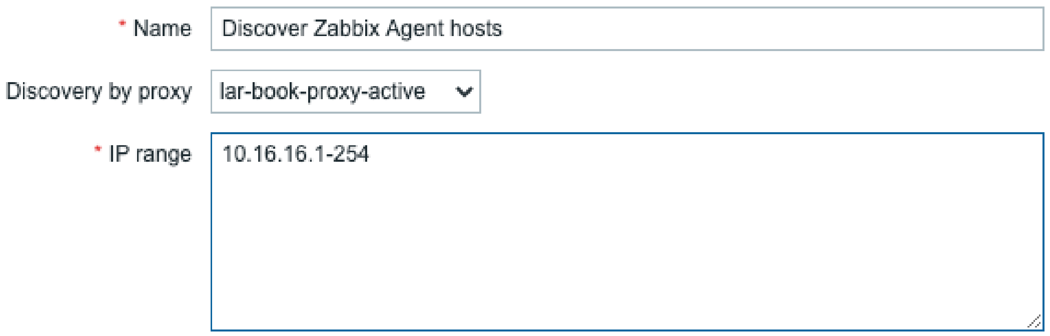
```
zabbix_agentd.exe --stop
zabbix_agentd.exe --start
```

5. Next, navigate to **Configuration** | **Hosts** and delete the discovered hosts:

   `lar-book-disc-lnx`

   `lar-book-discx-win`

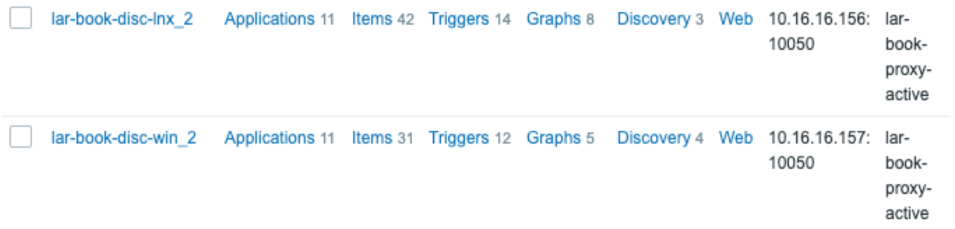
   We do this to prevent duplicate hosts.
6. Now, navigate to **Configuration** | **Discovery**.
7. Click on **Discover Zabbix Agent hosts** and change the **Discovered by proxy** field, as shown in the following screenshot:



* Name	Discover Zabbix Agent hosts
Discovery by proxy	lar-book-proxy-active
* IP range	10.16.16.1-254

Figure 7.17 – Configuration | Actions, drop-down menu for discovery by proxy lar-book-proxy-active

8. Click on the blue **Update** button, and that's all there is to editing your discovery rule to be monitored by a proxy.
9. You can now check out your newly discovered hosts under **Configuration** | **Hosts** and see that they are monitored by the proxy:



☐	lar-book-disc-lnx_2	Applications 11	Items 42	Triggers 14	Graphs 8	Discovery 3	Web	10.16.16.156: 10050	lar-book-proxy-active
☐	lar-book-disc-win_2	Applications 11	Items 31	Triggers 12	Graphs 5	Discovery 4	Web	10.16.16.157: 10050	lar-book-proxy-active

Figure 7.18 – Configuration | Hosts screen for discovered hosts

Active agent autoregistration

Moving on to active agent autoregistration, let's see how we can do this from our Zabbix proxy:

1. Start by navigating to **Configuration** | **Hosts** and deleting `lar-book-lnx-agent-auto`.
2. To do active agent autoregistration to a proxy, we have to log in to our `lar-book-lnx-agent-auto` host CLI.
3. Edit the Zabbix agent configuration file with the following command:

```
vim /etc/zabbix/zabbix_agent2.conf
```

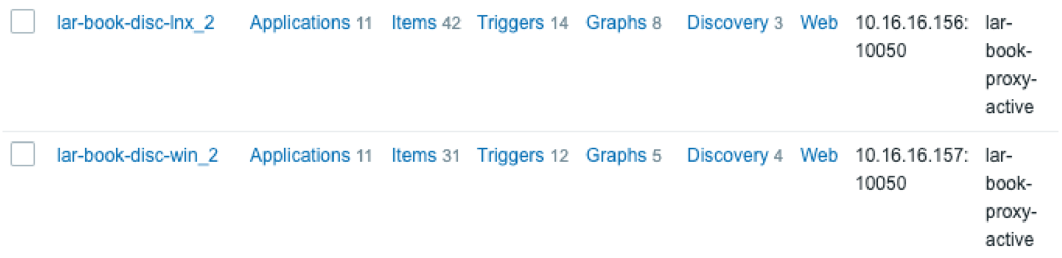
4. Make sure to edit the following line to the Zabbix proxy address instead of the Zabbix server address:

```
ServerActive=10.16.16.160
```

5. Restart the Zabbix agent:

```
systemctl restart zabbix-agent2.service
```

6. We can now see our host autoregister to the Zabbix proxy instead of the Zabbix server:



☐	lar-book-disc-lnx_2	Applications 11	Items 42	Triggers 14	Graphs 8	Discovery 3	Web	10.16.16.156: 10050	lar-book-proxy-active
☐	lar-book-disc-win_2	Applications 11	Items 31	Triggers 12	Graphs 5	Discovery 4	Web	10.16.16.157: 10050	lar-book-proxy-active

Figure 7.19 – Configuration | Hosts screen for our two auto registered hosts

How it works...

Discovery with a Zabbix proxy works exactly the same as discovery with Zabbix server. The only thing that changes is the location of where we are registering to or discovering from.

If you want to learn more about the process of discovery and auto registration, check out *Chapter 6, Using Discovery for Automatic Creation* if you haven't already.

Monitoring your Zabbix proxies

A lot of Zabbix users—or even monitoring users in general—forget a very important part of their monitoring. They forget to monitor the monitoring infrastructure. I want to make sure that when you set up Zabbix proxies, you also know how to monitor the health of these proxies.

Let's check out how to do so in this recipe.

Getting ready

For this recipe, we will need our new `lar-book-proxy-active` Zabbix proxy. We will also need our Zabbix server to monitor the Zabbix proxy.

How to do it...

We are going to build some monitoring in our Zabbix frontend, but we'll also check the integrated monitoring options for Zabbix proxies. Let's start by building our own.

Monitoring the proxy with Zabbix

We can monitor our Zabbix proxy with Zabbix proxy itself to make sure we know exactly what's going on:

1. Let's start by logging in to our `lar-book-proxy-active` Zabbix proxy CLI.
2. Issue the following command to install Zabbix agent 2 for CentOS 8:

   ```
   dnf install zabbix-agent2
   ```

   For Ubuntu, issue this command:

   ```
   apt-get install zabbix-agent2
   ```

3. Edit the Zabbix agent configuration file by issuing the following command:

   ```
   vim /etc/zabbix/zabbix_agent2.conf
   ```

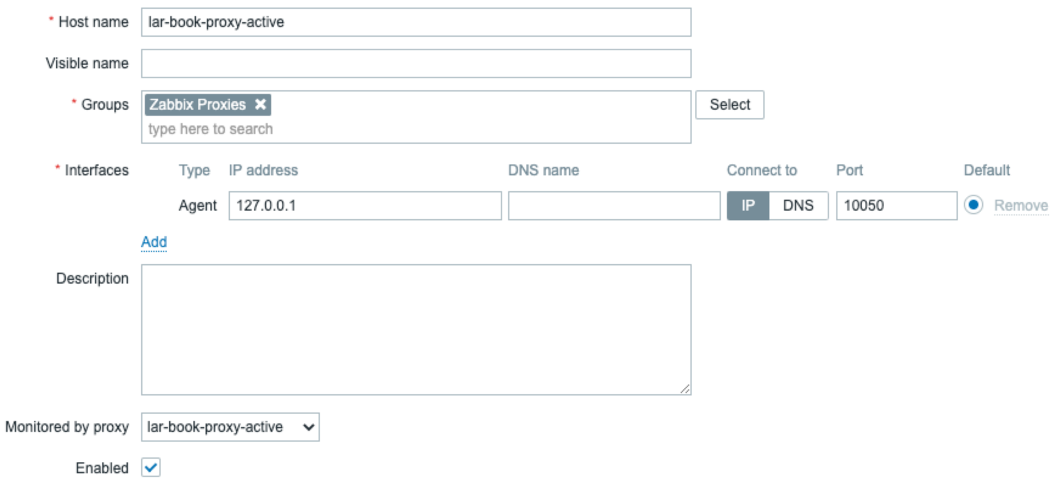
4. Edit the following lines to point toward `localhost`:

   ```
   Server=127.0.0.1
   ServerActive=127.0.0.1
   ```

5. Also, make sure to add the hostname to the Zabbix agent file:

   ```
   Hostname=lar-book-proxy-active
   ```

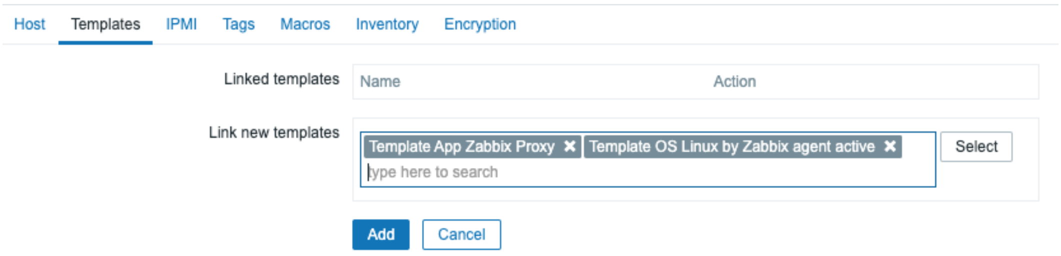
6. Now, log in to the Zabbix frontend and navigate to **Configuration | Hosts**.
7. Click on the blue **Create host** button in the top-right corner and add the following host:



* Host name	lar-book-proxy-active
Visible name	
* Groups	Zabbix Proxies
* Interfaces	Type: Agent; IP address: 127.0.0.1; DNS name: ; Connect to: IP; Port: 10050; Default: ●
Description	
Monitored by proxy	lar-book-proxy-active
Enabled	✓

Figure 7.20 – Configuration | Hosts, Create host page, lar-book-proxy-active

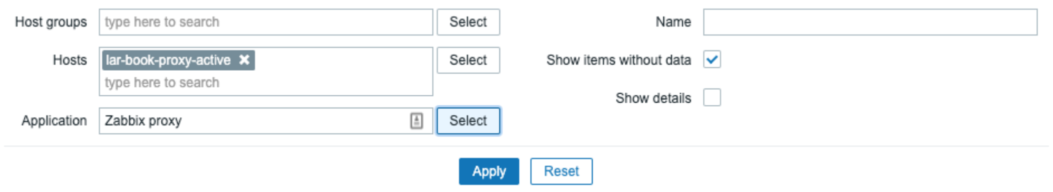
8. Take extra care at the **Monitored by proxy** field—we want to monitor from the proxy, because we are doing Zabbix internal checks, which need to be done to `localhost`.
9. Before clicking the blue **Add** button, click on **Templates**.
10. Add the following templates to the host:



Linked templates	Name / Action
Link new templates	Template App Zabbix Proxy, Template OS Linux by Zabbix agent active

Figure 7.21 – Configuration | Hosts, Create host page Templates tab for host lar-book-proxy-active

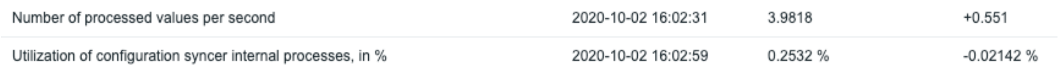
11. We can now press the blue **Add** button to create the host.
12. Now, navigate to **Monitoring** | **Latest data** and add the following filters:



Host groups	type here to search	Select	Name	
Hosts	lar-book-proxy-active ✖ type here to search	Select	Show items without data	✓
			Show details	
Application	Zabbix proxy	Select		

Figure 7.22 – Monitoring | Latest data page with filters, host lar-book-proxy-active

13. We can now see our Zabbix proxy statistics, such as **Number of processed values per second** and **Utilization of configuration syncer internal processes**:



Number of processed values per second	2020-10-02 16:02:31	3.9818	+0.551
Utilization of configuration syncer internal processes, in %	2020-10-02 16:02:59	0.2532 %	-0.02142 %

Figure 7.23 – Monitoring | Latest data page with data from our Zabbix proxy

Monitoring the proxy remotely from Zabbix server

We can also monitor our Zabbix proxy remotely from our Zabbix server, so let's see how that works:

1. Let's start by logging in to our `lar-book-proxy-active` host CLI and editing the following file:

```
vim /etc/zabbix/zabbix_agent2.conf
```

2. Edit the following lines to match your Zabbix server address:

```
Server=127.0.0.1,10.16.16.152
ServerActive=127.0.0.1,10.16.16.152
```

3. Also, edit the following file:

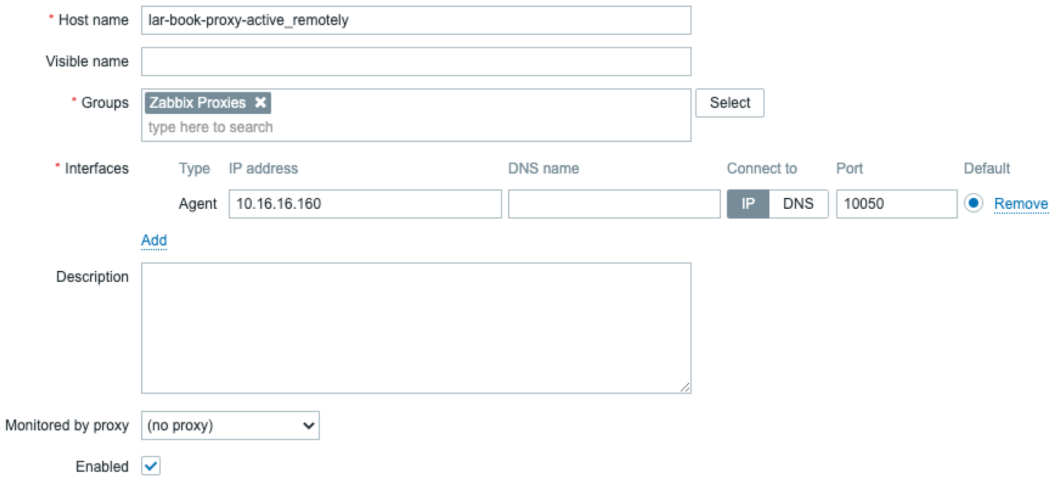
```
vim /etc/zabbix/zabbix_proxy.conf
```

4. Edit the following line to match your Zabbix server address:

```
StatsAllowedIP=127.0.0.1,10.16.16.152
```

5. Now, navigate to the Zabbix frontend and go to **Configuration** | **Hosts**.

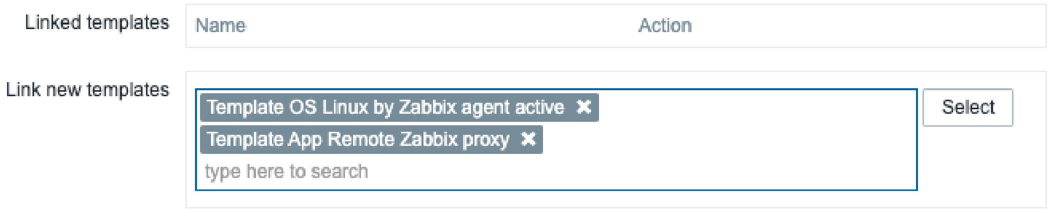
6. Click on the blue **Create host** button in the top-right corner and add the following host:



* Host name	lar-book-proxy-active_remotely
Visible name	
* Groups	Zabbix Proxies ✖ type here to search
* Interfaces	Agent 10.16.16.160, Connect to: IP, Port 10050, Default ●
Description	
Monitored by proxy	(no proxy)
Enabled	✓

Figure 7.24 – Configuration | Hosts, Create host page, lar-book-proxy-active_remotely

7. Before clicking the blue **Add** button, click on **Templates**.
8. Add the following templates to the host:

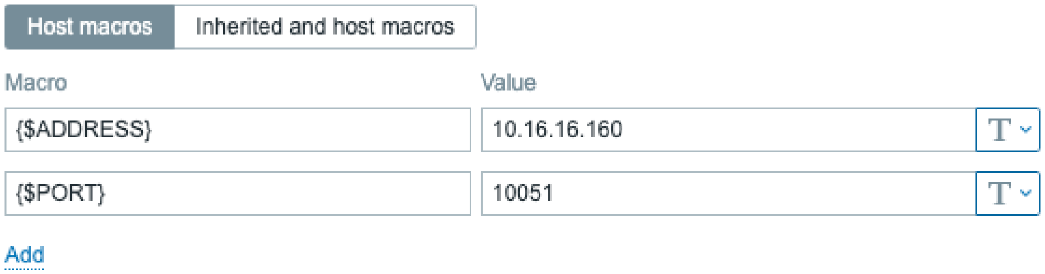


Linked templates	Name, Action
Link new templates	Template OS Linux by Zabbix agent active ✖ Template App Remote Zabbix proxy ✖ type here to search

Figure 7.25 – Configuration | Hosts, create new host page, Templates tab, lar-book-proxy-active_ remotely

9. We can now press the blue **Add** button to create the host.
10. Back at **Configuration** | **Hosts**, click on your new `lar-book-proxy-active_ remotely` host.

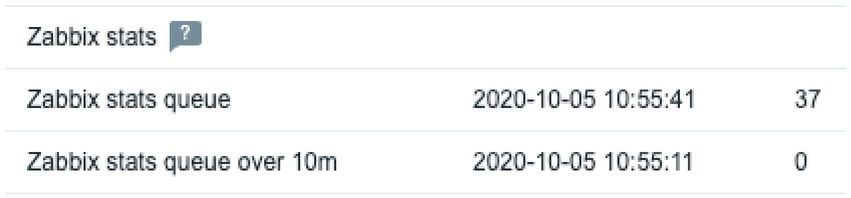
11. Go to **Macros** and add the following two macros:



Macro	Value
{$ADDRESS}	10.16.16.160
{$PORT}	10051

Figure 7.26 – Configuration | Hosts, Edit host page, Macros tab, lar-book-proxy-active_remotely

12. Now, click on the blue **Update** button, and that's it for this host.
13. If we navigate to **Monitoring** | **Latest data** and check the items for this host, we can see data coming in:



Zabbix stats		
Zabbix stats queue	2020-10-05 10:55:41	37
Zabbix stats queue over 10m	2020-10-05 10:55:11	0

Figure 7.27 – Monitoring | Latest data page for host lar-book-proxy-active_remotely

Monitoring the proxy from the Zabbix frontend

1. Let's start this off by navigating to **Administration** | **Queue**.
2. Use the drop-down menu to go to **Queue overview by proxy**:

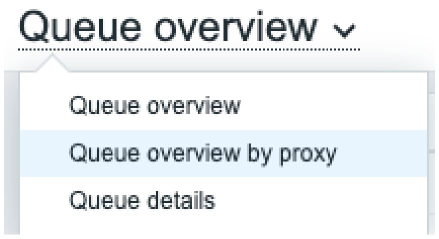
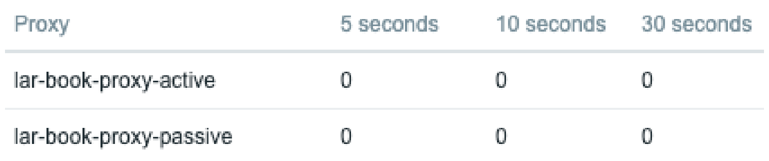


Figure 7.28 – Administration | Queue page drop-down menu

3. This will bring us to the page shown in the following screenshot:



Proxy	5 seconds	10 seconds	30 seconds
lar-book-proxy-active	0	0	0
lar-book-proxy-passive	0	0	0

Figure 7.29 – Administration | Queue page

How it works...

Monitoring your Zabbix proxies is an important task, thus we need to make sure that whenever we add a new Zabbix proxy, we are taking care of it like we would any other host.

Monitoring the proxy with Zabbix

By adding the Zabbix proxy as a host, we can make sure the Linux system that is running our Zabbix proxy is healthy. We also make sure that the Zabbix proxy applications running on this server are in good health.

Besides having the right triggers in these templates, we also get a load of options to troubleshoot issues with Zabbix proxy.

Zabbix proxy works just like Zabbix server when it comes to monitoring. This means that just as with Zabbix server, we need to keep the proxies in great health by tweaking the Zabbix proxy configuration file to our needs.

Scaling your proxies becomes a lot easier once you figure out what's going on with them. So, this is why we make sure to always monitor them. We monitor them from the proxy itself to make sure that we get the right information with the Zabbix internal checks.

Monitoring the proxy remotely from Zabbix server

Now, when we monitor with the **Template App Remote Zabbix proxy**, things go a little differently. Instead of doing our checks from the Zabbix proxy itself, we do them remotely from the Zabbix server by defining the Zabbix proxy address and port in the macros:

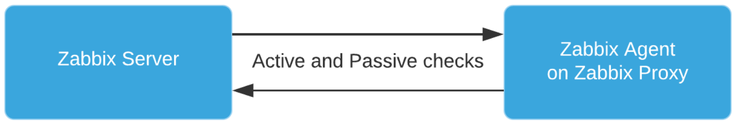


Figure 7.30 – Zabbix agent running on Zabbix proxy monitored by Zabbix server

This way, our Zabbix server is the one requesting and receiving information. Even when the proxy is having issues, the checks will still be done by Zabbix server.

We can use this template as a way to keep a closer eye on our proxy if we suspect issues with internal checks being performed locally, or we can use this template to bypass certain firewall setups. Both are valid reasons.

Monitoring the proxy from the Zabbix frontend

From the frontend, we can use the **Administration | Queue** page to monitor our proxies. The Zabbix **Queue** page is an important page, but a lot of new users neither know nor fully utilize this page.

When a part of Zabbix starts performing poorly, such as our example Zabbix proxy here, that's when we can see stuff happening in the queue. There are six options in the Zabbix **Queue**:

- **5 seconds**
- **10 seconds**
- **30 seconds**
- **1 minute**
- **5 minutes**
- **More than 10 minutes**

What the options in the **Queue** mean is that the Zabbix proxy has been waiting on receiving a value that's configured more than expected. I would state that anything up to 1 minute doesn't necessarily have to be an issue, but this depends on the type of check. The 5-minutes or more-than-10-minutes options can mean serious performance issues with your Zabbix proxy, and you would have to troubleshoot this issue. Make sure to keep a good eye on the Zabbix queue when you suspect issues.

8 Integrating Zabbix with External Services

In this chapter, we are going to set up some of the useful external service integrations that Zabbix has to offer. We can use these external services to notify our Zabbix users of ongoing problems.

We will start by learning how to set up in-company chat applications such as Slack and Microsoft Teams. Then, we will learn how to use a personal chat application such as Telegram before learning how to integrate Atlassian Opsgenie for even more extensive alerting.

Once you've completed these recipes, you will be able to effectively integrate certain services with Zabbix. This is a good starting point for working with external services in general and the easiest way to set up Slack, Teams, Opsgenie, and Telegram.

In this chapter, we will cover the following recipes:

- Setting up Slack alerting with Zabbix
- Setting up Microsoft Teams alerting with Zabbix
- Using Telegram bots with Zabbix
- Integrating Atlassian Opsgenie with Zabbix
- Let's get started!

Technical requirements

For this chapter, we are going to need our Zabbix server, preferably how we set it up throughout this book, though any Zabbix 5 server with some alerts on it will do.

We will also need access to a few external services, as follows:

- Slack (free, to an extent)
- Microsoft Teams (free, to an extent)
- Opsgenie (free, to an extent)
- Telegram (free)

We will not cover how to set up the services themselves, but how to integrate them with Zabbix. Make sure you have set up the required service by following a guide and that you have some knowledge of the services in general.

Setting up Slack alerting with Zabbix

Slack is a widely used tool for easy text messaging, voice/video chat, and collaboration. In this recipe, we will learn how to use Zabbix Slack integration to send our Zabbix problem information to Slack so that we can gain a good overview of issues.

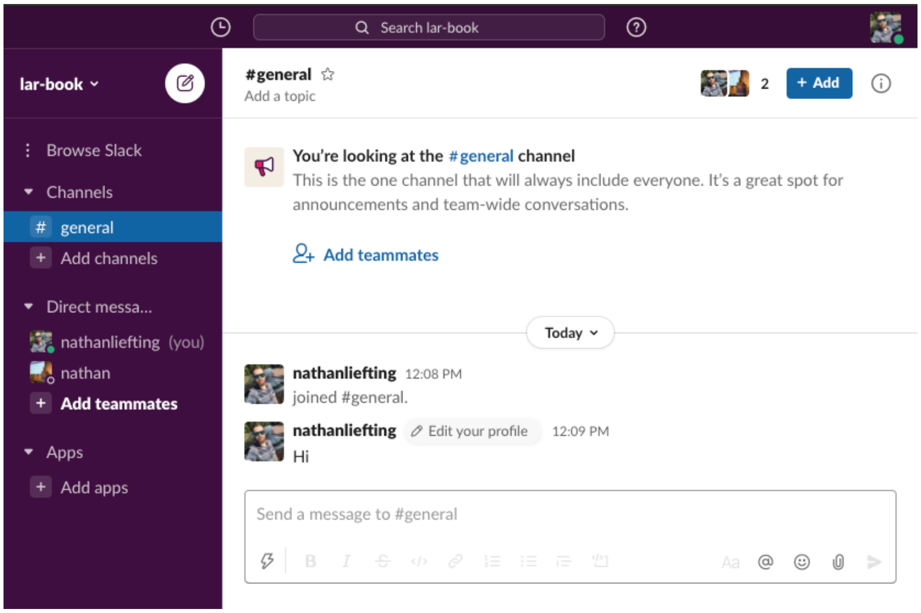
Getting ready

Make sure you have Slack set up. You can go to `slack.com` and set it up for free there. We will also need a Zabbix Server with some active problems.

How to do it…

Follow these steps to complete this recipe:

1. Once you have set up and opened Slack, you should see the following page:



You're looking at the #general channel
This is the one channel that will always include everyone. It's a great spot for announcements and team-wide conversations.

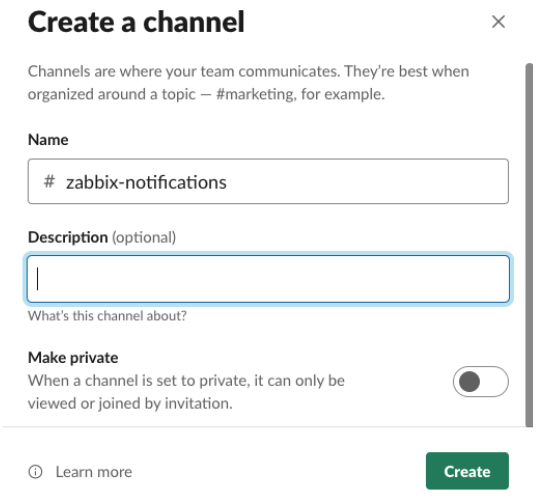
nathanliefting 12:08 PM
joined #general.

nathanliefting 12:09 PM
Hi

Send a message to #general

Figure 8.1 – Slack's default page

2. Let's create a new channel for our Zabbix notifications by clicking the **+Add channels** button. Then, from the dropdown that appears, click **Create a new channel**:



Create a channel

Channels are where your team communicates. They're best when organized around a topic — #marketing, for example.

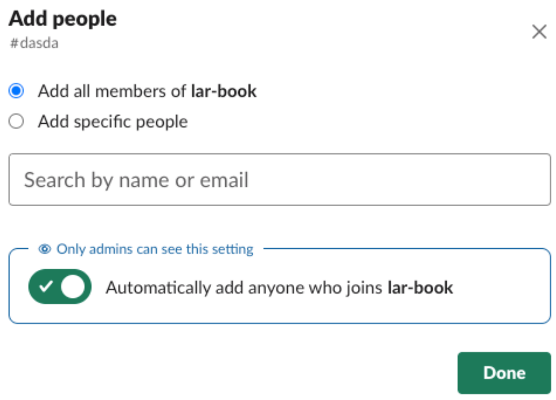
Name: # zabbix-notifications

Description (optional):
What's this channel about?

Make private
When a channel is set to private, it can only be viewed or joined by invitation.

Figure 8.2 – Slack – Create a channel window

3. Press the green **Create** button. Then, on the next window, click the green **Done** button to add all members:


Add people

#dasda

● Add all members of **lar-book**

○ Add specific people

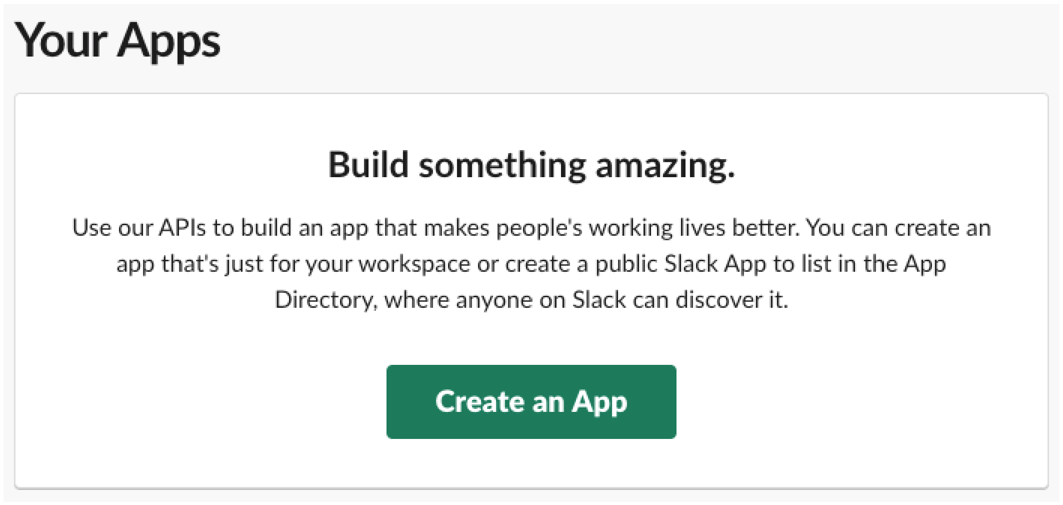
Search by name or email

Only admins can see this setting

✓ Automatically add anyone who joins **lar-book**

Figure 8.3 – Slack – Add people window

4. Now, navigate to the following link to create a Slack Bot for working with Zabbix: `https://api.slack.com/apps`.
5. You will see the option to **Create an App** on this page. Click the **Create an App** button:

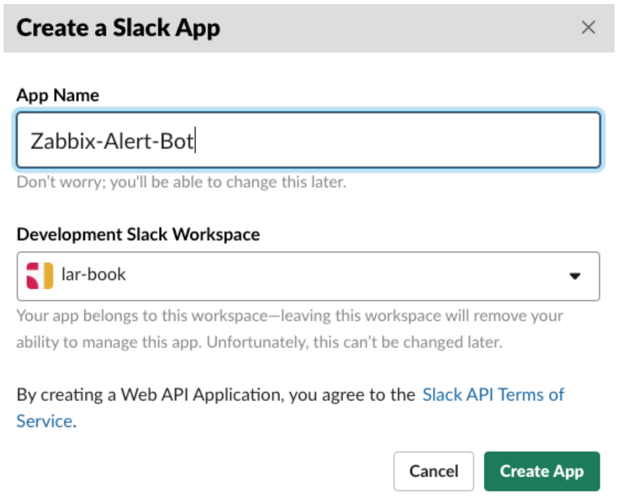

Your Apps

Build something amazing.

Use our APIs to build an app that makes people's working lives better. You can create an app that's just for your workspace or create a public Slack App to list in the App Directory, where anyone on Slack can discover it.

Figure 8.4 – Slack API – Your Apps page

6. After clicking the **Create an App** button, you'll see a pop-up window where you can set up our new Slack Bot:



Create a Slack App

App Name

Zabbix-Alert-Bot

Don't worry; you'll be able to change this later.

Development Slack Workspace

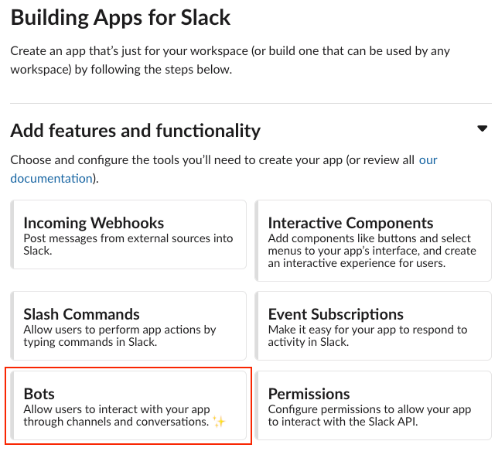
lar-book

Your app belongs to this workspace—leaving this workspace will remove your ability to manage this app. Unfortunately, this can't be changed later.

By creating a Web API Application, you agree to the Slack API Terms of Service.

Figure 8.5 – Slack API – Create a Slack App window

7. Click on **Create App**. This will take you to the **Add features and functionality** page. On this page, click on **Bots**, as highlighted in the following screenshot:



Building Apps for Slack

Create an app that's just for your workspace (or build one that can be used by any workspace) by following the steps below.

Add features and functionality

Choose and configure the tools you'll need to create your app (or review all our documentation).

Incoming Webhooks
Post messages from external sources into Slack.

Interactive Components
Add components like buttons and select menus to your app's interface, and create an interactive experience for users.

Slash Commands
Allow users to perform app actions by typing commands in Slack.

Event Subscriptions
Make it easy for your app to respond to activity in Slack.

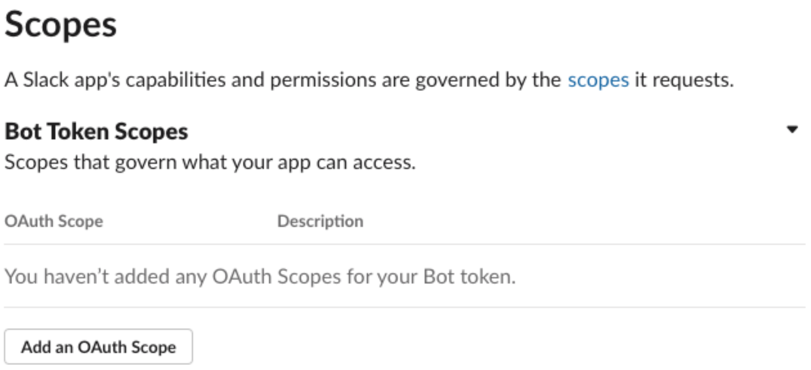
Bots
Allow users to interact with your app through channels and conversations.

Permissions
Configure permissions to allow your app to interact with the Slack API.

Figure 8.6 – Slack API – Add features and functionality page

8. This will take you to the new app's **Home** page. On the left-hand side of the page, click the **OAuth & Permissions** link.

9. Scroll down to **Scopes** and click on **Add an OAuth Scope**:



Scopes

A Slack app's capabilities and permissions are governed by the scopes it requests.

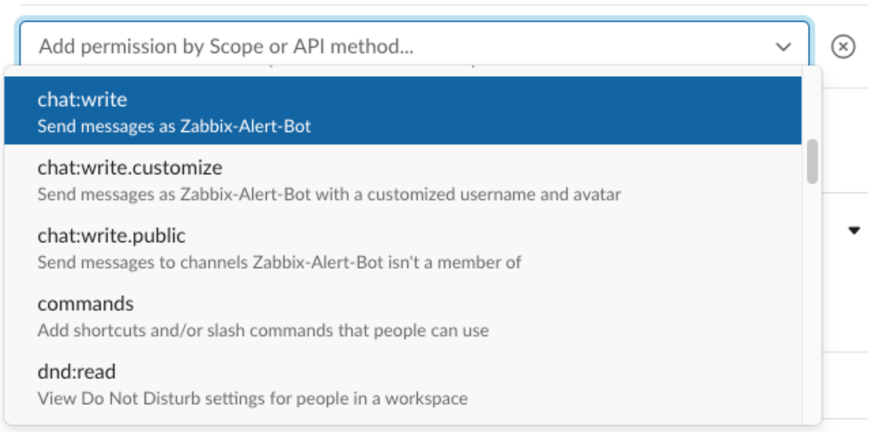
Bot Token Scopes

Scopes that govern what your app can access.

OAuth Scope	Description
You haven't added any OAuth Scopes for your Bot token.	

Figure 8.7 – Slack API – Scopes

10. From the drop - down menu, click on **chat:write** to allow our bot to write to a channel:



chat:write
Send messages as Zabbix-Alert-Bot

chat:write.customize
Send messages as Zabbix-Alert-Bot with a customized username and avatar

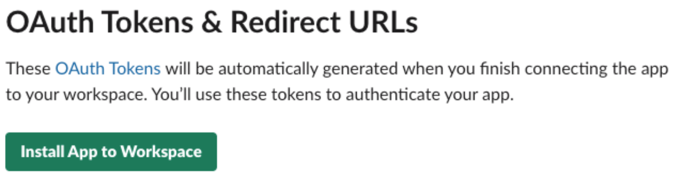
chat:write.public
Send messages to channels Zabbix-Alert-Bot isn't a member of

commands
Add shortcuts and/or slash commands that people can use

dnd:read
View Do Not Disturb settings for people in a workspace

Figure 8.8 – Slack API – Add OAuth scope dropdown

11. Scroll back up and click on the green **Install App to Workspace** button to finish setting up this app:

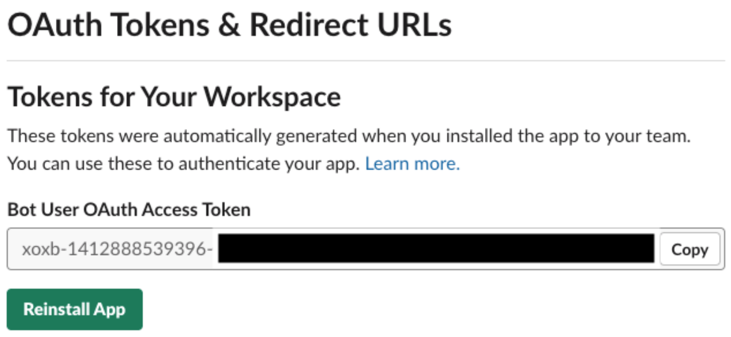


OAuth Tokens & Redirect URLs

These OAuth Tokens will be automatically generated when you finish connecting the app to your workspace. You'll use these tokens to authenticate your app.

Figure 8.9 – Slack API – Install App to Workspace button

12. Next, you will see a pop-up message. Click the green **Allow** button that appears.
13. After clicking **Allow**, you will see your new token. Copy the token by clicking the gray **Copy** button:


OAuth Tokens & Redirect URLs

Tokens for Your Workspace

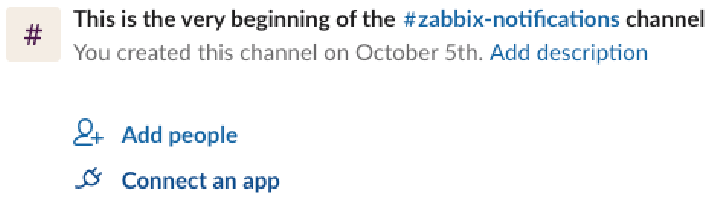
These tokens were automatically generated when you installed the app to your team. You can use these to authenticate your app. Learn more.

Bot User OAuth Access Token

xoxb-1412888539396-[redacted]

Figure 8.10 – Slack API – Our new Bot User OAuth Access Token

14. Lastly, add your bot to the **#zabbix-notifications** channel by going back to your Slack channel and clicking **Connect an app**:


This is the very beginning of the #zabbix-notifications channel

You created this channel on October 5th. Add description

Figure 8.11 – Slack – Connect an app

15. Simply add **Zabbix-Alert-Bot** by clicking the white **Add** button:

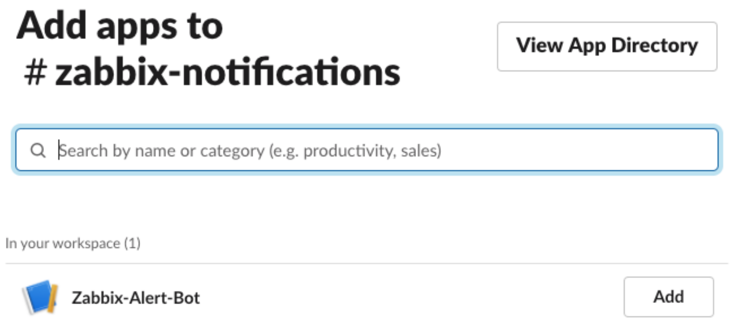
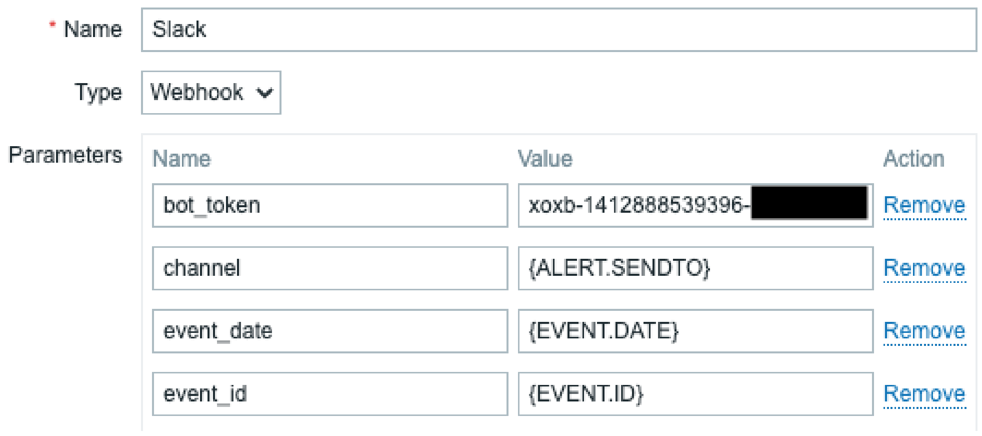

Add apps to # zabbix-notifications

Figure 8.12 – Slack – Connect an app with bot

16. Now, navigate to your Zabbix frontend and go to **Administration** | **Media types**.
17. Click on **Slack** to edit the Slack media type. You will see a whole list of preconfigured parameters. We need to paste our OAuth token into the **bot_token** parameter, like this:



* Name: Slack

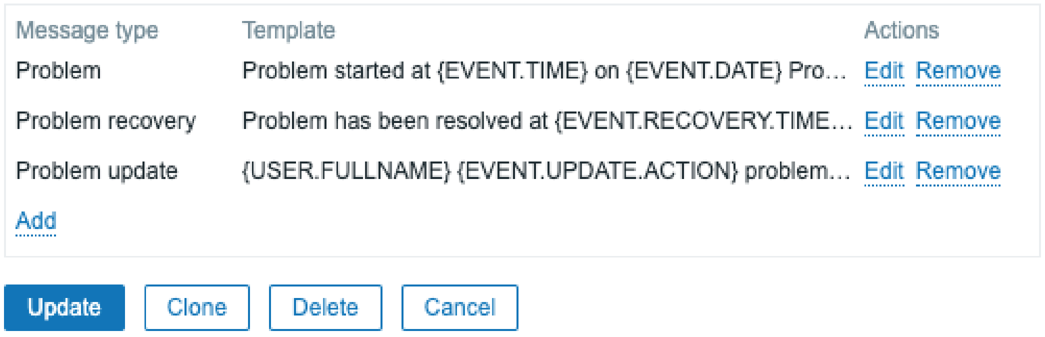
Type: Webhook

Parameters

Name	Value	Action
bot_token	xoxb-1412888539396-[redacted]	Remove
channel	{ALERT.SENDTO}	Remove
event_date	{EVENT.DATE}	Remove
event_id	{EVENT.ID}	Remove

Figure 8.13 – Zabbix Slack media type – Edit page

18. Go to the **Message templates** tab and add the following three **Message types**. They are filled with a template by default when we create them using the underlined **Add** text:

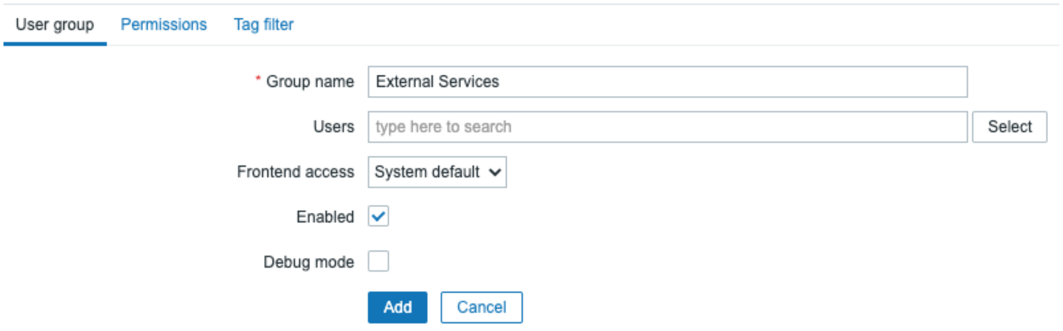


Message type	Template	Actions
Problem	Problem started at {EVENT.TIME} on {EVENT.DATE} Pro…	Edit Remove
Problem recovery	Problem has been resolved at {EVENT.RECOVERY.TIME…	Edit Remove
Problem update	{USER.FULLNAME} {EVENT.UPDATE.ACTION} problem…	Edit Remove

Figure 8.14 – Zabbix media type Slack – Edit page for Message types

19. Click on the blue **Update** button at the bottom of the page to finish editing the Slack media type.

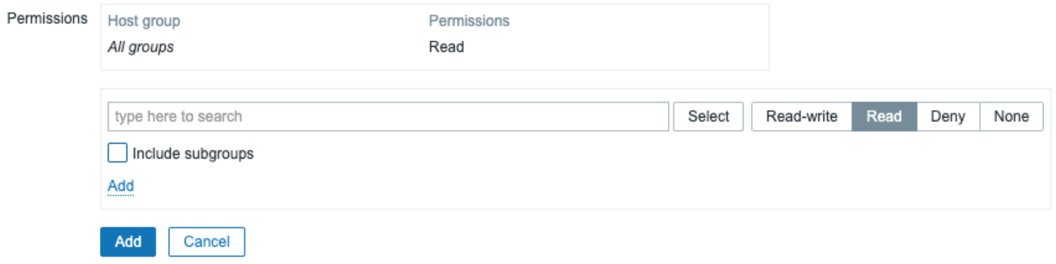
20. Now, let's create a new user group for our media types by navigating to **Administration** | **User groups** and clicking the **Create user group** button in the top-right corner. Add the following user group:



* Group name	External Services
Users	type here to search
Frontend access	System default
Enabled	✓
Debug mode	

Figure 8.15 – Zabbix – Create user group page for the External Services group

21. Click on the **Permissions** tab and then click on **Select**. Make sure that you select all the groups and subgroups with **Read** permissions, like so:


Permissions

Host group	Permissions
All groups	Read

Figure 8.16 – Zabbix – Create user group permissions page for the External Services group

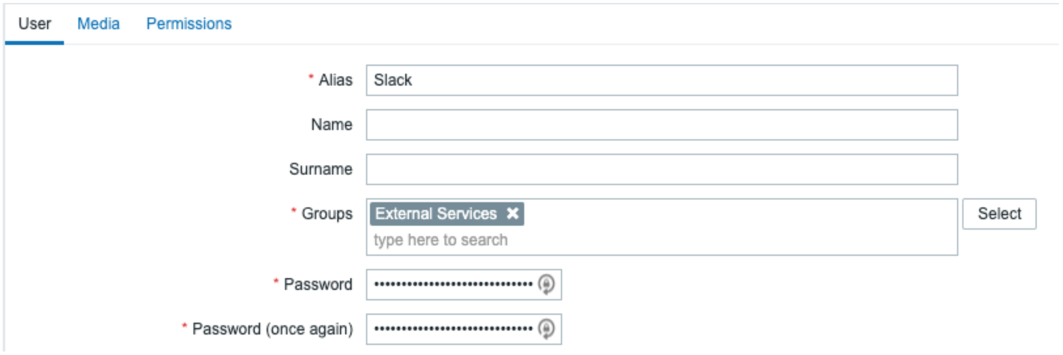
Important note

When applying permissions to the user group, make sure that you only add the host groups you want to receive notifications from. In my lab and even some production environments, I add all groups, but sometimes, we want to filter the notifications down. A great way to do this is to use different user groups and users so that you only receive notifications from host groups in certain channels.

22. Now, click on the blue **Add** button and finish creating this user group.

23. Next, navigate to **Administration** | **Users** to create a Slack user. Click on the blue **Create user** button in the top-right corner.

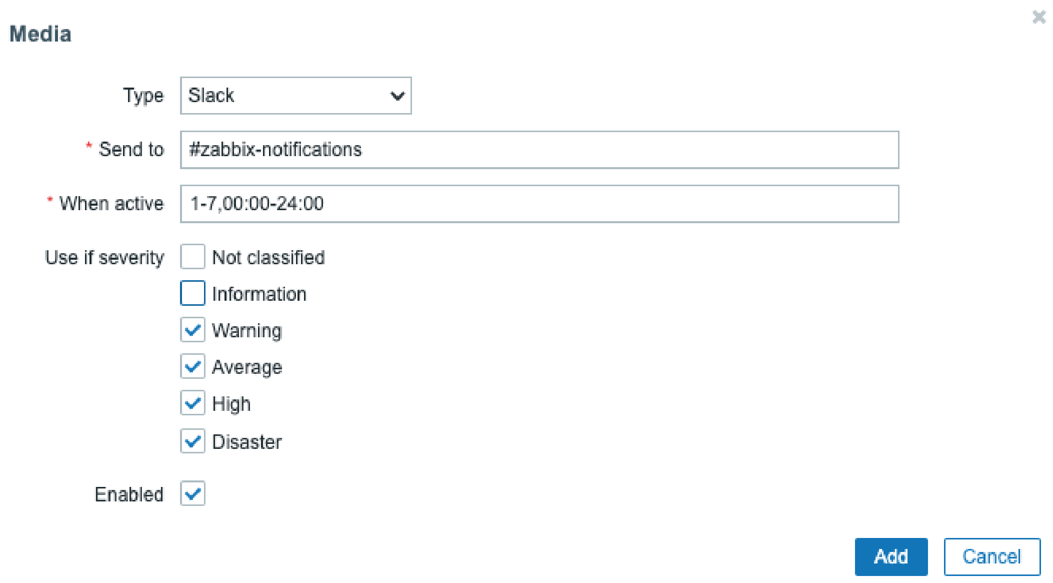
24. Add the following user to your Zabbix Server:



* Alias	Slack
Name	
Surname	
* Groups	External Services ✖ type here to search
* Password	••••••••••••••••••••••••••••
* Password (once again)	••••••••••••••••••••••••••••

Figure 8.17 – Zabbix – Create user page, user Slack

25. Now, click on the **Media** tab and click on the underlined **Add** text. We will add the following media to this user:

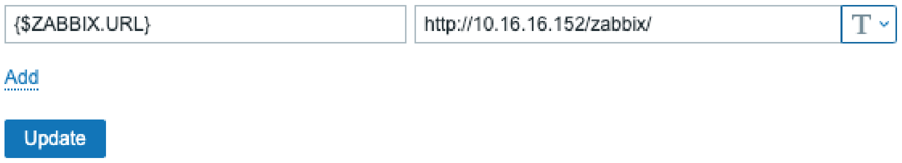


Type	Slack
* Send to	#zabbix-notifications
* When active	1-7,00:00-24:00
Use if severity	☐ Not classified
	☐ Information
	✓ Warning
	✓ Average
	✓ High
	✓ Disaster
Enabled	✓

Figure 8.18 – Zabbix – Create user media page, user Slack

26. Click on the blue **Add** button at the bottom of the window and then on the blue **Add** button at the bottom of the page.

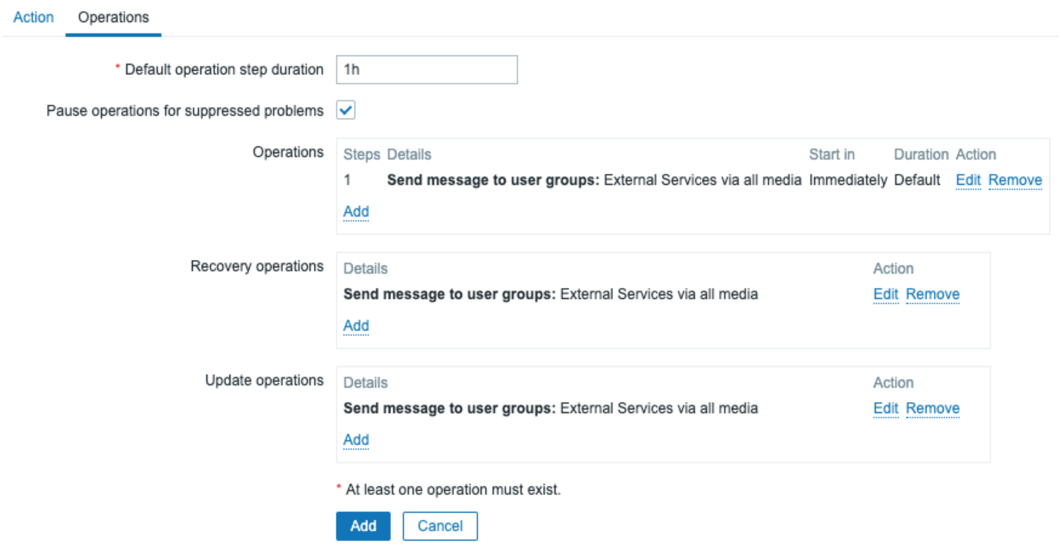
27. We will also need to add a macro to **Administration | General**. Use the drop-down menu to select **Macros** and add the following macro, which contains your Zabbix URL:



{$ZABBIX.URL}	http://10.16.16.152/zabbix/

Figure 8.19 – Zabbix – Administration | General macros page with Zabbix URL

28. Click on the blue **Update** button. Last but not least, go to **Administration | Actions** and on the **Trigger Actions** page, click on the blue **Create action** button.
29. Use **Notify external services** as the name of your action and go to **Operations**. Add the following operations:



* Default operation step duration: 1h

Pause operations for suppressed problems: ✓

Operations

Steps	Details	Start in	Duration	Action
1	**Send message to user groups:** External Services via all media	Immediately	Default	Edit Remove

Recovery operations

Details	Action
Send message to user groups: External Services via all media	Edit Remove

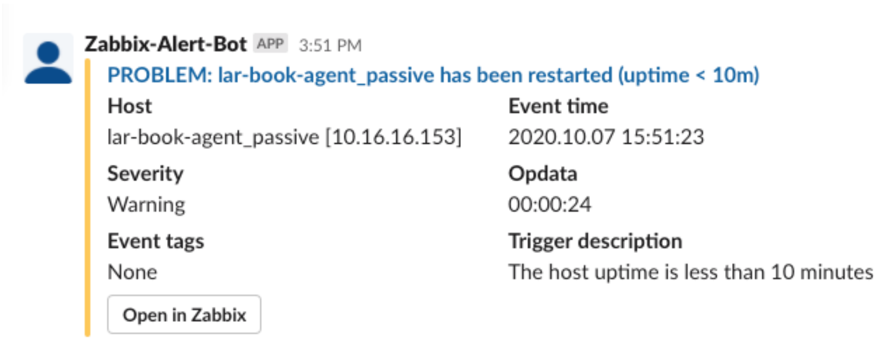
Update operations

Details	Action
Send message to user groups: External Services via all media	Edit Remove

* At least one operation must exist.

Figure 8.20 – Zabbix – Create action operations page for Notify external services

30. Now, click on the blue **Add** button. With that, you're done. You can now view new problems in your Slack channel:


Zabbix-Alert-Bot APP 3:51 PM

PROBLEM: lar-book-agent_passive has been restarted (uptime < 10m)

Host	**Event time**
lar-book-agent_passive [10.16.16.153]	2020.10.07 15:51:23
Severity	**Opdata**
Warning	00:00:24
Event tags	**Trigger description**
None	The host uptime is less than 10 minutes

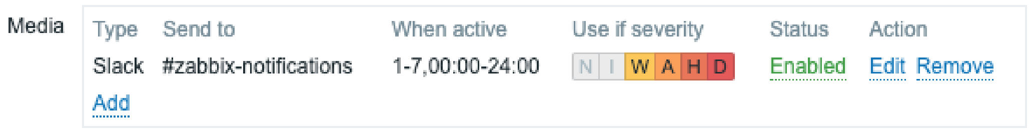
Figure 8.21 – Slack – Zabbix notification sent from our Zabbix Server

How it works...

Working with media types might be something completely new to you, or you might have done it in the past. Regardless, in Zabbix 5, the process has changed a bit. In the days of Zabbix before Zabbix 5, we had to find the right media types online or make them ourselves.

Now, with Zabbix 5, we get a preconfigured media type ready to go. We just need to do the necessary setup and fill in the right information, just like we just did for Slack. We are then ready to send our problem-related information from Zabbix 5 to Slack every time a Media Type is created.

In this recipe, we told our Zabbix Server to only send problem-related information with a severity of Warning or higher to Slack, as shown in the following screenshot:


Media

Type	Send to	When active	Use if severity	Status	Action
Slack	#zabbix-notifications	1-7,00:00-24:00	N I W A H D	Enabled	Edit Remove

Figure 8.22 – Zabbix Media page for Slack user

We can fully customize these severities, but we can also fully customize what severities we send to our Slack setup.

What we configured in this recipe is called a Zabbix webhook. A problem gets created in Zabbix and this problem matches our configured criteria, such as its severity. Our media type gets triggered and sends it to the configured links:

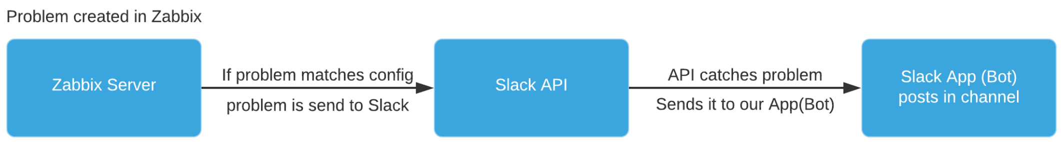


Figure 8.23 – Zabbix Slack integration diagram

Zabbix sends the problem to the Slack API, and then the API processes it has configured there. Then, the app we configured in Slack posts the problem to our channel.

Now that we've completed this recipe, we can view problems in Slack and keep an eye on our Zabbix alerts from there.

See also

If you want to do more with this integration, check out the Slack API documentation. There's a lot we can do with this API, and we can build a number of awesome apps/bots for our channels: `https://api.slack.com/`.

Setting up Microsoft Teams alerting with Zabbix

At the time of writing this book, I'm living in The Netherlands and in the COVID-19 crisis. A lot of IT companies have been requested to make their employees work from home. Due to this, there's been a rise in the use of Microsoft Teams and applications like it. Suddenly, a lot of companies have started using Microsoft Teams and others to make working from home and collaboration easier.

Let's learn how we can make working with Microsoft Teams even better by integrating our Zabbix alerting into it.

Getting ready

We will need our Zabbix Server to be able to create some problems for us. For this, you can use `lar-book-centos` from our previous chapters or any Zabbix Server that you prefer.

We will also need general Microsoft Teams knowledge and, of course, Microsoft Teams itself set up and ready to go.

How to do it…

Follow these steps to complete this recipe:

1. Let's start by opening our Microsoft Teams application on either Windows, Mac, or Linux and creating a new channel. Go to **Teams** and click on the three dots (**…**) next to your team name, as shown in the following screenshot:

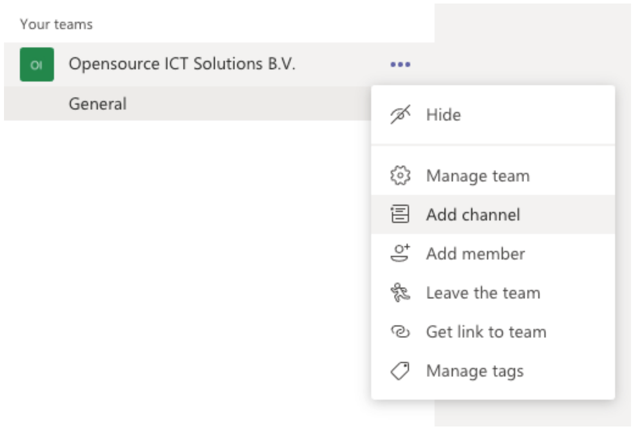
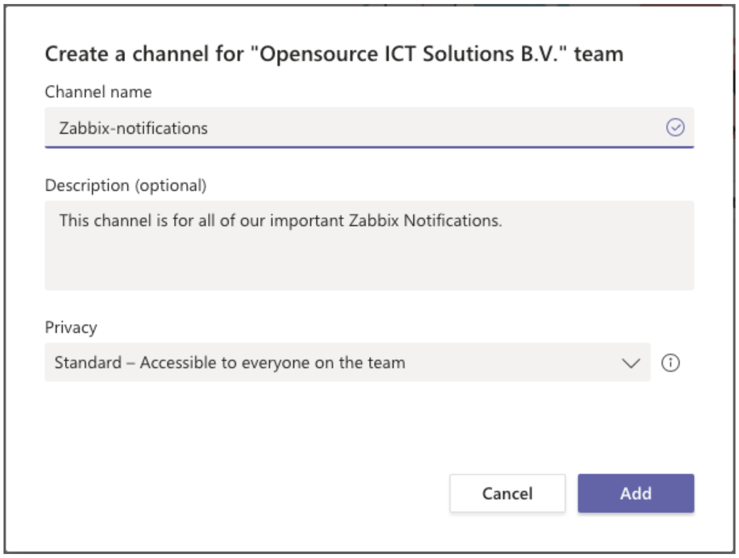


Figure 8.24 – MS Teams – Add channel option

2. In the **Add channel** window, fill in the following information to create our new channel:



Create a channel for "Opensource ICT Solutions B.V." team

Channel name: Zabbix-notifications

Description (optional): This channel is for all of our important Zabbix Notifications.

Privacy: Standard – Accessible to everyone on the team

Figure 8.25 – MS Teams – Add channel window

3. Now, click the purple **Add** button to add the channel. Upon doing this, we will be able to see our new channel in the list.
4. Click on the three dots (**...**) next to your channel, as shown in the following screenshot:

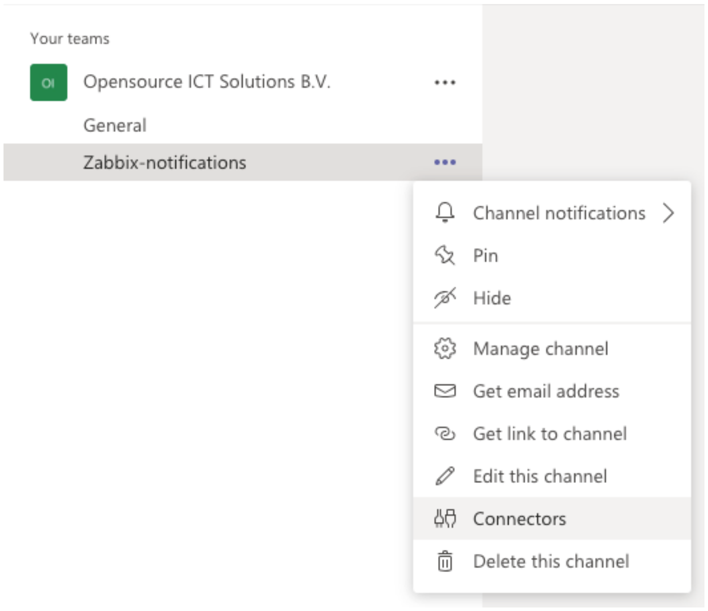
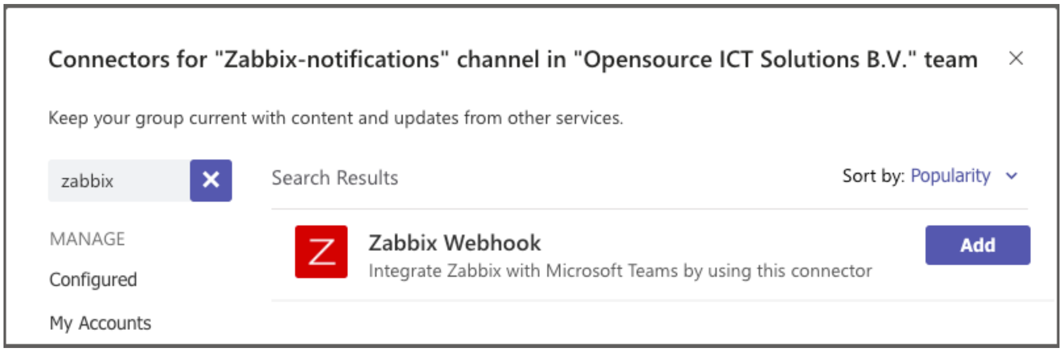


Figure 8.26 – MS Teams – Add Connectors option

5. We want to select the **Connectors** options from this drop-down menu. This allows us to add our Microsoft Teams connector to this channel.
6. We are using the search field here to find the official **Zabbix Webhook** connector:


Connectors for "Zabbix-notifications" channel in "Opensource ICT Solutions B.V." team

Keep your group current with content and updates from other services.

zabbix

Search Results

Sort by: Popularity

MANAGE

Configured

My Accounts

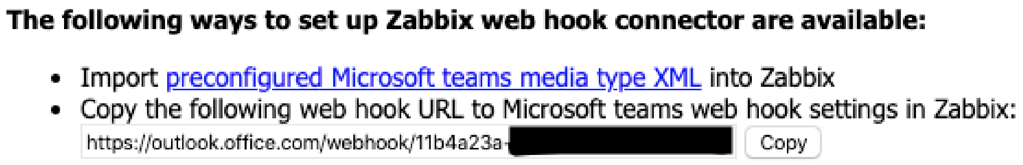
Zabbix Webhook

Integrate Zabbix with Microsoft Teams by using this connector

Add

Figure 8.27 – MS Teams – Add Connectors window

7. Click on the purple **Add** button next to the **Zabbix Webhook** connector to add this connector to our channel. This will open another pop-up window, where we must press the purple **Add** button again.
8. You will get another pop-up window, where you need to copy the webhook URL. Do this by pressing the white **Copy** button:



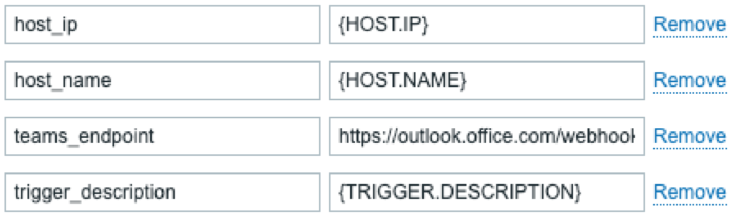
The following ways to set up Zabbix web hook connector are available:

- Import preconfigured Microsoft teams media type XML into Zabbix
- Copy the following web hook URL to Microsoft teams web hook settings in Zabbix:

https://outlook.office.com/webhook/11b4a23a-[redacted]

Figure 8.28 – MS Teams – Webhook URL

9. Now, press the purple **Save** button. Upon doing this, you can close the pop-up window.
10. Go to the Zabbix frontend and navigate to **Administration** | **Media types**. Click on the **MS Teams** media type here.
11. Scroll down until you see **teams_endpoint**. Paste the URL you copied previously here, as shown in the following screenshot:

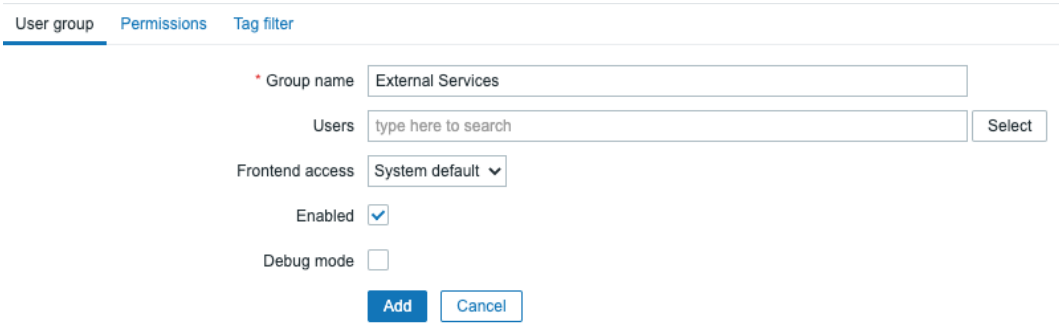


host_ip	{HOST.IP}	Remove
host_name	{HOST.NAME}	Remove
teams_endpoint	https://outlook.office.com/webhool	Remove
trigger_description	{TRIGGER.DESCRIPTION}	Remove

Figure 8.29 – Zabbix Administration | Media types, edit MS Teams page

12. Now, press the blue **Update** button at the bottom of the page.

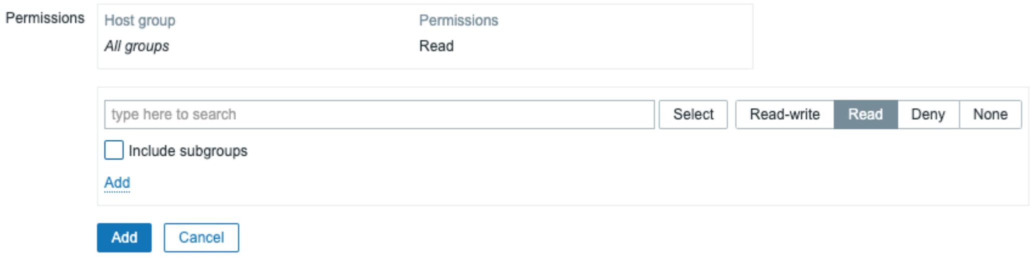
13. If you didn't follow the previous recipe, then create a new user group for our media types by navigating to **Administration** | **User groups** and clicking the **Create user group** button in the top-right corner. Add the following user group:



* Group name	External Services
Users	type here to search
Frontend access	System default
Enabled	✓
Debug mode	

Figure 8.30 – Zabbix, Create user group page, the External Services group

14. Click on the **Permissions** tab and click on **Select**. Make sure that you select all the groups and subgroups with **Read** permissions. The **Permissions** tab will look like this:

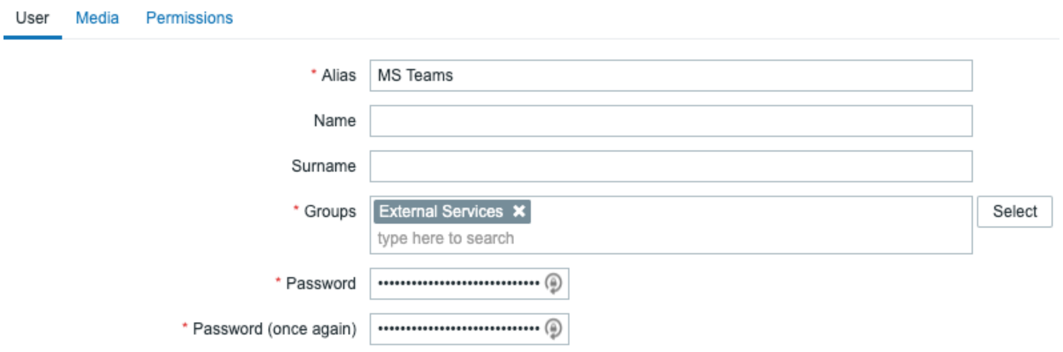


Host group	Permissions
All groups	Read

Figure 8.31 – Zabbix – Create user group permissions page, the External Services group

15. Now, click on the blue **Add** button and finish creating this user group.

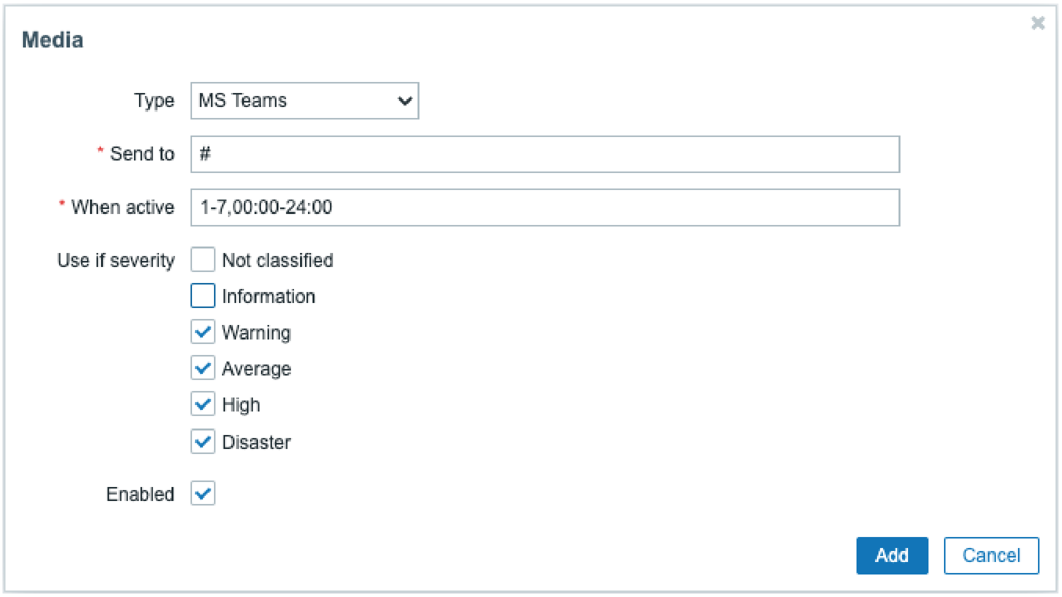
16. Navigate to **Administration** | **Users** and click on the blue **Create user** button in the top-right corner. Add the following user:



* Alias	MS Teams
Name	
Surname	
* Groups	External Services ✕ / type here to search
* Password	••••••••••••••••••••••••••••
* Password (once again)	••••••••••••••••••••••••••••

Figure 8.32 – Zabbix Administration | Media types – Create new user page, MS Teams

17. Next, go to the **Media** tab of the **Create user** page. Click the underlined **Add** text here to create the following media:

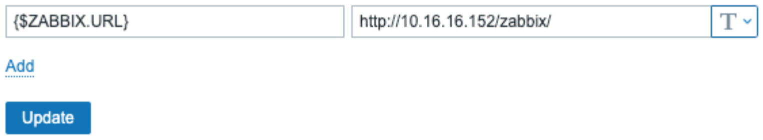


Type	MS Teams
* Send to	#
* When active	1-7,00:00-24:00
Use if severity	☐ Not classified
	☐ Information
	✓ Warning
	✓ Average
	✓ High
	✓ Disaster
Enabled	✓

Figure 8.33 – Zabbix Administration | Media types – Create new user page, MS Teams

18. Once you've filled in this information, click the blue **Add** button at the bottom of the window and then the blue **Add** button at the bottom of the page.

19. If you didn't follow the previous recipe, then you will also need to add a macro to **Administration** | **General**. Use the drop-down menu on this page to select **Macros** and add the following macro, which contains your Zabbix URL:

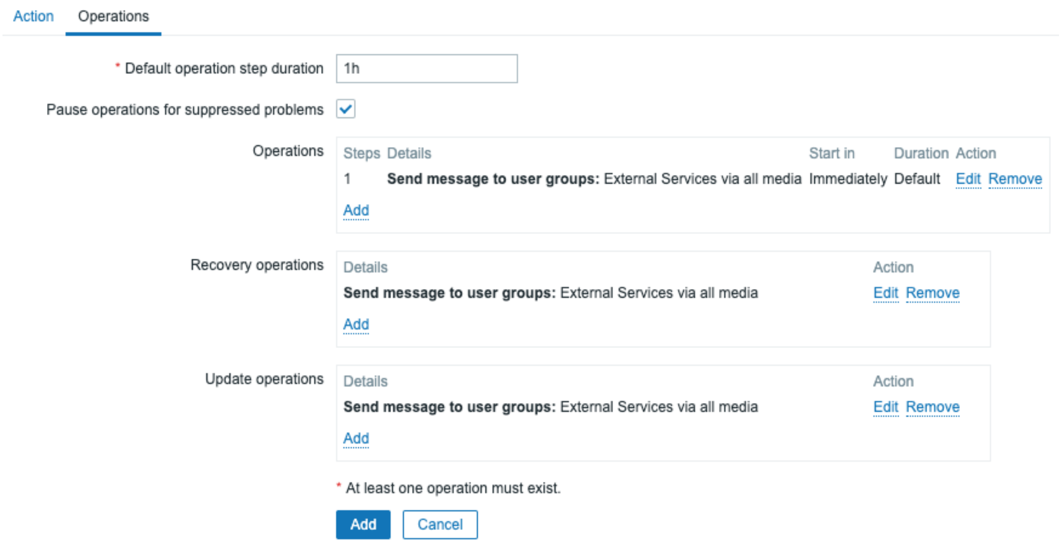


{$ZABBIX.URL}	http://10.16.16.152/zabbix/

Figure 8.34 – Zabbix Administration | General macros page, Zabbix URL for use with MS Teams

20. Click on the blue **Update** button. You will also need to go to **Administration** | **Actions** if you didn't follow the previous recipe and, on the **Trigger Actions** page, click on the blue **Create action** button.

21. Use **Notify external services** for the name of the action and go to **Operations**. Add the following operations:



* Default operation step duration: 1h

Pause operations for suppressed problems: ✓

Operations

Steps	Details	Start in	Duration	Action
1	**Send message to user groups:** External Services via all media	Immediately	Default	Edit Remove

Recovery operations

Details	Action
Send message to user groups: External Services via all media	Edit Remove

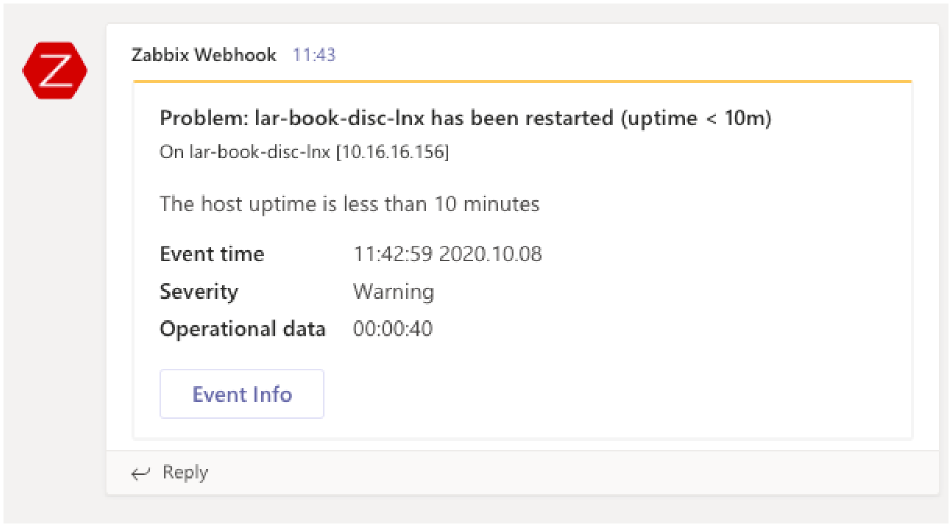
Update operations

Details	Action
Send message to user groups: External Services via all media	Edit Remove

* At least one operation must exist.

Figure 8.35 – Zabbix – Create action operations page, Notify external services for use with MS Teams

22. Now, click on the blue **Add** button and you'll be done. You can now view new problems as they occur in your MS Teams channel:



Zabbix Webhook 11:43

Problem: lar-book-disc-lnx has been restarted (uptime < 10m)

On lar-book-disc-lnx [10.16.16.156]

The host uptime is less than 10 minutes

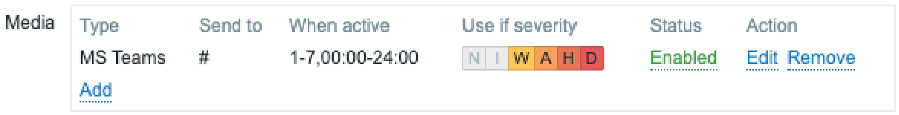
Event time	11:42:59 2020.10.08
Severity	Warning
Operational data	00:00:40

Figure 8.36 – A Zabbix problem in an MS Teams channel

How it works...

Microsoft Teams works in about the same way as our Slack setup. A problem is created in Zabbix Server and if that problem matches our configured requirements in Zabbix, we send that problem to our Microsoft Teams connector.

For instance, we configured Zabbix so that it only sends problems with a severity of Warning or higher to Microsoft Teams, as shown here:



Media

Type	Send to	When active	Use if severity	Status	Action
MS Teams	#	1-7,00:00-24:00	N I W A H D	Enabled	Edit Remove
Add					

Figure 8.37 – Zabbix Media page for MS Teams users

Our Microsoft Teams connector catches our problem and since this connector is configured directly on our channel, it posts a notification to the channel:

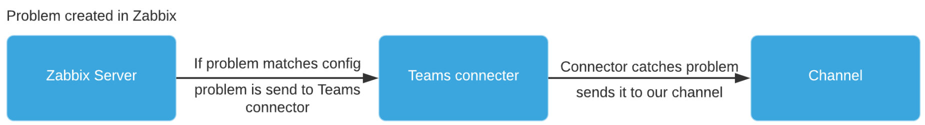

Problem created in Zabbix
Zabbix Server
If problem matches config
problem is send to Teams connector
Teams connecter
Connector catches problem
sends it to our channel
Channel

Figure 8.38 – Zabbix Microsoft Teams integration diagram

Now, we can see our Teams notifications in our channel and keep up to date with all our Zabbix issues directly via Microsoft Teams.

See also

For more information about the Zabbix Webhook connector, check out this page: `https://appsource.microsoft.com/en-us/product/office/WA200001837?tab=Overview`.

Using Telegram bots with Zabbix

If you love automation in chat applications, you might have heard of or used Telegram Messenger. Telegram has an extensive API and amazing bot features.

In this recipe, we are going to use a Telegram bot to create a Telegram group for Zabbix alerts. Let's get started.

Getting ready

Make sure you have your Zabbix server ready. You can use `lar-book-centos` or any Zabbix Server capable of sending some alerts.

It would be useful if you have some knowledge of Telegram, but I'll be describing how to set it up step by step, even for those of you who have never used Telegram bots. Just make sure that you have the Telegram app on your computer and that your account is set up.

How to do it...

Follow these steps to complete this recipe:

1. First, let's create a new channel in Telegram. Click on the **Create** icon next to the search box and select **New Group**:

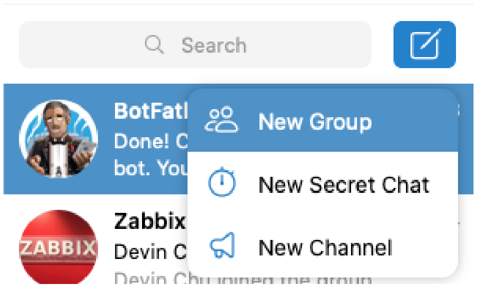


Figure 8.39 – Telegram – New Group button

2. Add another user – someone in your team that needs to get notifications as well. Fill in a group name and add a picture if you want:

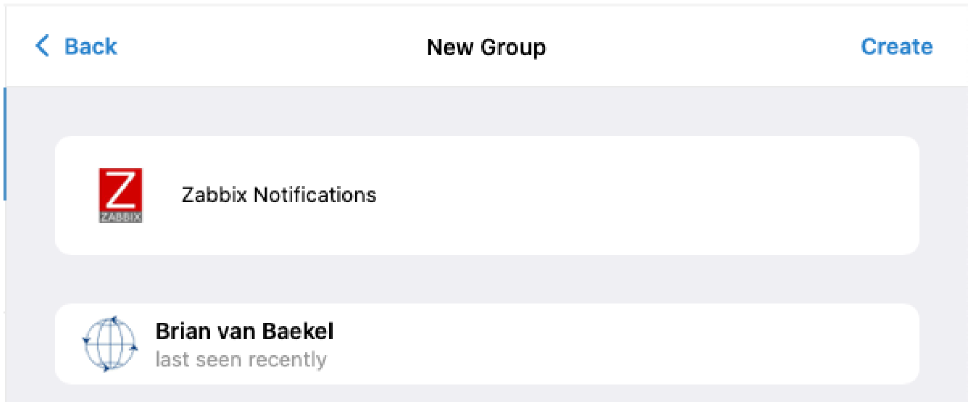


Figure 8.40 – Telegram – New Group page

3. Now, click on the **Create** button in the top-right corner. This will take you to the new groups page.

4. Working with Telegram bots is made easy with the **@BotFather** user on Telegram. We can start creating our bot by searching for `botfather` and clicking the specified contact:

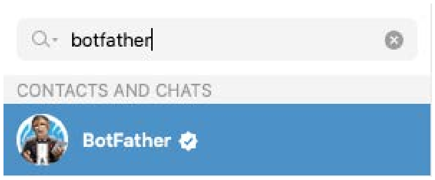
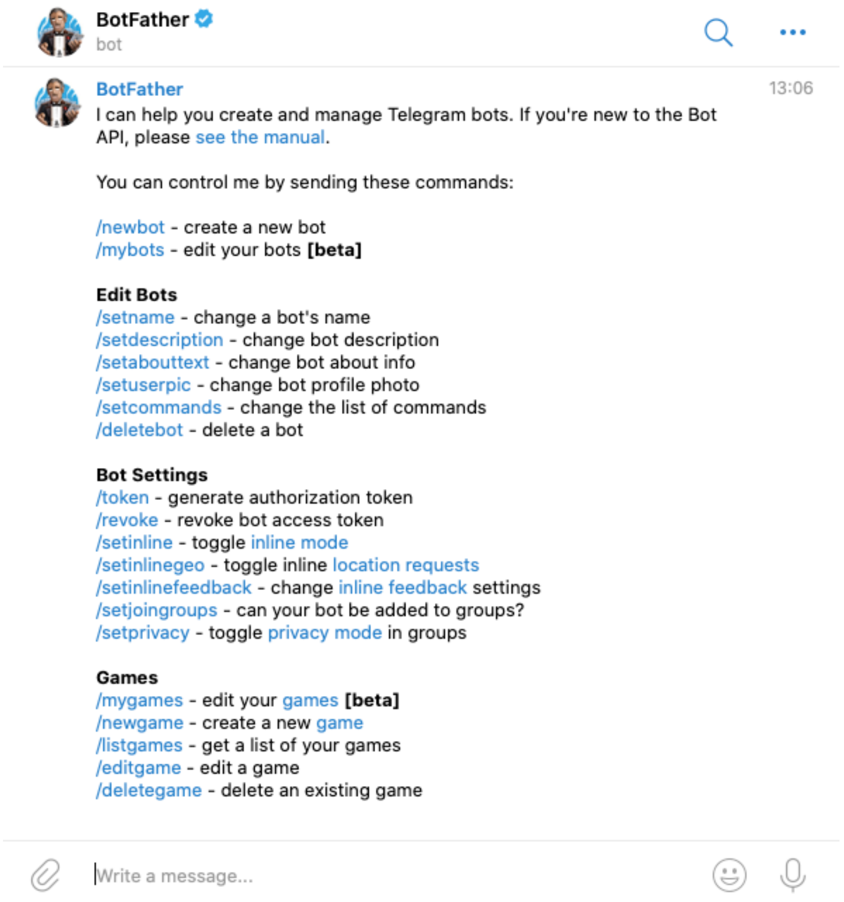


Figure 8.41 – Telegram – BotFather user

5. Let's start by issuing the `/start` command in the chat. This will provide you with a list of commands you can use:



I can help you create and manage Telegram bots. If you're new to the Bot API, please see the manual.

You can control me by sending these commands:

/newbot - create a new bot
/mybots - edit your bots **[beta]**

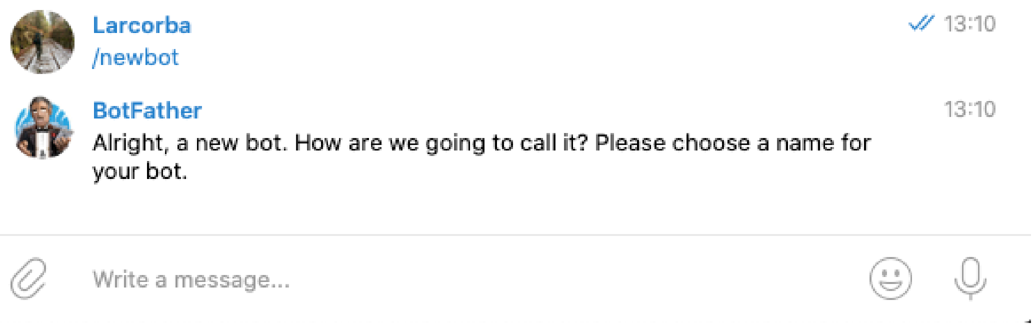
Edit Bots
/setname - change a bot's name
/setdescription - change bot description
/setabouttext - change bot about info
/setuserpic - change bot profile photo
/setcommands - change the list of commands
/deletebot - delete a bot

Bot Settings
/token - generate authorization token
/revoke - revoke bot access token
/setinline - toggle inline mode
/setinlinegeo - toggle inline location requests
/setinlinefeedback - change inline feedback settings
/setjoingroups - can your bot be added to groups?
/setprivacy - toggle privacy mode in groups

Games
/mygames - edit your games **[beta]**
/newgame - create a new game
/listgames - get a list of your games
/editgame - edit a game
/deletegame - delete an existing game

Figure 8.42 – Telegram – BotFather user help list

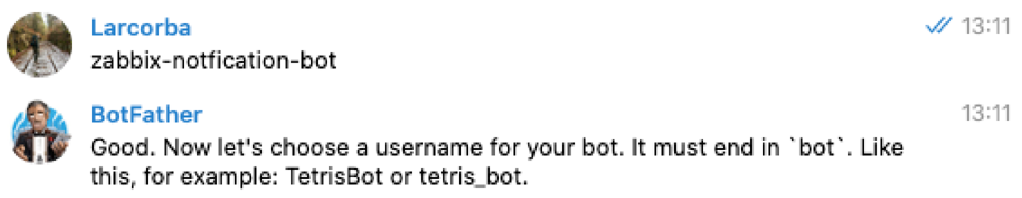
6. Now, let's immediately create a new bot by typing `/newbot` into the **BotFather** chat. Press *Enter* to send your message. This will give us the following result:


Larcorba 13:10
/newbot

BotFather 13:10
Alright, a new bot. How are we going to call it? Please choose a name for your bot.

Figure 8.43 – Telegram – BotFather /newbot command

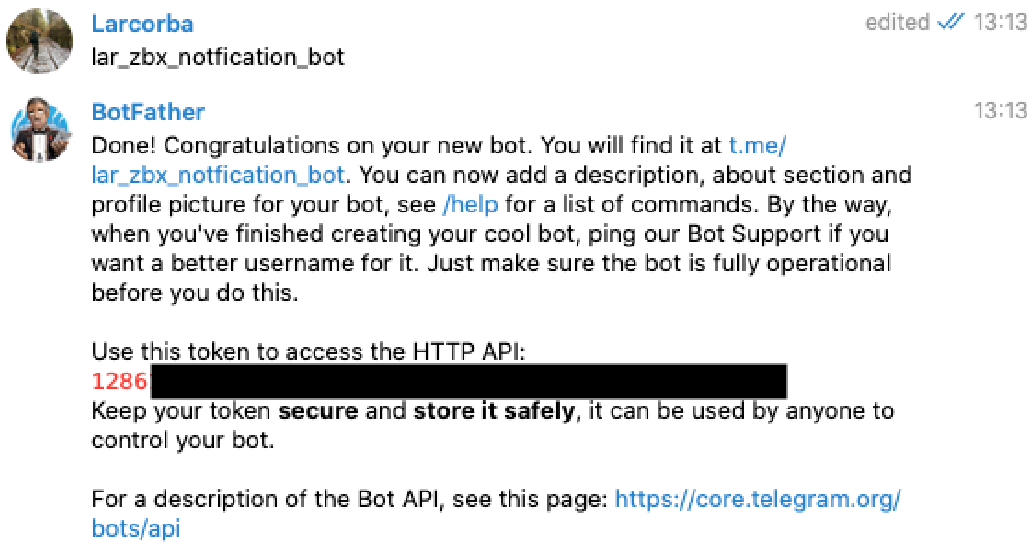
7. Type in the new name of the bot; we will call it `zabbix-notfication-bot`. Press *Enter* to send your name:


Larcorba 13:11
zabbix-notfication-bot

BotFather 13:11
Good. Now let's choose a username for your bot. It must end in `bot`. Like this, for example: TetrisBot or tetris_bot.

Figure 8.44 – Telegram – BotFather bot username

8. You will then be asked what username you want to give the bot. I will use `lar_zbx_notfication_bot`:



Larcorba edited 13:13
lar_zbx_notfication_bot

BotFather 13:13
Done! Congratulations on your new bot. You will find it at t.me/lar_zbx_notfication_bot. You can now add a description, about section and profile picture for your bot, see /help for a list of commands. By the way, when you've finished creating your cool bot, ping our Bot Support if you want a better username for it. Just make sure the bot is fully operational before you do this.

Use this token to access the HTTP API:
1286[redacted]
Keep your token **secure** and **store it safely**, it can be used by anyone to control your bot.

For a description of the Bot API, see this page: https://core.telegram.org/bots/api

Figure 8.45 – Telegram – BotFather bot token

Important note

Your bot username must be unique, so you can use `lar_zbx_notification_bot` here. Pick a unique bot name that suits yourself and use that name throughout this recipe.

9. Make sure that you save the **HTTP API** key somewhere safe.
10. Let's go back to our **Zabbix Notifications** group and add our bot. Click on the group in your list of chats and click on the group's name.

11. Now, click on **Add** to add the bot, as follows:

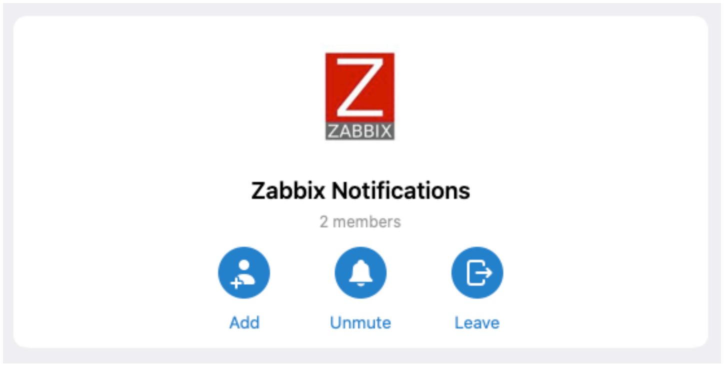


Figure 8.46 – Telegram – Add user to group button

12. Next, you will need to search for your bot using its username, as shown in the following screenshot:

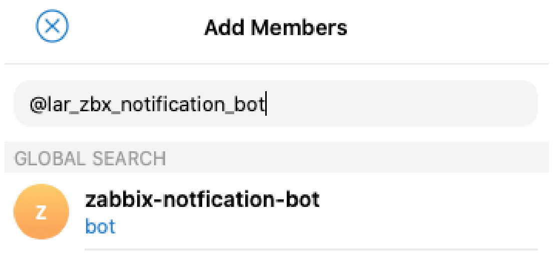
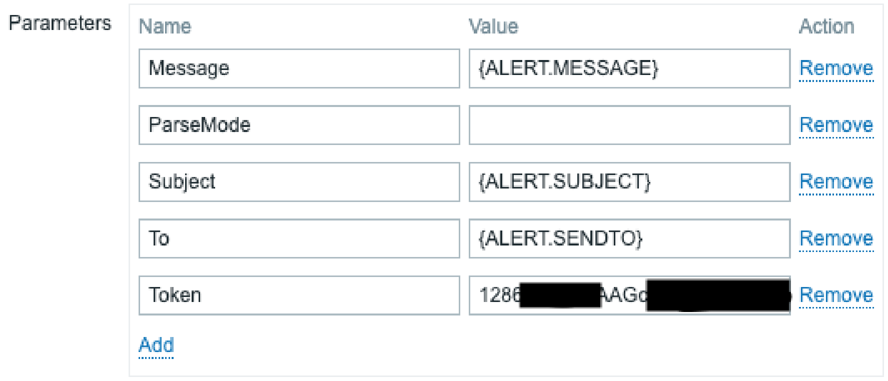


Figure 8.47 – Telegram – Add user to group page

13. Click on the bot and click on the **Add** button. With that, your bot has been added to the channel.
14. Let's navigate to the Zabbix frontend and go to **Administration** | **Media types**. Click on the media type titled **Telegram**.

15. Here, you must add the **HTTP API** key you generated earlier to the **Token** field of our media type:



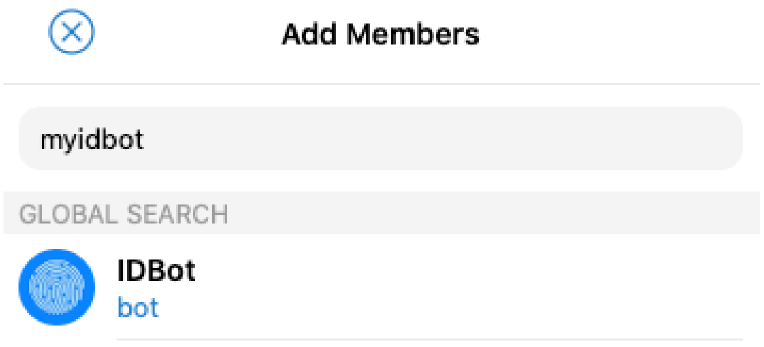
Parameters

Name	Value	Action
Message	{ALERT.MESSAGE}	Remove
ParseMode		Remove
Subject	{ALERT.SUBJECT}	Remove
To	{ALERT.SENDTO}	Remove
Token	1286[redacted]AGc[redacted]	Remove

Add

Figure 8.48 – Zabbix Administration | Media types – Edit Telegram Media type page

16. Click on the blue **Update** button at the bottom of the page to finish editing the **Telegram** media type.
17. Now, let's go back to Telegram and add another bot to our group. Go to our new group and click on the group's name. Click on **Add** to add the following user:



Add Members

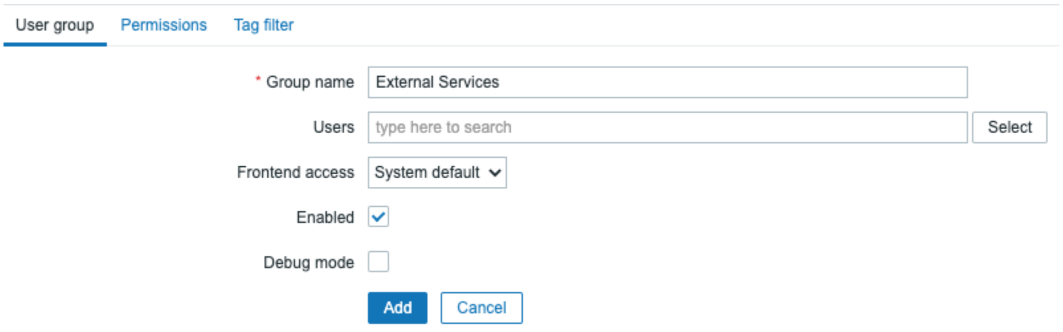
myidbot

GLOBAL SEARCH

IDBot
bot

Figure 8.49 – Add user page for a Telegram group

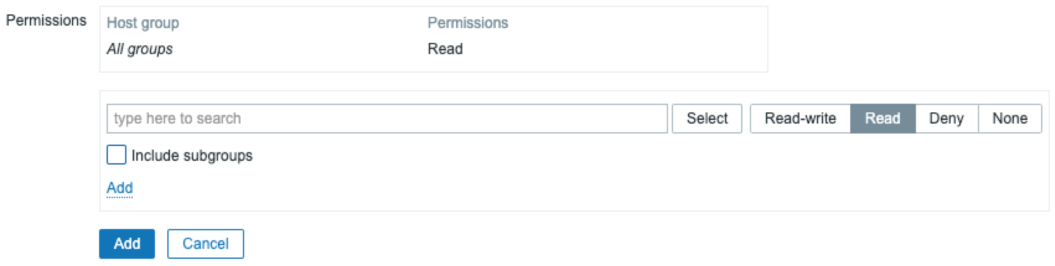
18. Click on the user and click on **Add**. Then, navigate back to the Zabbix frontend.
19. If you haven't followed any of the preceding recipes, create a new user group for our media types by navigating to **Administration** | **User groups** and clicking the **Create user group** button in the top-right corner. Add the following user group:



* Group name	External Services
Users	type here to search [Select]
Frontend access	System default
Enabled	✓
Debug mode	

Figure 8.50 – Zabbix – Create user group page, External Services for use with Telegram

20. Click on the **Permissions** tab and click on **Select**. Make sure that you select all the groups and subgroups with **Read** permissions, as follows:


Permissions

Host group	Permissions
All groups	Read

Figure 8.51 – Zabbix – Create user group permissions page, External Services for use with Telegram

21. Now, click on the blue **Add** button and finish creating this user group.

22. At this point, you must create a new user in Zabbix. However, to create this user, you are going to need our new group ID. Go back to Telegram and issue `/getgroupid@myidbot` in the group chat. You will receive a value that you will need to copy:

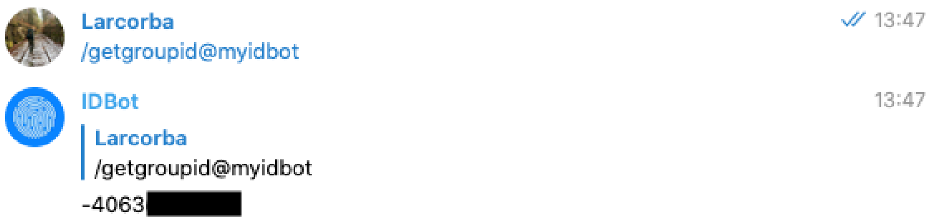
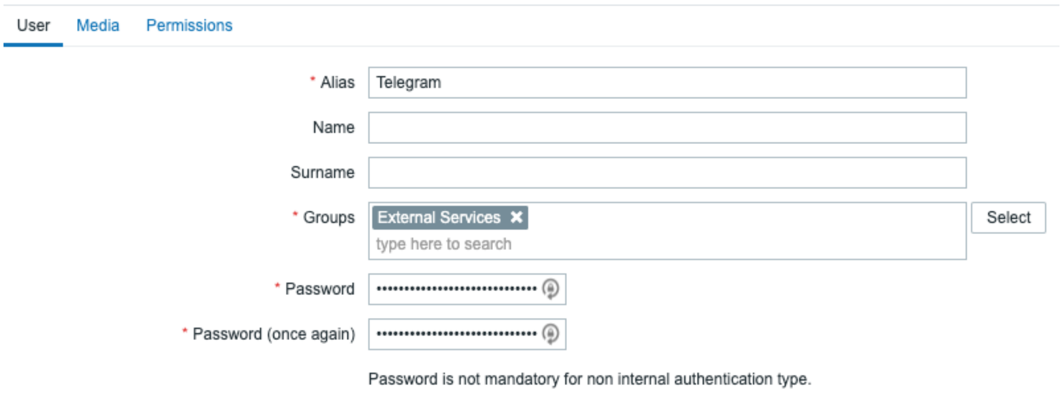

Larcorba
/getgroupid@myidbot
13:47
IDBot
13:47
Larcorba
/getgroupid@myidbot
-4063[redacted]

Figure 8.52 – Telegram user group ID

23. Let's navigate to **Administration** | **Users** and click the blue **Create user** button. Add the following user:

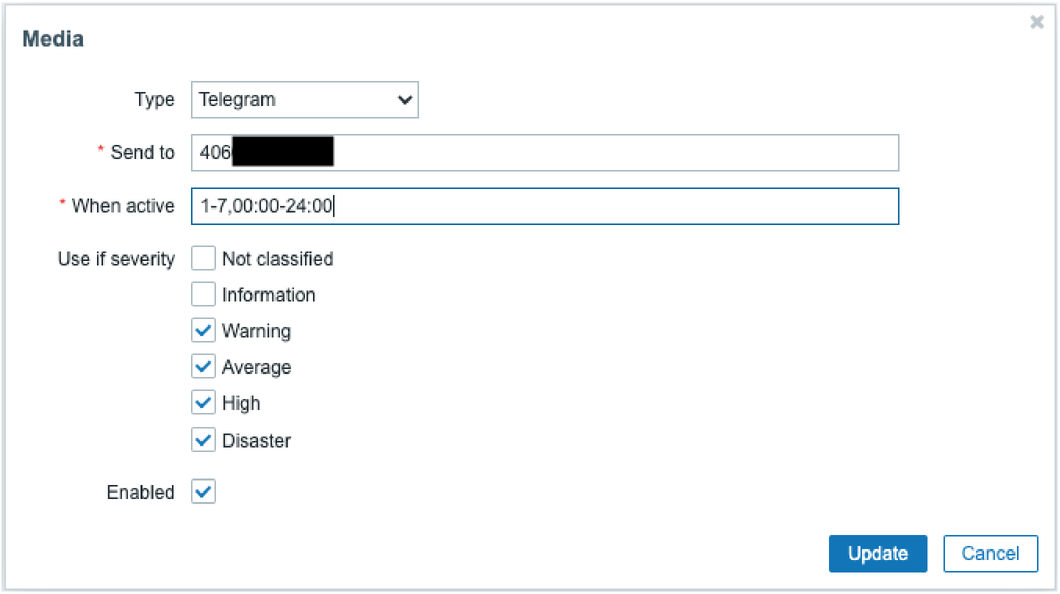


* Alias	Telegram
Name	
Surname	
* Groups	External Services ✕ type here to search
* Password	••••••••••••••••••••••••••••••
* Password (once again)	••••••••••••••••••••••••••••••

Password is not mandatory for non internal authentication type.

Figure 8.53 – Zabbix – Create user page, user Telegram

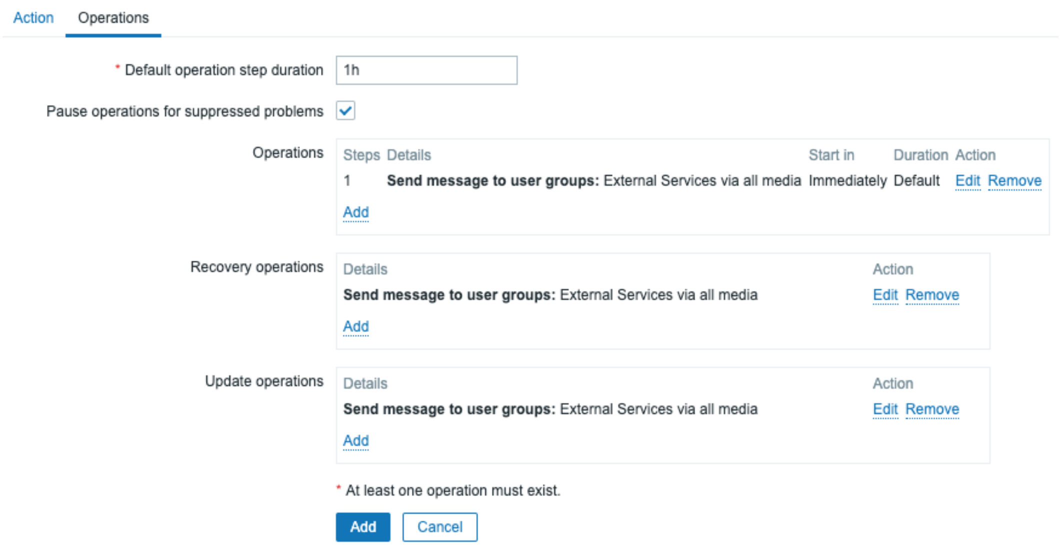
24. Now, select the **Media** tab and click on the underlined **Add** text. Add the following media:


Media

Type	Telegram
* Send to	406[redacted]
* When active	1-7,00:00-24:00
Use if severity	☐ Not classified
	☐ Information
	✓ Warning
	✓ Average
	✓ High
	✓ Disaster
Enabled	✓

Figure 8.54 – Zabbix – Create user media page, user Telegram

25. Make sure that you add the group ID to the **Send to** field without the – text and click on the blue **Add** button.
26. If you haven't followed the previous recipes, you will also need to go to **Administration** | **Actions**. Then, on the **Trigger Actions** page, click on the blue **Create action** button.
27. Use **Notify external services** as the name of the action and go to **Operations**. Add the following operations:



* Default operation step duration	1h
Pause operations for suppressed problems	✓

Operations

Steps	Details	Start in	Duration	Action
1	**Send message to user groups:** External Services via all media	Immediately	Default	Edit Remove

Recovery operations

Details	Action
Send message to user groups: External Services via all media	Edit Remove

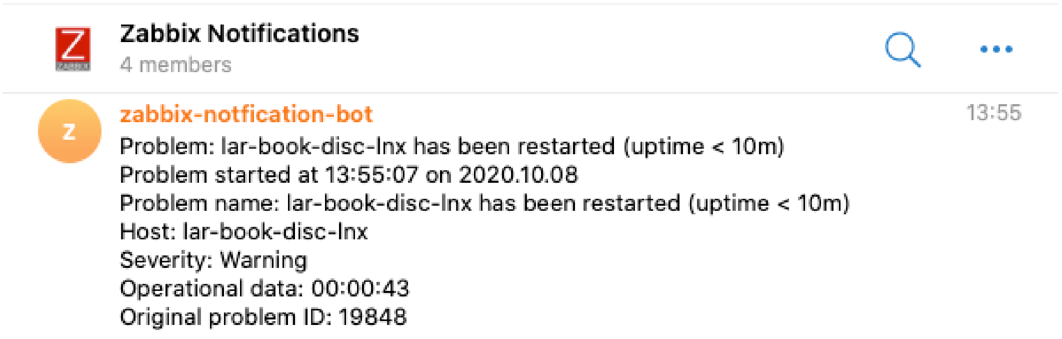
Update operations

Details	Action
Send message to user groups: External Services via all media	Edit Remove

* At least one operation must exist.

Figure 8.55 – Zabbix – Create action operations page, Notify external services for use with Telegram

28. Now, click on the blue **Add** button. With that, you're done. You can now view new problems in your Telegram group:



zabbix-notfication-bot 13:55

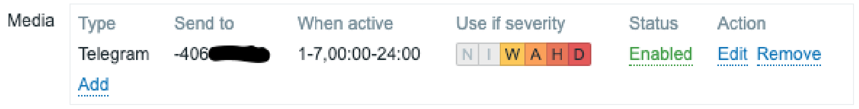
Problem: lar-book-disc-lnx has been restarted (uptime < 10m)
Problem started at 13:55:07 on 2020.10.08
Problem name: lar-book-disc-lnx has been restarted (uptime < 10m)
Host: lar-book-disc-lnx
Severity: Warning
Operational data: 00:00:43
Original problem ID: 19848

Figure 8.56 – Zabbix notifications in Telegram chat

How it works...

Slack apps, Microsoft connectors, and Telegram bots – they all work the same in the end. There's just another backend provided by the respective companies, but the Zabbix webhook remains.

Now that we've added our Zabbix Telegram integration, we are receiving notifications in our Telegram group via the Zabbix webhook:



Media

Type	Send to	When active	Use if severity	Status	Action
Telegram	-406[redacted]	1-7,00:00-24:00	N I W A H D	Enabled	Edit Remove
Add					

Figure 8.57 – Zabbix Administration | Users – Edit user media page

However, we will only receive these notifications if they match our configured settings. For instance, we've added our media type so that it only sends problems with a severity of Warning or higher to our telegram bot:

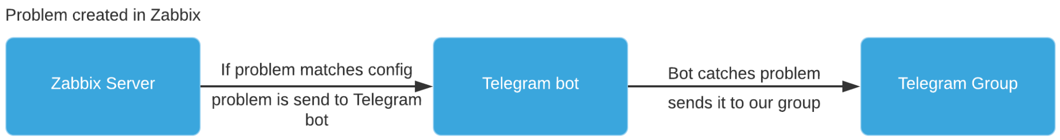

Problem created in Zabbix
Zabbix Server
If problem matches config
problem is send to Telegram bot
Telegram bot
Bot catches problem
sends it to our group
Telegram Group

Figure 8.58 – Zabbix Telegram integration diagram

Our Zabbix server is now sending our problems that match certain criteria to our Telegram bot. The bot catches these problems successfully. Because our bot is in our Telegram group, the problem is posted in our Telegram group.

There's more…

There's a very cool Zabbix community group on Telegram. Now that you have Telegram, do not forget to join using the following invite link: `https://t.me/ZabbixTech`.

See also

Make sure that you check out all the awesome features Telegram bots have to offer. There's a lot of information available directly from Telegram, and you can build amazing integrations by using them: `https://core.telegram.org/bots`.

Integrating Atlassian Opsgenie with Zabbix

Atlassian Opsgenie is so much more than just another integration service for receiving notifications. Opsgenie offers us a call system, an SMS system, iOS and android apps, two-way acknowledgements, and even an on-call schedule.

I think Opsgenie is the best tool for replacing old-school call and SMS systems and fully integrating them with Zabbix. So, let's get started with Opsgenie and see how we can get this amazing tool set up.

Getting ready

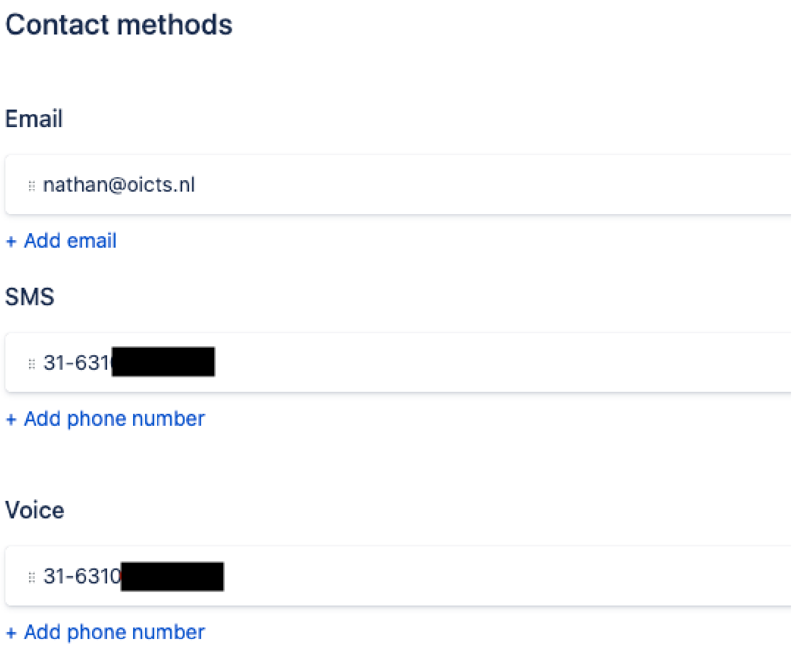
Ensure that your Zabbix server is ready. I'll be using the `lar-book-centos` server, but any Zabbix Server ready to send problems should work.

You are also going to need an Atlassian Opsgenie account with Opsgenie ready to go. I won't show you how to create accounts, but we'll start this recipe with Opsgenie ready to go.

How to do it...

Follow these steps to complete this recipe:

1. Let's start by logging into our Atlassian Opsgenie setup and going to **Settings** on the home page. From the left sidebar, click on **Notifications**.
2. Make sure that you add you email and phone number here by using the **+Add email** and **+Add phone number** buttons. We need these in order to receive notifications:



Contact methods

Email

nathan@oicts.nl

SMS

31-631[redacted]

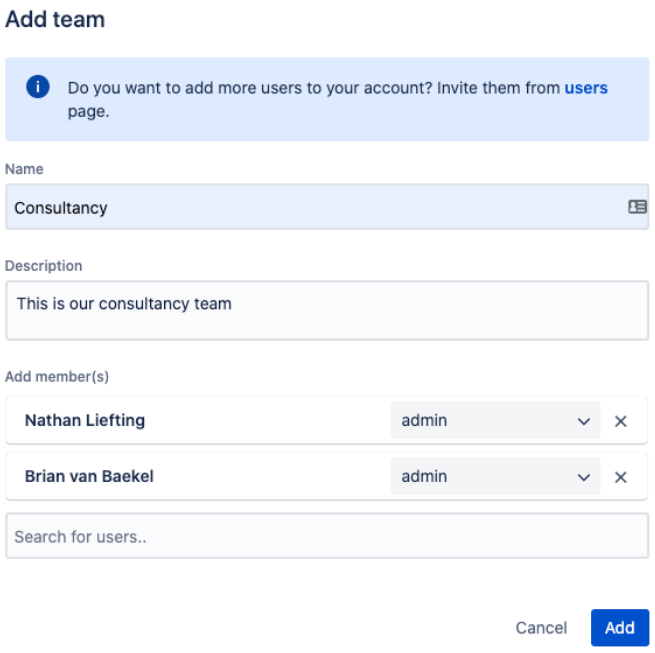
Voice

31-6310[redacted]

Figure 8.59 – Opsgenie profile – Contact methods

3. Your settings will be automatically saved once you've added them, which means we can navigate away from **Settings** to **Teams** using the top bar.

4. From the **Teams** tab, click on the blue **Add team** button in the top-right corner. Then, add the following information:


Add team

Do you want to add more users to your account? Invite them from users page.

Name: Consultancy

Description: This is our consultancy team

Add member(s)

Nathan Liefting	admin
Brian van Baekel	admin

Search for users..

Figure 8.60 – Opsgenie – Add team window

5. I've set up our **Consultancy** team with two users that are part of this team. Click on the blue **Add** button at the bottom of the window to add the new team.
6. This will take you to the new **Consultancy team** page. Click on **Integrations** and click on the blue **Add integration** button in the top-right corner.
7. When we use the search field to search for the Zabbix integration, we can see it immediately, as shown in the following screenshot:

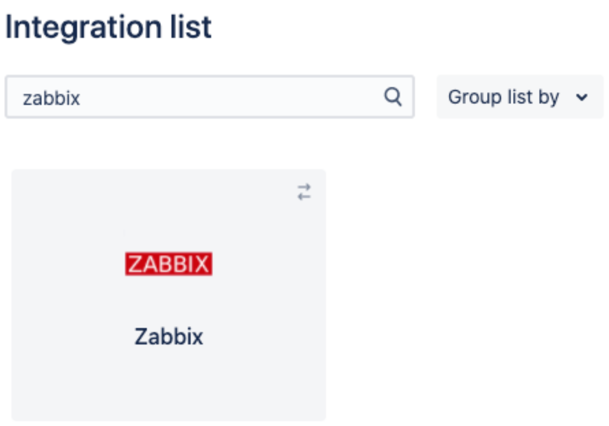
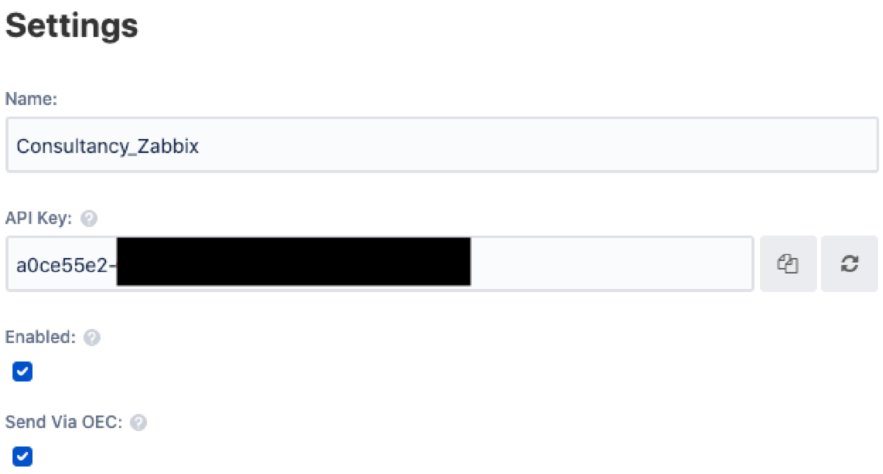


Figure 8.61 – Opsgenie – Add integration page

8. Hover over the Zabbix integration and click the blue **Add** button. This will take you to the next page, where a **Name** and **API Key** will be generated:


Settings

Name:
Consultancy_Zabbix

API Key:
a0ce55e2-[redacted]

Enabled: ✓

Send Via OEC: ✓

Figure 8.62 – Opsgenie – Add Zabbix integration page

9. Copy the **API Key** information and scroll to the bottom of the page. Here, click on the blue **Save integration** button.

10. Now, navigate to your Zabbix Server CLI and execute the following code:

    For Red Hat-based systems:

    ```
    wget https://og-release-cicd-public-oregon.s3-us-west-2.
    amazonaws.com/purpose=public/project=oec-scripts/
    env=prod/branch=master/module=oec-scripts/version=1.1.0/
    opsgenie-zabbix-1.1.0-rpm-x86-64.rpm
    ```

    For Debian-based systems:

    ```
    wget https://og-release-cicd-public-oregon.s3-us-west-2.
    amazonaws.com/purpose=public/project=oec-scripts/
    env=prod/branch=master/module=oec-scripts/version=1.1.0/
    opsgenie-zabbix-1.1.0-deb-amd64.deb
    ```

11. We can now install the downloaded Zabbix Opsgenie plugin by issuing the following command(s):

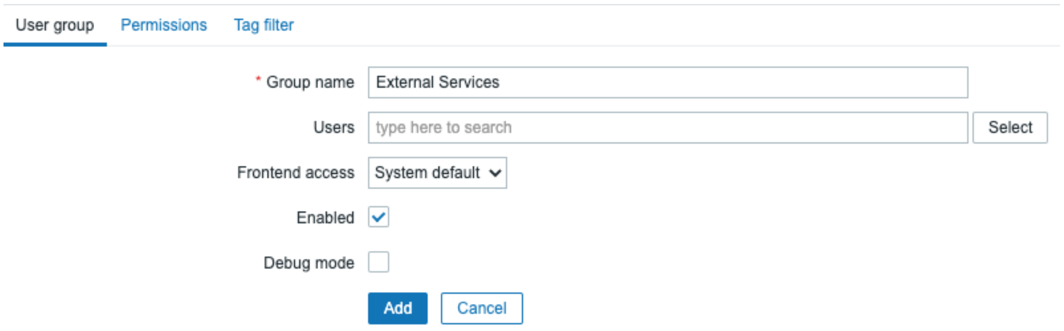
    For Red Hat-based systems:

    ```
    rpm -i opsgenie-zabbix-1.1.0-rpm-x86-64.rpm
    ```

    For Debian-based systems:

    ```
    dpkg -i opsgenie-zabbix-1.1.0-rpm-x86-64.rpm
    ```

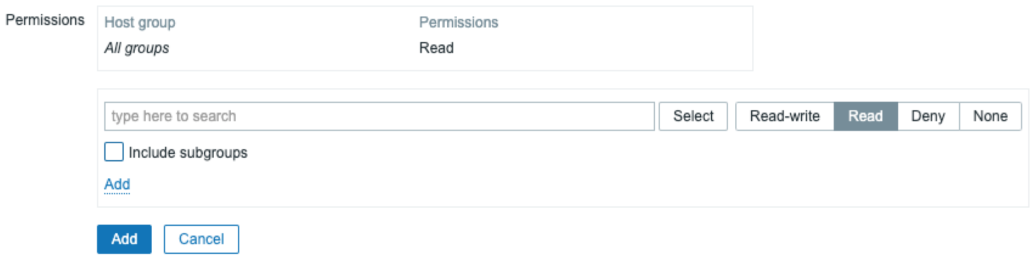
12. Once you've installed the plugin, go to the Zabbix frontend. If you haven't followed any of the previous recipes, create a new user group for our media types by navigating to **Administration | User groups** and clicking the **Create user group** button in the top-right corner. Add the following user group:



* Group name	External Services
Users	type here to search
Frontend access	System default
Enabled	✓
Debug mode	

Figure 8.63 – Zabbix – Create user group page, the External Services group for use with Opsgenie

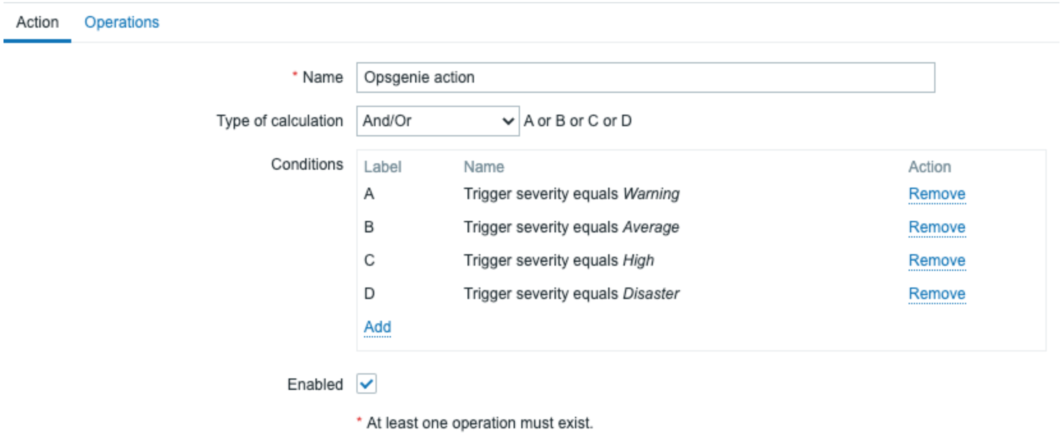
13. Click on the **Permissions** tab and click on **Select**. Make sure that you select all the groups and subgroups with **Read** permissions, as follows:



Host group	Permissions
All groups	Read

Figure 8.64 – Zabbix – Create user group permissions page, the External Services group for use with Opsgenie

14. Now, click on the blue **Add** button and finish creating this user group.
15. Next, navigate to **Administration | Actions**. On the **Trigger actions** page, click on the blue **Create action** button in the top-right corner.
16. In the **Name** field, type Opsgenie action and add the following items to **Conditions**:



* Name: Opsgenie action

Type of calculation: And/Or A or B or C or D

Conditions:

Label	Name	Action
A	Trigger severity equals *Warning*	Remove
B	Trigger severity equals *Average*	Remove
C	Trigger severity equals *High*	Remove
D	Trigger severity equals *Disaster*	Remove

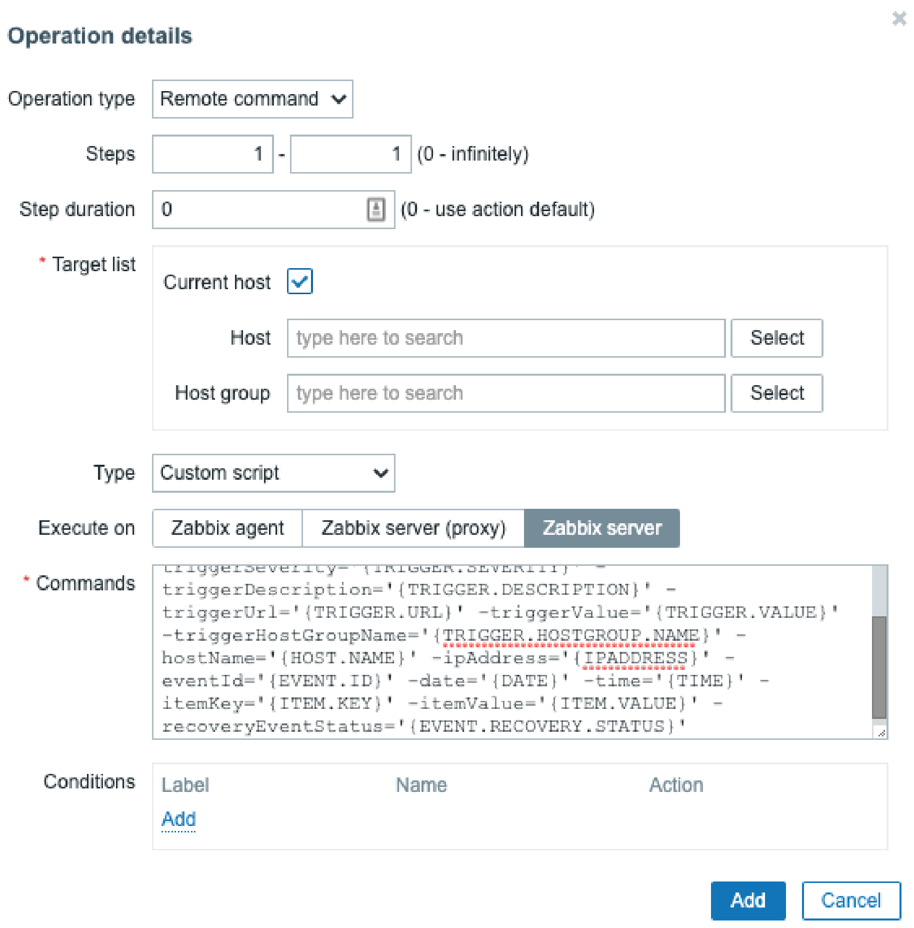
Enabled: ✓

* At least one operation must exist.

Figure 8.65 – Zabbix – Create new action page, Opsgenie action

17. Now, click on the **Operations** tab.

18. Click the underlined **Add text** option next to **Operations** to add your first operation. Set **Operation type** to **Remote command** and paste in the contents of the `/home/opsgenie/oec/opsgenie-zabbix/actionCommand.txt` file, as shown in the following screenshot:



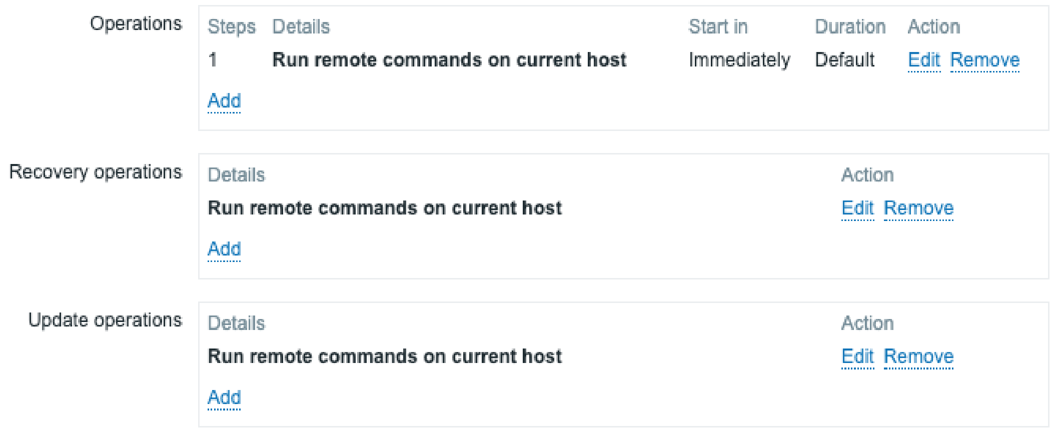
Operation details

Operation type	Remote command
Steps	1 - 1 (0 - infinitely)
Step duration	0 (0 - use action default)
* Target list	Current host ✓; Host: type here to search; Host group: type here to search
Type	Custom script
Execute on	Zabbix agent / Zabbix server (proxy) / [✓ Zabbix server]
* Commands	(see below)
Conditions	Label / Name / Action — Add

```
triggerSeverity='{TRIGGER.SEVERITY}' -
triggerDescription='{TRIGGER.DESCRIPTION}' -
triggerUrl='{TRIGGER.URL}' -triggerValue='{TRIGGER.VALUE}'
-triggerHostGroupName='{TRIGGER.HOSTGROUP.NAME}' -
hostName='{HOST.NAME}' -ipAddress='{IPADDRESS}' -
eventId='{EVENT.ID}' -date='{DATE}' -time='{TIME}' -
itemKey='{ITEM.KEY}' -itemValue='{ITEM.VALUE}' -
recoveryEventStatus='{EVENT.RECOVERY.STATUS}'
```

Figure 8.66 – Zabbix – Create action operations window, Opsgenie action

19. Repeat *step 17* for **Recovery operation** and **Update operation**. It should look like this:



Operations	Steps	Details	Start in	Duration	Action
	1	Run remote commands on current host	Immediately	Default	Edit Remove
	Add				

Recovery operations	Details	Action
	Run remote commands on current host	Edit Remove
	Add	

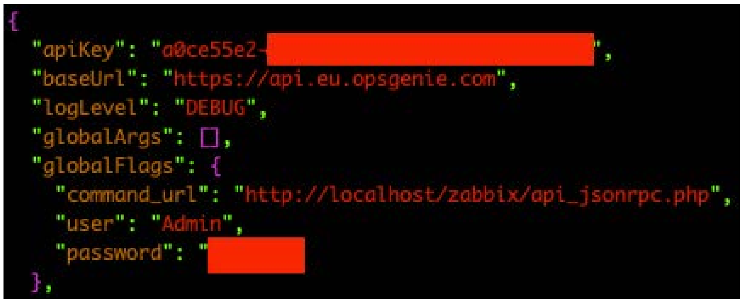
Update operations	Details	Action
	Run remote commands on current host	Edit Remove
	Add	

Figure 8.67 – Zabbix – Create action operations page, Opsgenie action

20. Click on the blue **Add** button at the bottom of the page to finish setting up the action.
21. Now, you must configure the Opsgenie-to-Zabbix integration. Edit the config file with the following command:

```
vim /home/opsgenie/oec/conf/config.json
```

22. Make sure that you edit the `apiKey`, `command_url`, `user`, and `password` lines, as shown in the following screenshot:

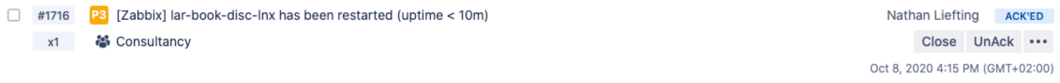


```
{
  "apiKey": "a0ce55e2-[redacted]",
  "baseUrl": "https://api.eu.opsgenie.com",
  "logLevel": "DEBUG",
  "globalArgs": [],
  "globalFlags": {
    "command_url": "http://localhost/zabbix/api_jsonrpc.php",
    "user": "Admin",
    "password": "[redacted]
  },
```

Figure 8.68 – Opsgenie config.json file

> **Important note**
>
> You will need to edit `baseUrl` if you are not located in the United States. I am in Europe, so I changed it to `https://api.eu.opsgenie.com`.

23. That's it! You can now see your alerts coming in and acknowledge them from Opsgenie:



#1716 P3 [Zabbix] lar-book-disc-lnx has been restarted (uptime < 10m)
x1 Consultancy
Nathan Liefting ACK'ED
Close UnAck •••
Oct 8, 2020 4:15 PM (GMT+02:00)

Figure 8.69 – Opsgenie alert from Zabbix

How it works...

When an alert is created in Zabbix, it is sent to Opsgenie via the Zabbix integration. This integration utilizes the Opsgenie API to catch our Zabbix information and send back a reply if required. This way, we have two-way communication between the two applications:

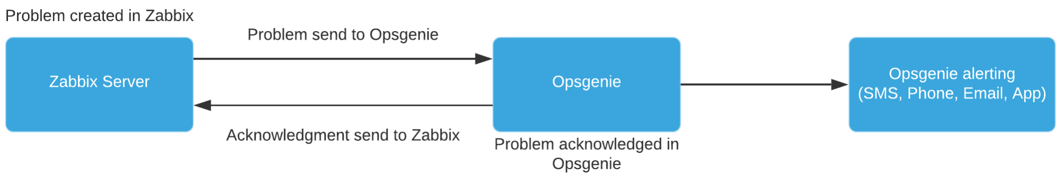

Problem created in Zabbix
Zabbix Server
Problem send to Opsgenie
Acknowledgment send to Zabbix
Opsgenie
Problem acknowledged in Opsgenie
Opsgenie alerting (SMS, Phone, Email, App)

Figure 8.70 – Opsgenie setup diagram

Opsgenie is an amazing tool that can take several tasks away from your Zabbix Server. I've used it in the past to migrate away from another monitoring tool to Zabbix. Opsgenie makes it easy to receive alerts from our products and centralize notifications:

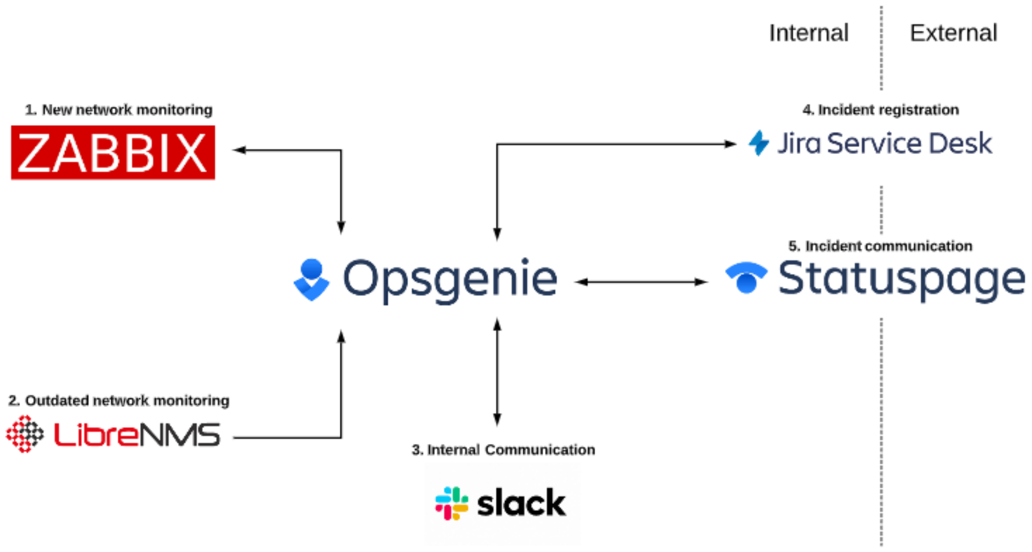

Internal
External
1. New network monitoring
ZABBIX
4. Incident registration
Jira Service Desk
Opsgenie
5. Incident communication
Statuspage
2. Outdated network monitoring
LibreNMS
3. Internal Communication
slack

Figure 8.71 – Opsgenie setup example, inspired by Wadie

Another great feature from Atlassian Opsgenie is the integration it offers with other Atlassian products. We can build a setup like the one shown in the preceding diagram to integrate all the products used in our company.

There's more...

In older versions of Opsgenie, it was possible to acknowledge problems from Zabbix to Opsgenie. You might have seen this in my blog post:

`https://blog.zabbix.com/scheduling-and-escalation-made-easy-zabbix-and-opsgenie-integration/10005/`.

Unfortunately, this doesn't work anymore. I've created a pull request with Opsgenie for part of the code to allow the acknowledgement. This can be found at the following link: `https://github.com/opsgenie/opsgenie-integration/pull/57`.

Atlassian Opsgenie may add this in the future. Please use the aforementioned GitHub link if you have any additional information you would like to provide or want this feature to be added. If you are running an older version of Opsgenie, please refer to my Zabbix blog post.

9
Extending Zabbix Functionality with Custom Scripts and the Zabbix API

Zabbix offers a lot of functionality out of the box. But where Zabbix really shines is customization, not only through the default frontend but especially with scripts and the Zabbix API.

In this chapter, I will go over the basics of using the Zabbix API. We will then see how a Python script can utilize the API to build something cool, such as a jumphost. After that, we'll use some scripts written by *Brian van Baekel* to schedule maintenance periods as a Zabbix user and to enable and disable hosts with limited permissions from a Zabbix map.

After following these recipes, you'll be more than ready to tackle the Zabbix API and you'll know how to use scripts to extend Zabbix functionality. This chapter will expand your possibilities with Zabbix to almost endless proportions and you'll be ready to become a professional Zabbix user yourself.

In this chapter, we will cover the following topics:

- Using the Zabbix API for extending functionality
- Building a jumphost using the Zabbix API and Python
- Creating maintenance periods as a Zabbix user
- Enabling and disabling a host from Zabbix maps

Technical requirements

We are going to need a Zabbix server as well as some new Linux hosts. We will also need to have a general knowledge of scripting and programming. We are going to use Python to extend some functionality of Zabbix, which we'll provide scripts for.

The code required for the chapter can be found at the following link:

`https://github.com/PacktPublishing/Zabbix-5-Network-Monitoring-Cookbook/tree/master/chapter9`

Make sure to keep all of this ready and you'll be sure to nail these recipes.

Using the Zabbix API for extending functionality

An API is your gateway to getting started with extending the functionality of any piece of software. Luckily, Zabbix offers a solid working API that we can use to extend our functionality with ease.

In this recipe, we'll explore the use of the Zabbix API to do some tasks, creating a good basis to start working with the Zabbix API in your actual production environments.

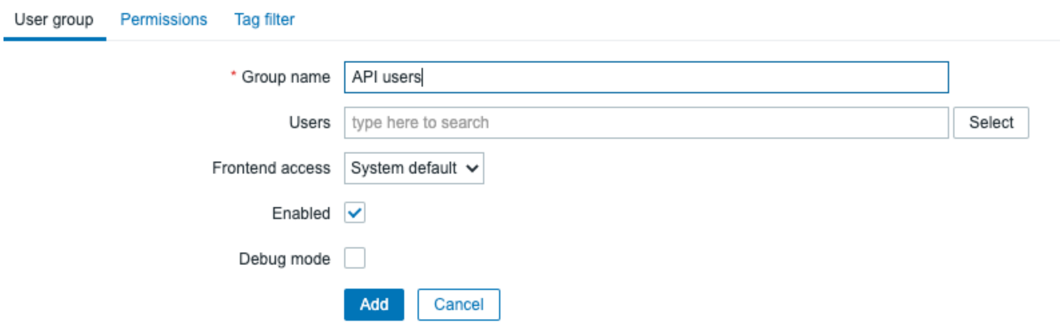
Getting ready

We are going to need a Zabbix server with some hosts. I'll be using our host `lar-book-centos` from the previous chapters, but feel free to use any Zabbix server. I will also use another Linux host to do the API calls from, and this can be any Linux host.

We will need to install Python 3 on the Linux host, though, as we'll be using this to create our API calls.

How to do it

1. First things first, let's log in to our Zabbix frontend. Navigate to **Administration** | **User groups**.
2. We are then going to need a new user for our API calls, but to do that we need a user group first. Click the blue **Create user group** button.
3. Now let's create the following **User group**:



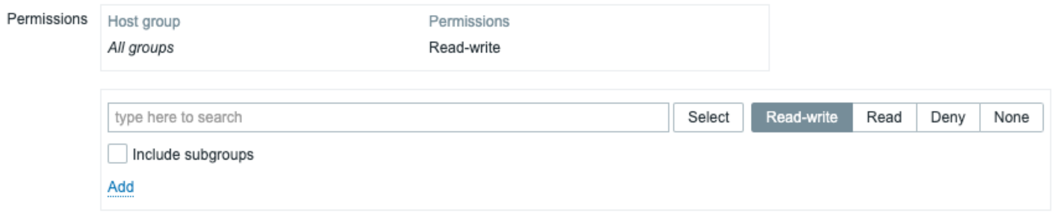
* Group name	API users
Users	type here to search
Frontend access	System default
Enabled	✓
Debug mode	

Figure 9.1 – Zabbix Administration | Users group, create user group page, API users

4. Now, let's click on **Permissions** to fill out some more information before saving.
5. Click on the white **Select** button and select all the user groups available in your Zabbix server. Then press the blue **Select** button.

> **Important note**
> Do not just add all host groups to a production environment, as you might expose your information through the Zabbix API. Always make sure to only add permissions for the host groups that the user requires access to.

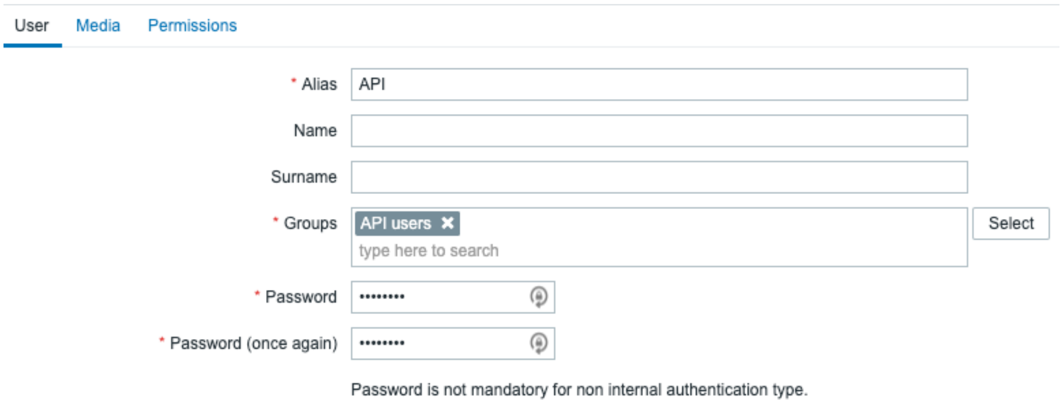
6. Click on **Read-write** and then press the underlined **Add** button. It should now look like this:



Host group	Permissions
All groups	Read-write

Figure 9.2 – Zabbix Administration | Users group, create user group permissions page, API users

7. Notice how it says **All groups** with **Read-write** permissions. Now you can click the blue **Add** button at the bottom of the page.
8. Next up, go to **Administration | Users** and click on **Create user**. We'll create the following user:

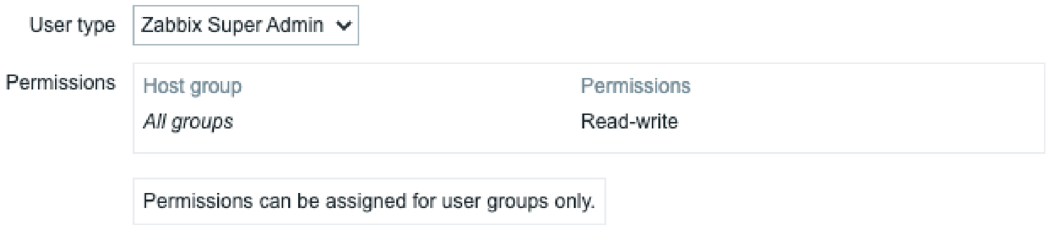


* Alias	API
Name	
Surname	
* Groups	API users ✖ type here to search
* Password	••••••••
* Password (once again)	••••••••

Password is not mandatory for non internal authentication type.

Figure 9.3 – Zabbix Administration | Users, Create user page, API user

9. In general, you should always create a secure password for a production environment. For this lab environment, I'll just be using the password `password` though.
10. Let's edit the **Permissions** tab for the API user as well. Add the following:



User type: Zabbix Super Admin

Permissions

Host group	Permissions
All groups	Read-write

Permissions can be assigned for user groups only.

Figure 9.4 – Zabbix Administration | Users, Create user Permissions page, API user

11. Click on the blue **Add** button at the bottom of the page to finish creating this user.
12. Now let's log in to the Linux host CLI. This can be any Linux host, besides our Zabbix server. I'll be using a CentOS host.
13. Install Python 3 on the host CLI with the following command:

    For RHEL-based systems:

```
dnf install python3
```

For Debian-based systems:

```
apt-get install python3
```

14. Python `pip` should've been installed with this package by default as well. If not, issue the following command:

    For RHEL-based systems:

    ```
    dnf install python3-pip
    ```

    For Debian-based systems:

    ```
    apt-get install python3-pip
    ```

15. Now let's install our dependencies using Python `pip`. We'll need these dependencies as they'll be used in the script:

    ```
    pip3 install requests
    pip3 install lxml
    ```

16. Download the start of our script from the Packt GitHub repo by issuing the following command:

    ```
    wget https://raw.githubusercontent.com/PacktPublishing/
    Zabbix-5-Network-Monitoring-Cookbook/master/chapter9/api_
    test.py
    ```

17. If you can't use `wget` from your host, you can download the script at the following URL: `https://github.com/PacktPublishing/Zabbix-5-Network-Monitoring-Cookbook/blob/master/chapter9/api_test.py`
18. Next up, we are going to edit our newly downloaded script by executing the following command:

    ```
    vim api_test.py
    ```

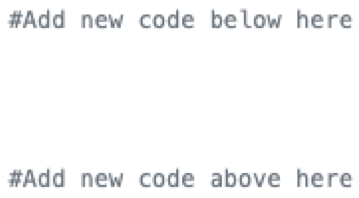
19. Change the following lines to match your Zabbix configuration, with `url` being the Zabbix frontend URL of your installation and `username` and `password` being the values of the user we just created:

    ```
    url = 'http://10.16.16.152/zabbix/api_jsonrpc.php?'
    username = "API"
    password = "password"
    ```

20. Save your file and move on. This script already contains our login part to make it easier for us. Let's edit the script to do an API call. Execute the following:

```
vim api_test.py
```

21. We are going to add some lines to our script to retrieve our host ID, hostname, and the interfaces of all our Zabbix hosts. Make sure to add your new code between the comments shown in the following screenshot:



```
#Add new code below here



#Add new code above here
```

Figure 9.5 – Comments showing where to put code

22. Now add the following lines of code:

```
#Function to retrieve the hosts and interfaces
def get_hosts(api_token, url):

    payload = {
    "jsonrpc": "2.0",
    "method": "host.get",
    "params": {
        "output": [
            "hostid",
            "host"
        ],
        "selectInterfaces": [
            "interfaceid",
            "ip",
            "main"
        ]
    },
    "id": 2,
        "auth": api_token
        }

    resp = requests.post(url=url, json=payload )
    out = resp.json()

    return out['result']
```

23. Then, we'll also add lines to write the requested information to a file so we can see what happens after execution:

```
#Write the results to a file
def generate_host_file(hosts,host_file):

    hostname = None
    f = open(host_file, "w")


    #Write the host entries retrieved from Zabbix
    for host in hosts:
        hostname = host['host']
        for interface in host["interfaces"]:
            if interface["main"] == "1":
                f.write(hostname + " " + interface["ip"]
+ "\n")

    f.close()
    return
```

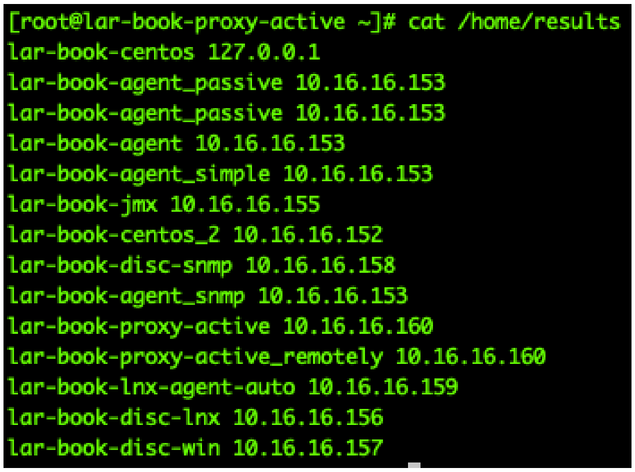
24. You should be able to execute this now by executing the following:

```
python3 api_test.py
```

25. This should run but it won't give you any output. If this doesn't work, make sure to retrace your steps.
26. Let's check out the file to see what happened by executing the following:

```
cat /home/results/
```

27. The output of the preceding command should look like this:



```
[root@lar-book-proxy-active ~]# cat /home/results
lar-book-centos 127.0.0.1
lar-book-agent_passive 10.16.16.153
lar-book-agent_passive 10.16.16.153
lar-book-agent 10.16.16.153
lar-book-agent_simple 10.16.16.153
lar-book-jmx 10.16.16.155
lar-book-centos_2 10.16.16.152
lar-book-disc-snmp 10.16.16.158
lar-book-agent_snmp 10.16.16.153
lar-book-proxy-active 10.16.16.160
lar-book-proxy-active_remotely 10.16.16.160
lar-book-lnx-agent-auto 10.16.16.159
lar-book-disc-lnx 10.16.16.156
lar-book-disc-win 10.16.16.157
```

Figure 9.6 – The cat command with our results showing in the file

We've now written a short script in Python to use the Zabbix API.

How it works

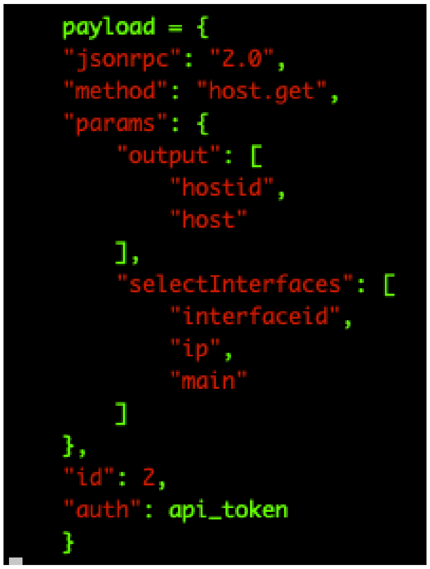
Coding with the Zabbix API can be done with Python, but it's definitely not our only option. We can use a wide variety of coding languages, including Perl, Go, C#, and Java.

In our example though, we've used Python, so let's see what we do here. If we look at the script, we have three main functions:

- `get_api_token`
- `get_hosts`
- `generate_host_file`

Our `get_api_token` function is used to log in to the Zabbix API as a user with the credentials provided in our `url`, `username`, and `password` variables. Zabbix returns an API token for us to use – if we extend this script we can save the API token and make the login process even more secure.

We then call on the `get_hosts` function to retrieve information from the Zabbix API with the acquired token:



```
payload = {
"jsonrpc": "2.0",
"method": "host.get",
"params": {
    "output": [
        "hostid",
        "host"
    ],
    "selectInterfaces": [
        "interfaceid",
        "ip",
        "main"
    ]
},
"id": 2,
"auth": api_token
}
```

Figure 9.7 – Python function Zabbix API payload

Looking at the code, we use a JSON `payload` to request information such as `host` for the hostname, `hostid` for the host ID, and `ip` for the interface's IP address.

Now, if we look at our last function, `generate_host_file`, we can see that we write the host with an interface IP to the `/home/results` file. This way we have a solid script for writing host information to a file.

Now, if you're not familiar with Python or coding in general, working with the Zabbix API might be a big step to take. Let's take a look at how the API actually works:

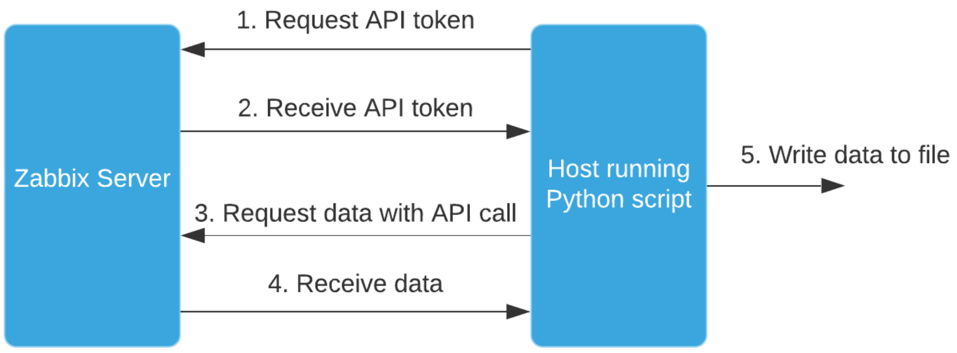


Figure 9.8 – Python script Zabbix API functionality diagram

Looking at the preceding diagram, we can see our steps again. In *step 1* and *step 2*, we utilize Python to authenticate to the Zabbix API and we receive our API token.

Then in *step 3*, we make an API call, using our authentication. Next, in *step 4*, we receive the data from Zabbix to use further in our Python script.

Step 5 is then our data processing step. We can do anything we want with the data received from the Zabbix API, but in our case, we write our information to a file. That's how we use the Zabbix API for extending functionality. This is the step where our file gets filled with hostnames and IP information.

See also

If you are interested in learning more about the Zabbix API, check out the Zabbix documentation at `https://www.zabbix.com/documentation/current/manual/api`.

If you are looking for Zabbix API libraries, check out the community-maintained libraries available at `https://zabbix.org/wiki/Docs/api/libraries`.

Building a jumphost using the Zabbix API and Python

A lot of organizations have a jumphost to access servers, switches, and their other equipment from a host. A jumphost generally has all the firewall rules needed to access everything important. Now if we keep our monitoring up to date, we should have every single host in there as well.

My friend, ex-colleague, and fellow Zabbix geek, *Yadvir Singh*, had the amazing idea to create a Python script to export all Zabbix hosts with their IPs to the `/etc/hosts` file on another Linux host. Let's see how we can build a jumphost just like his.

Getting ready

We are going to need a new host for this recipe with Linux installed and ready. We'll call this host `lar-book-jump`. We will also need our Zabbix server, for which I'll use `lar-book-centos`.

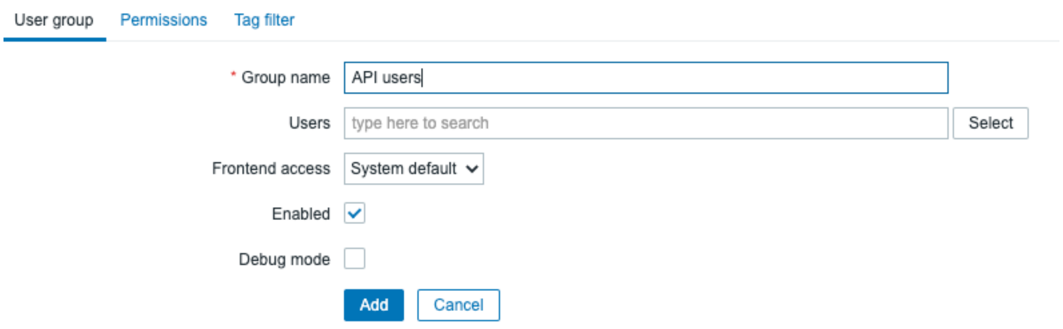
Also, it is important to navigate to *Yadvir* on his GitHub account, drop him a follow, and star his repository if you too think this is a cool script: `https://github.com/cheatas/zabbix_scripts`.

Important note

Setting up this script will override your `/etc/hosts` file every time the script is executed. Only use this script when you understand what it's doing, make sure you use an empty host for this lab, and check the default `/etc/hosts` settings.

How to do it...

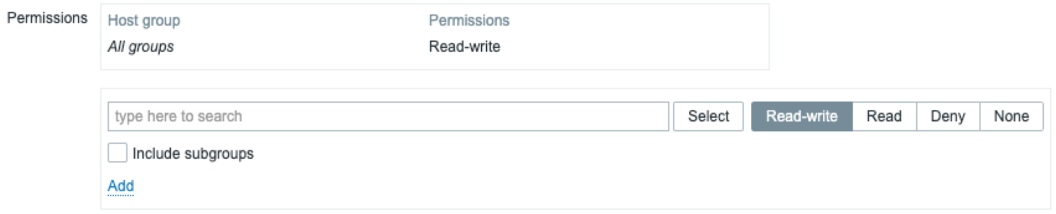
1. If you haven't already created an API user, then know that we are going to need a new user for API calls, but to create that we need a user group first. Click the blue **Create user group** button.
2. Now let's create the following **User group**:



* Group name	API users
Users	type here to search
Frontend access	System default
Enabled	✓
Debug mode	

Figure 9.9 – Zabbix Administration | Users group, create user group page, API users for jumphost

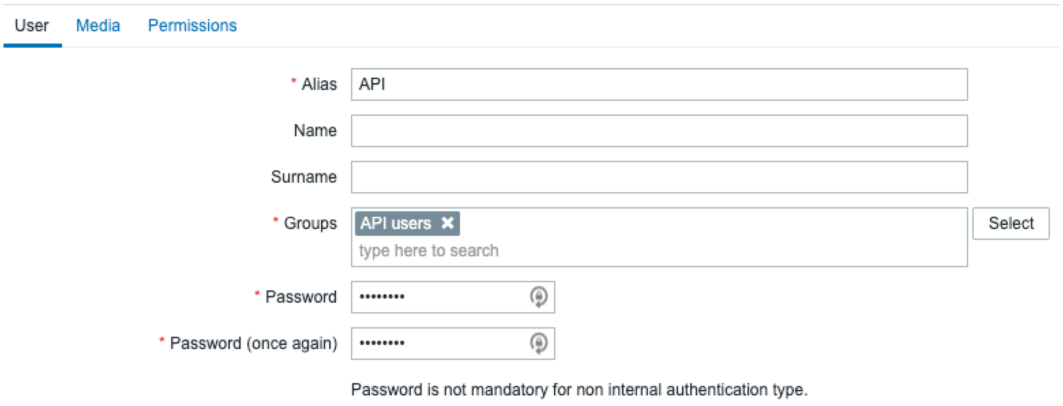
3. Now let's click on **Permissions** to fill out some more information before saving.
4. Click on the white **Select** button and select all the user groups available in your Zabbix server. Then press the blue **Select** button.
5. Click on **Read-write** and then press the underlined **Add** button. It should now look like this:



Host group	Permissions
All groups	Read-write

Figure 9.10 – Zabbix Administration | Users group, create user group permissions page, API users for jumphost

6. Notice how it says **All groups** with **Read-write** permissions. Now, you can click the blue **Add** button at the bottom of the page.
7. Next up, go to **Administration | Users** and click on **Create user**. We'll create the following user:

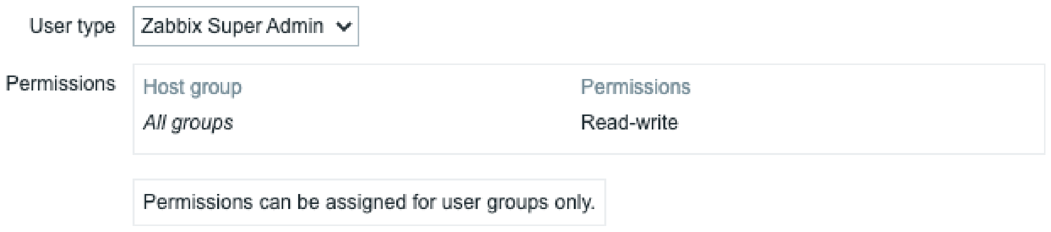


* Alias	API
Name	
Surname	
* Groups	API users ✕ type here to search
* Password	••••••••
* Password (once again)	••••••••

Password is not mandatory for non internal authentication type.

Figure 9.11 – Zabbix Administration | Users, create user page, API user for jumphost

8. In general, you should always create a secure password for a production environment. For this lab environment, I'll just be using the password `password` though.
9. Let's edit the **Permissions** tab for the API user as well. Add the following:



User type: Zabbix Super Admin

Permissions

Host group	Permissions
All groups	Read-write

Permissions can be assigned for user groups only.

Figure 9.12 – Zabbix Administration | Users, create user Permissions page, API user for jumphost

10. Click on the blue **Add** button at the bottom of the page to finish creating this user.

11. Install Python 3 on the host CLI with the following command:

    For RHEL-based systems:

    ```
    dnf install python3
    ```

    For Debian-based systems:

    ```
    apt-get install python3
    ```

12. Python `pip` should've been installed with this package by default as well. If not, issue the following command:

    For RHEL-based systems:

    ```
    dnf install python3-pip
    ```

    For Debian-based systems:

    ```
    apt-get install python3-pip
    ```

13. Now let's install our dependencies using Python `pip`. We'll need these dependencies as they'll be used in the script:

    ```
    pip3 install requests
    ```

14. First things first, log in to our new Linux host, `lar-book-jump`, and download Yadvir's script to your Linux host with the following command:

    ```
    wget https://raw.githubusercontent.com/cheatas/zabbix_
    scripts/main/host_pull_zabbix.py
    ```

15. If you can't use `wget` from your host, you can download the script at the following URL: `https://github.com/cheatas/zabbix_scripts/blob/main/host_pull_zabbix.py`.
16. As a backup, we also provide this script in the Packt repository. You may download this version at `https://github.com/PacktPublishing/Zabbix-5-Network-Monitoring-Cookbook/blob/master/chapter9/host_pull_zabbix.py`.
17. Now let's edit the script by executing the following command:

    ```
    vim host_pull_zabbix.py
    ```

18. We will need to request our API token and change the following lines to match your Zabbix configuration, with `zabbix_url` being the Zabbix frontend URL of your installation, and `zabbix_username` and `zabbix_password` being the values of the user we just created:

```
zabbix_url = 'http://10.16.16.152/zabbix/api_jsonrpc.
php?'
zabbix_username = "API"
zabbix_password = "password"
```

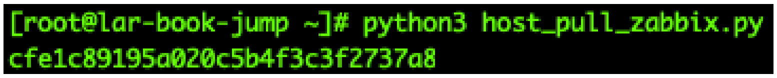
19. We also need to uncomment the following line at the end of the file by removing # from the beginning of the following line like this:

```
print(get_api_token(zabbix_url))
```

20. Now execute the script with the following command:

```
python3 host_pull_zabbix.py
```

21. This will return a token like this:



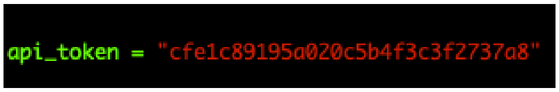
```
[root@lar-book-jump ~]# python3 host_pull_zabbix.py
cfe1c89195a020c5b4f3c3f2737a8
```

Figure 9.13 – Token returned after execution

22. Copy this token and edit the file again with the following command:

```
vim host_pull_zabbix.py
```

23. Paste the API token that was printed between the double quotes at `api_token = ""`. It will look like this:



```
api_token = "cfe1c89195a020c5b4f3c3f2737a8"
```

Figure 9.14 – Token filled in our script

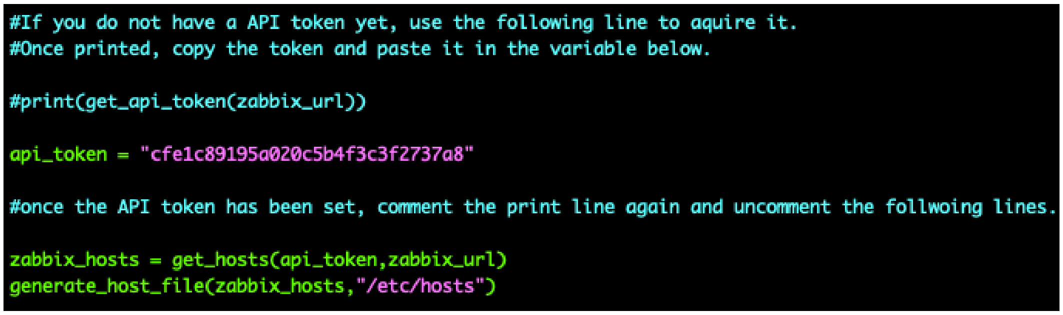
24. Then re-comment the following line by placing a # before the following line:

```
#print(get_api_token(zabbix_url))
```

25. We also need to uncomment the following lines:

```
zabbix_hosts = get_hosts(api_token,zabbix_url)
generate_host_file(zabbix_hosts,"/etc/hosts")
```

26. The end of the script should now look like this:

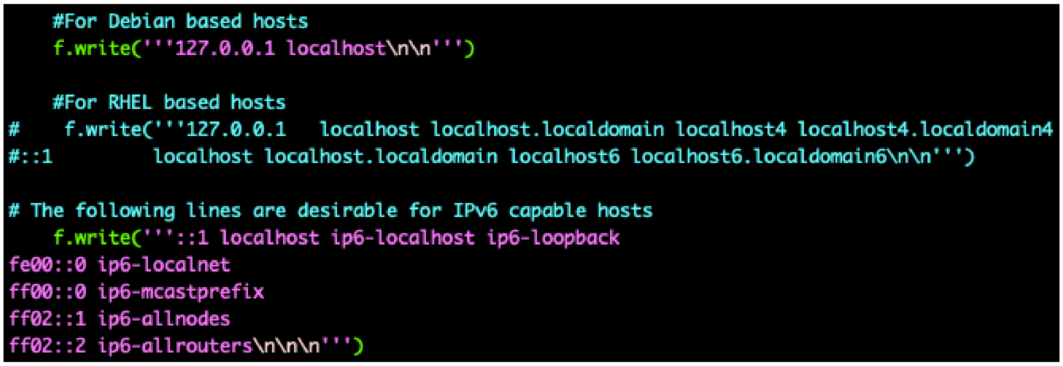


```
#If you do not have a API token yet, use the following line to aquire it.
#Once printed, copy the token and paste it in the variable below.

#print(get_api_token(zabbix_url))

api_token = "cfe1c89195a020c5b4f3c3f2737a8"

#once the API token has been set, comment the print line again and uncomment the follwoing lines.

zabbix_hosts = get_hosts(api_token,zabbix_url)
generate_host_file(zabbix_hosts,"/etc/hosts")
```

Figure 9.15 – End of the script after receiving the API token and with commenting removed

27. We can now remove the `zabbix_password` entry from our file.
28. Last but not least, make sure to comment and uncomment the right lines for your Linux distro. It will look like this:

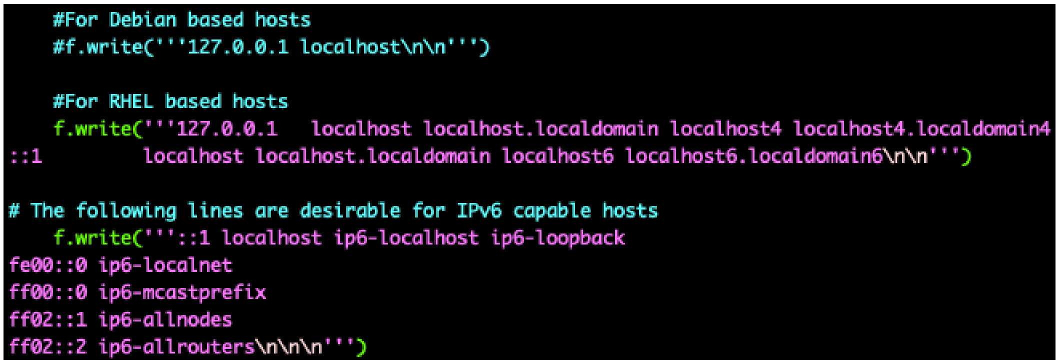
    Debian-based:



```
    #For Debian based hosts
    f.write('''127.0.0.1 localhost\n\n''')

    #For RHEL based hosts
#    f.write('''127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
#::1         localhost localhost.localdomain localhost6 localhost6.localdomain6\n\n''')

# The following lines are desirable for IPv6 capable hosts
    f.write(''':1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters\n\n\n''')
```

Figure 9.16 – Print to file for Debian-based systems

For RHEL-based systems:



```
    #For Debian based hosts
    #f.write('''127.0.0.1 localhost\n\n''')

    #For RHEL based hosts
    f.write('''127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6\n\n''')

# The following lines are desirable for IPv6 capable hosts
    f.write('''::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters\n\n\n''')
```

Figure 9.17 – Print to file for RHEL-based systems

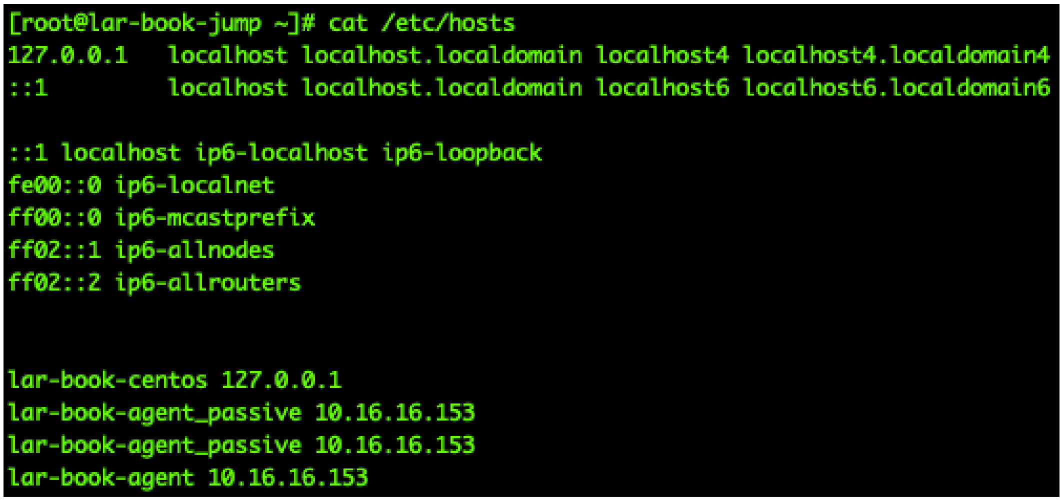
29. That's all there is to do, so we can now execute the script again and start using it. Let's execute the script as follows:

```
python3 host_pull_zabbix.py
```

30. Test whether it worked by looking at the host file with the following command:

```
cat /etc/hosts
```

This should give us an output like that in the following screenshot:



```
[root@lar-book-jump ~]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6

::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters


lar-book-centos 127.0.0.1
lar-book-agent_passive 10.16.16.153
lar-book-agent_passive 10.16.16.153
lar-book-agent 10.16.16.153
```

Figure 9.18 – /etc/hosts filled with our script information

31. We can now try to SSH directly to the name of a host, instead of having to use the IP, by issuing the following command:

```
ssh lar-book-agent_passive
```

32. We can also use it to find hosts from the file with the following command:

```
cat /etc/hosts | grep agent
```

33. Let's do one more thing. We want this script to be as up to date as possible. So, let's add a cronjob. Issue the following command to add a cronjob:

```
crontab -e
```

34. Then add the following line, making sure to fill in the right script location for your setup:

```
*/15 * * * * $(which python3) /home/host_pull_zabbix.py
>> ~/cron.log 2>&1
```

That's it – we will now have an up-to-date `/etc/hosts` file all the time with our new Python script and Zabbix.

How it works

If your organization uses Zabbix as the main monitoring system, you now have the skills and knowledge to create an organized, reliably up-to-date, and easy-to-use jumphost. Jumphosts are super useful when set up correctly, but it's important to keep them clean so that they are easy to update.

By using this script, we only add Python 3 and a simple script as a requirement to the server, but the end result is a jumphost that knows about all hosts in the environment.

If you've followed along with the previous *Using the Zabbix API for extending functionality* recipe, then you might notice that it works in roughly the same way. We can see in the following diagram how we utilize the script:

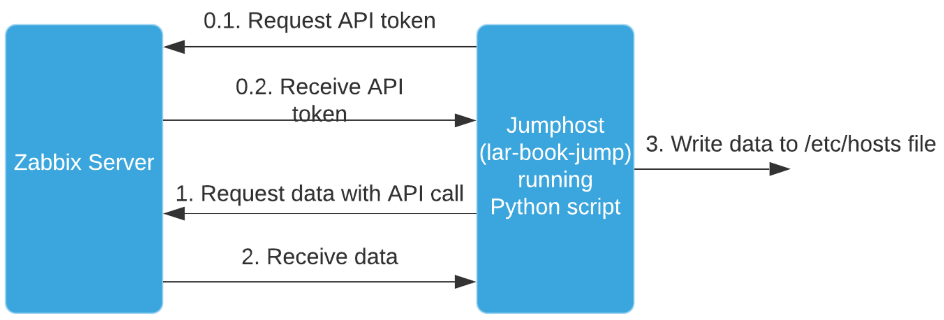

0.1. Request API token
0.2. Receive API token
Zabbix Server
Jumphost (lar-book-jump) running Python script
3. Write data to /etc/hosts file
1. Request data with API call
2. Receive data

Figure 9.19 – Jumphost using script functionality diagram

In *step 0.1* of the preceding diagram, our script requests an API token, which we receive in *step 0.2* by printing it to the CLI. We edit our script (as we did in *step 22*) to use the token instead of the username and password for security. This step is not repeated in the process, which is why these are *step 0.1* and *step 0.2*. Every time the script is executed from now on, it will start at *step 1*.

Important note

In Zabbix 6, we might see the ability to configure API keys in the Zabbix frontend. This would eliminate the need to script the API key retrieval and improve security. At the time of writing, though, this feature is on the Zabbix development team's idea board. So, I'm not making any promises.

The requested API token will not expire unless user auto-logout is configured. I do not recommend configuring this option for your Zabbix API users.

After editing, our script will start at *step 1* of the preceding diagram to request data with an API call. We receive this data in *step 2*. In the script, we add our default values and then write all the hostnames and IP addresses to the `/etc/hosts` file.

Now, because a Linux host uses the `/etc/hosts` file for hostname-to-IP translation, we can use the real names of servers in Zabbix to SSH to the hosts. This makes it easier for us to use the jumphost, as we can use the same name as the hostname we know from the Zabbix frontend.

See also

Yadvir will keep updating the script after writing this recipe (we've been using `version 1.0` so far). Make sure to follow his GitHub account and star his repository to get the updates. Also, if you have some cool ideas for additions you can always open a pull request.

The Zabbix community is all about sharing cool ideas and useful scripts like this one. As *Yadvir* has shown, we can get very valuable stuff from each other. Be like *Yadvir*, use the Zabbix Share portal and support other Zabbix users by adding to their GitHub repositories.

Creating maintenance periods as a Zabbix User

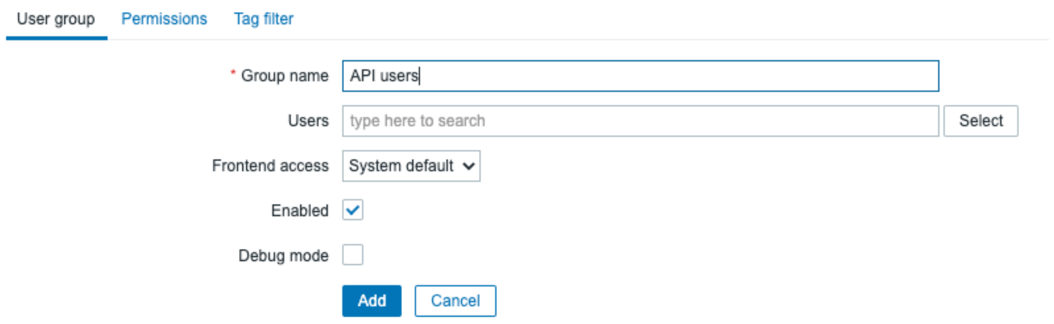
I've noticed that it is not possible to schedule maintenance periods as a Zabbix User. For some companies, this may be a requirement, so I created an extension for it. In this recipe, I will show you just how to work with this Python script.

Getting ready

For this recipe, all we are going to need is our Zabbix server, some knowledge of Python, and some knowledge of the Zabbix API.

How to do it

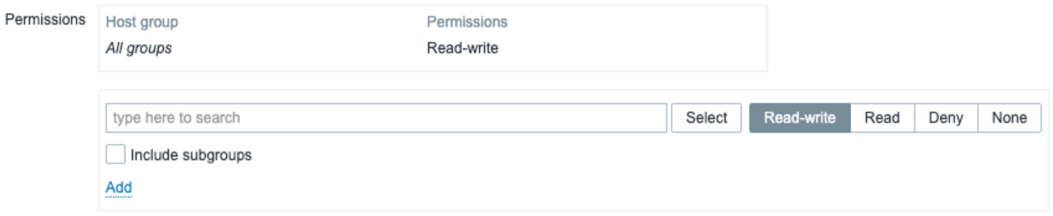
1. If you haven't already created an API user then we are going to need a new user for API calls, but to do that we need a user group first. Click the blue **Create user group** button.
2. Now let's create the following **User group**:



* Group name	API users
Users	type here to search
Frontend access	System default
Enabled	✓
Debug mode	

Figure 9.20 – Zabbix Administration | Users group, create user group page, API users for maintenance

3. Now, let's click on **Permissions** to fill out some more information before saving.
4. Click on the white **Select** button and select all the user groups available in your Zabbix server. Then press the blue **Select** button.
5. Click on **Read-write** and then press the underlined **Add** button. It should now look like this:

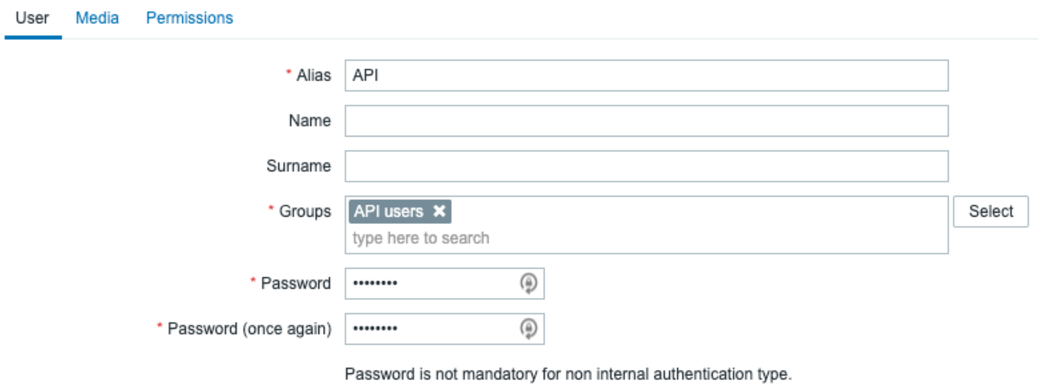


Permissions

Host group	Permissions
All groups	Read-write

Figure 9.21 – Zabbix Administration | Users group, create user group permissions page, API users for maintenance

6. Notice how it says **All groups** with **Read-write** permissions. Now you can click the blue **Add** button at the bottom of the page.
7. Next up, go to **Administration** | **Users** and click on **Create user**. We'll create the following user:



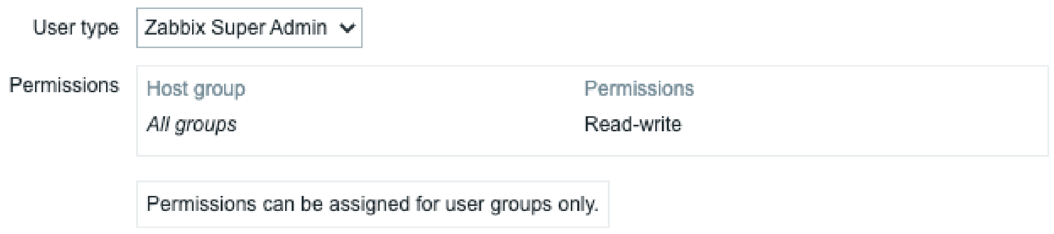
* Alias	API
Name	
Surname	
* Groups	API users
* Password	••••••••
* Password (once again)	••••••••

Password is not mandatory for non internal authentication type.

Figure 9.22 – Zabbix Administration | Users, Create user page, API user for maintenance

8. In general, you should always create a secure password for a production environment. For this lab environment, I'll just be using the password `password` though.

9. Let's edit the **Permissions** tab for the API user as well. Add the following:



User type: Zabbix Super Admin

Permissions

Host group	Permissions
All groups	Read-write

Permissions can be assigned for user groups only.

Figure 9.23 – Zabbix Administration | Users, Create user Permissions page, API user for maintenance

10. Click on the blue **Add** button at the bottom of the page to finish creating this user.
11. Now, let's log in to our Zabbix server CLI and create a new directory:

```
mkdir /etc/zabbix/frontendscripts
```

12. Change to the new directory:

```
cd /etc/zabbix/frontendscripts
```

13. Now download the `Public` script from the *Opensource ICT Solutions* GitHub:

```
wget https://github.com/OpensourceICTSolutions/zabbix-
maintenance-from-frontend/archive/v1.0.tar.gz
```

14. If you can't use `wget` from your host, check out the script here: `https://github.com/OpensourceICTSolutions/zabbix-maintenance-from-frontend/releases/tag/v1.0`.
15. Unzip the file with the following command:

```
tar -xvzf v1.0.tar.gz
```

16. Remove the `tar` file using the following command:

```
rm v1.0.tar.gz
```

17. Move the script over from the newly created folder with the following command:

```
mv zabbix-maintenance-from-frontend-1.0/maintenance.py ./
```

18. We are going to need Python to use this script so let's install it as follows:

    For RHEL-based systems:

    ```
    dnf install python3 python3-pip
    ```

    For Debian-based systems:

    ```
    apt-get install python python-pip
    ```

19. We will also need the `requests` module from `pip`, so let's install it with the following command:

    ```
    pip3 install requests
    ```

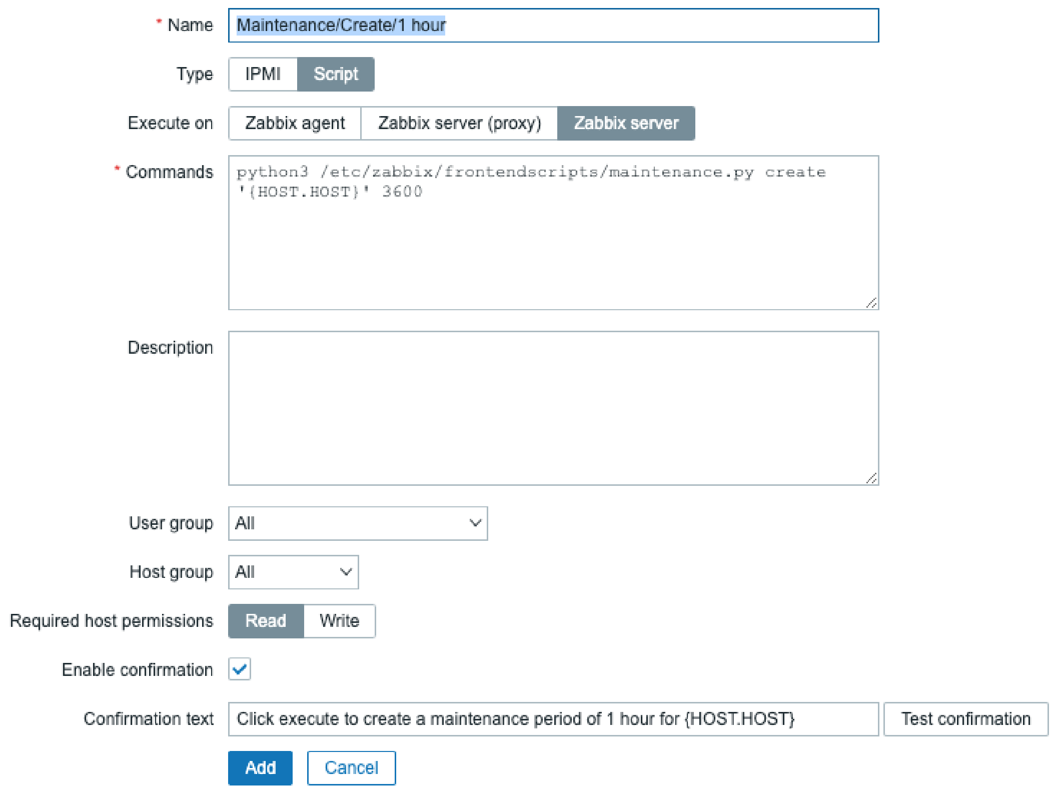
20. Now let's edit the script with the following command:

    ```
    vim maintenance.py
    ```

21. In the section starting with `url`, we will need to enter our own setup information. I will fill in the following, but be sure to enter your own information:

    ```
    url = 'http://10.16.16.152/zabbix/api_jsonrpc.php?'
    username = "API"
    password = "password"
    ```

22. Now we can move on to our Zabbix frontend to add a frontend script. Navigate to **Administration | Scripts** then click the blue **Create script** button in the top right corner.
23. Now add the following script:

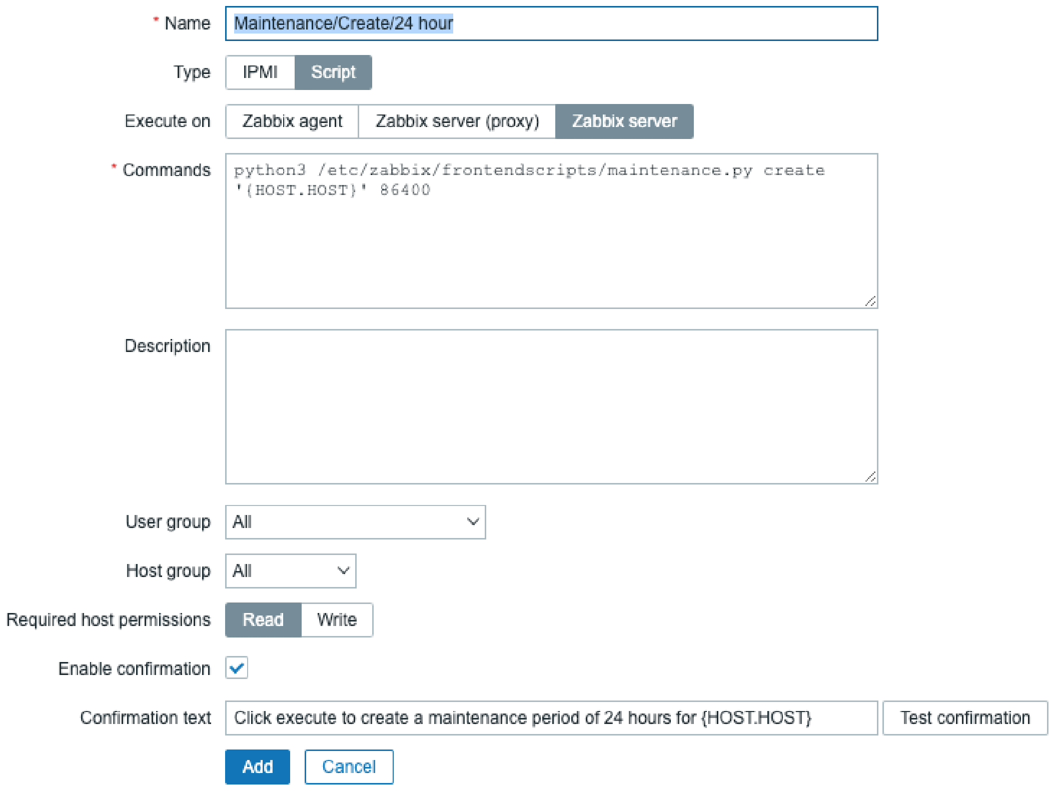


* Name	Maintenance/Create/1 hour
Type	IPMI [✓ Script]
Execute on	Zabbix agent / Zabbix server (proxy) / [✓ Zabbix server]
* Commands	`python3 /etc/zabbix/frontendscripts/maintenance.py create '{HOST.HOST}' 3600`
Description	
User group	All
Host group	All
Required host permissions	[✓ Read] Write
Enable confirmation	✓
Confirmation text	Click execute to create a maintenance period of 1 hour for {HOST.HOST}

Figure 9.24 – Zabbix Administration | Scripts, create script page, Maintenance/Create/1 hour

24. Click on the blue **Add** button and then, on the next page, click the blue **Create script** button in the top-right corner again.

25. Add the second script as follows:

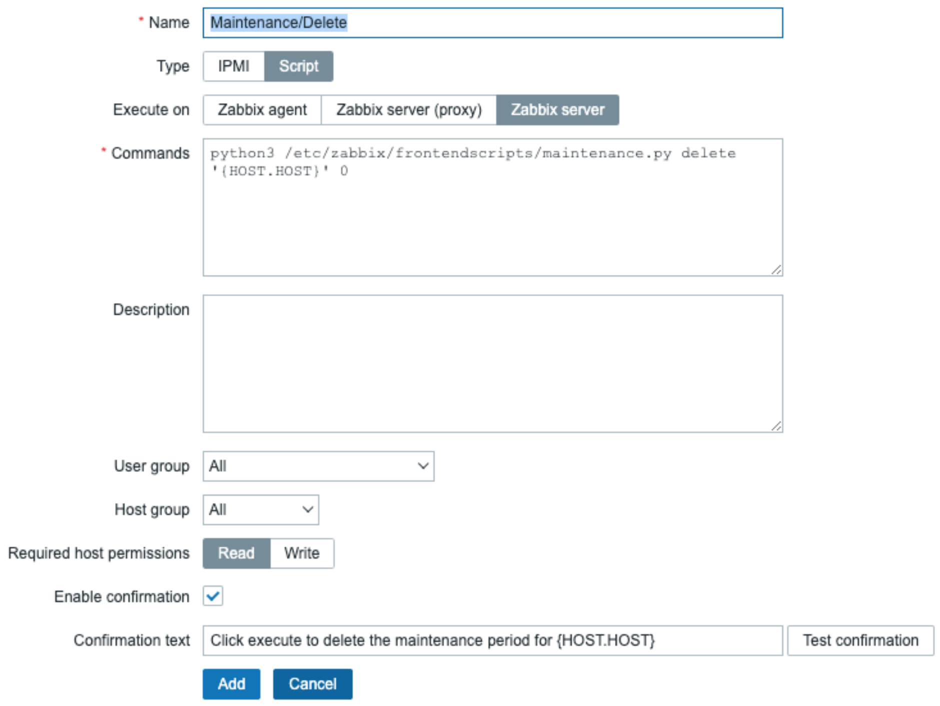


* Name	Maintenance/Create/24 hour
Type	IPMI [✓ Script]
Execute on	Zabbix agent Zabbix server (proxy) [✓ Zabbix server]
* Commands	`python3 /etc/zabbix/frontendscripts/maintenance.py create '{HOST.HOST}' 86400`
Description	
User group	All
Host group	All
Required host permissions	[✓ Read] Write
Enable confirmation	✓
Confirmation text	Click execute to create a maintenance period of 24 hours for {HOST.HOST}

Figure 9.25 – Zabbix Administration | Scripts, Create script page, Maintenance/Create/24 hour

26. Click on the blue **Add** button and then, on the next page, click the blue **Create script** button in the top-right corner again.

27. Add the third and final script as follows:



* Name	Maintenance/Delete
Type	IPMI [✓ Script]
Execute on	Zabbix agent, Zabbix server (proxy) [✓ Zabbix server]
* Commands	`python3 /etc/zabbix/frontendscripts/maintenance.py delete '{HOST.HOST}' 0`
Description	
User group	All
Host group	All
Required host permissions	[✓ Read] Write
Enable confirmation	✓
Confirmation text	Click execute to delete the maintenance period for {HOST.HOST}

Figure 9.26 – Zabbix Administration | Scripts, create script page, Maintenance/Delete

28. Now click the blue **Add** button to finish adding the final script.
29. Let's test the script by navigating to **Monitoring** | **Hosts**. Here we can click on any hostname available.
30. This will then open a drop-down menu where we can select to schedule maintenance for this host:

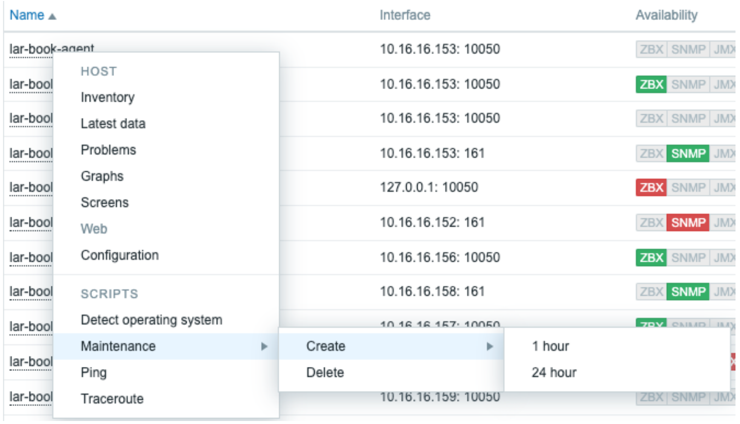
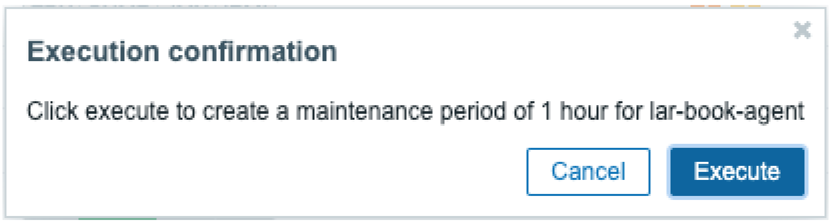


Figure 9.27 – Zabbix Monitoring | Hosts page with dropdown from underlined hostname

31. If we click on **1 hour**, we get a confirmation window to execute the scheduling of our maintenance period:

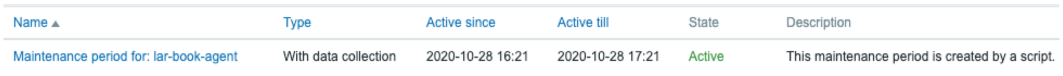


Execution confirmation

Click execute to create a maintenance period of 1 hour for lar-book-agent

Figure 9.28 – Zabbix confirmation window before execution

32. Click on **Execute** and then navigate to **Configuration | Maintenance**. We can now see our new maintenance period:



Name ▲	Type	Active since	Active till	State	Description
Maintenance period for: lar-book-agent	With data collection	2020-10-28 16:21	2020-10-28 17:21	Active	This maintenance period is created by a script.

Figure 9.29 – Zabbix Configuration | Maintenance page showing host lar-book-agent

How it works

The script we just used was built in Python utilizing the Zabbix API. With this script, we can now schedule maintenance periods from the Zabbix frontend as a Zabbix User.

This works because the **Monitoring | Hosts** option is available even to Zabbix Users. Instead of using the user's frontend permissions, our script uses the API user for execution. Because our Zabbix API user has more user permissions, it can execute the script that gets host information from a Zabbix database and creates a maintenance period using the information. Looking at the following diagram, we can see the process of this script:

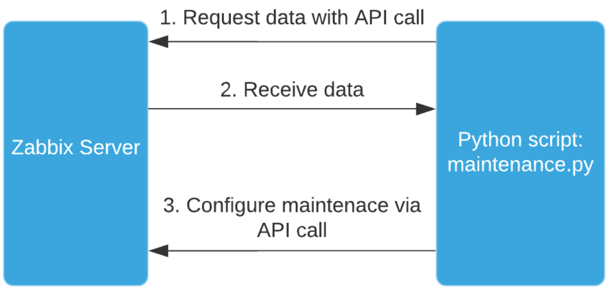

1. Request data with API call
2. Receive data
Zabbix Server
Python script: maintenance.py
3. Configure maintenace via API call

Figure 9.30 – Python script maintenance.py execution diagram

As we can see in the preceding diagram, our script follows roughly the same steps as other Zabbix API utilities. Since the Zabbix API is very flexible, we can pull data and write data to do almost anything we could do from the frontend.

We can now use this new function from anywhere in the Zabbix frontend where we see a dotted line with the hostname, even from Zabbix maps. We can see an example of when we clicked on such a dotted line in *Figure 9.27*.

See also

Brian van Baekel created the script used in this recipe for a customer at *Opensource ICT Solutions* and then open sourced it. Because Zabbix has a very cool community working to extend the possibilities of Zabbix even further, we too upload some of our scripts. Sharing is caring, so check out the other open-sourced scripts on the GitHub repo at `https://github.com/OpensourceICTSolutions`.

Enabling and disabling a host from Zabbix maps

I've noticed that it is not possible to enable and disable hosts as a Zabbix User. For some companies, this may be a requirement, so I've created an extension for it. In this recipe, I will show you just how to work with this Python script and execute it from a map.

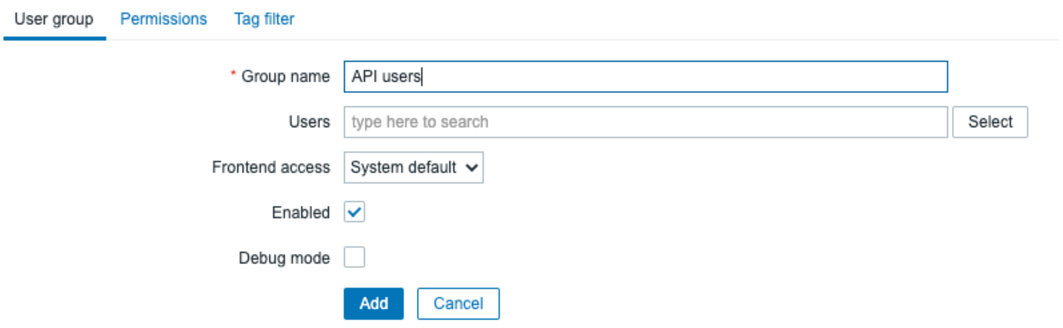
Getting ready

For this recipe, all we are going to need is our Zabbix server, some knowledge of Python, and some knowledge of the Zabbix API.

How to do it

1. If you haven't already created an API user, then note that we are going to need a new user for API calls, but to create this, we need a user group first. Click the blue **Create user group** button.

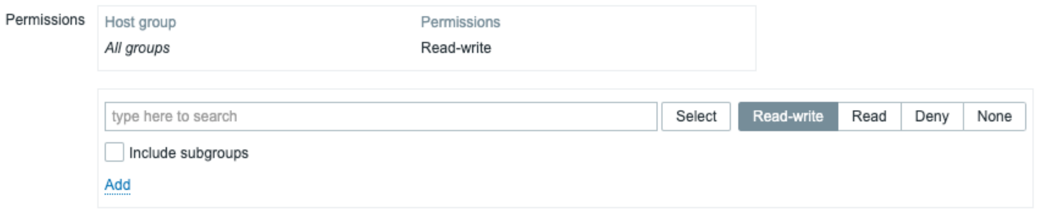
2. Now let's create the following **User group**:



* Group name	API users
Users	type here to search [Select]
Frontend access	System default
Enabled	✓
Debug mode	

Figure 9.31 – Zabbix Administration | Users group, Create user group page, API users for maps

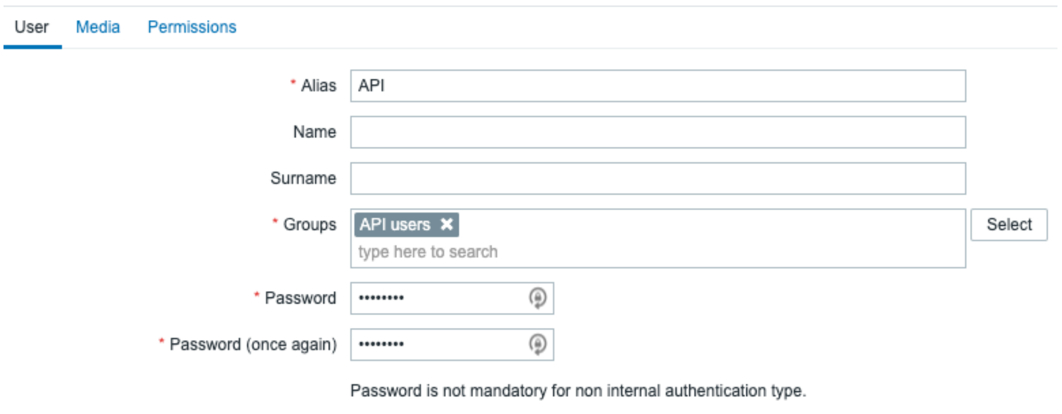
3. Now, let's click on **Permissions** and fill out some information here too.
4. Click on the white **Select** button and select all the user groups available in your Zabbix server. Then press the blue **Select** button.
5. Click on **Read-write** and then press the underlined **Add** button. It should now look like this:


Permissions

Host group	Permissions
All groups	Read-write

Figure 9.32 – Zabbix Administration | Users group, Create user group permissions page, API users for maps

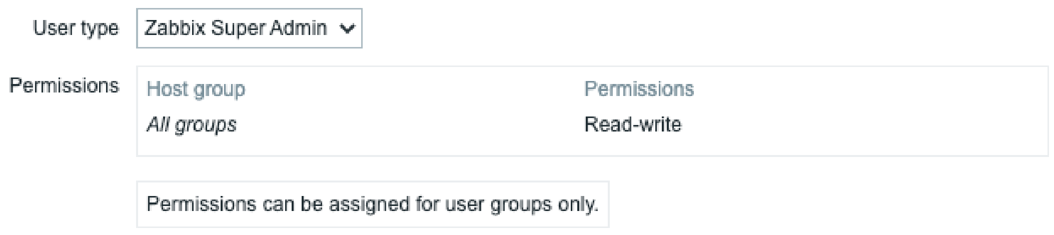
6. Notice how it says **All groups** with **Read-write** permissions. Now you can click the blue **Add** button at the bottom of the page.
7. Next up, go to **Administration** | **Users** and click on **Create user**. We'll create the following user:



* Alias	API
Name	
Surname	
* Groups	API users ✕ type here to search
* Password	••••••••
* Password (once again)	••••••••
	Password is not mandatory for non internal authentication type.

Figure 9.33 – Zabbix Administration | Users, Create user page for API user for maps

8. In general, you should always create a secure password for a production environment. For this lab environment, I'll just be using the password `password` though.
9. Let's edit the **Permissions** tab for the API user as well. Add the following:



User type: Zabbix Super Admin

Permissions:

Host group	Permissions
All groups	Read-write

Permissions can be assigned for user groups only.

Figure 9.34 – Zabbix Administration | Users, Create user Permissions page for API user for maps

10. Click on the blue **Add** button at the bottom of the page to finish creating this user.
11. Now, let's log in to our Zabbix server CLI and create a new directory:

```
mkdir /etc/zabbix/frontendscripts
```

12. Change to the new directory:

```
cd /etc/zabbix/frontendscripts
```

13. Now download the Public script from the *Opensource ICT Solutions* GitHub:

```
wget https://github.com/OpensourceICTSolutions/zabbix-
toggle-hosts-from-frontend/archive/v1.0.tar.gz
```

14. If you can't use `wget` from your host, you can check out the script here: `https://github.com/OpensourceICTSolutions/zabbix-toggle-hosts-from-frontend/releases/tag/v1.0`.
15. Unzip the file with the following command:

```
tar -xvzf v1.0.tar.gz
```

16. Remove the `tar` file using the following command:

```
rm v1.0.tar.gz
```

17. Move the script over from the newly created folder with the following command:

```
mv zabbix-toggle-hosts-from-frontend-1.0/enable_disable-
host.py ./
```

18. We are going to need Python to use this script, so let's install it as follows:

    For RHEL-based systems:

```
dnf install python3 python3-pip
```

    For Debian-based systems:

```
apt-get install python python-pip
```

19. We will also need the `requests` module from `pip`. Install it as follows:

```
pip3 install requests
```

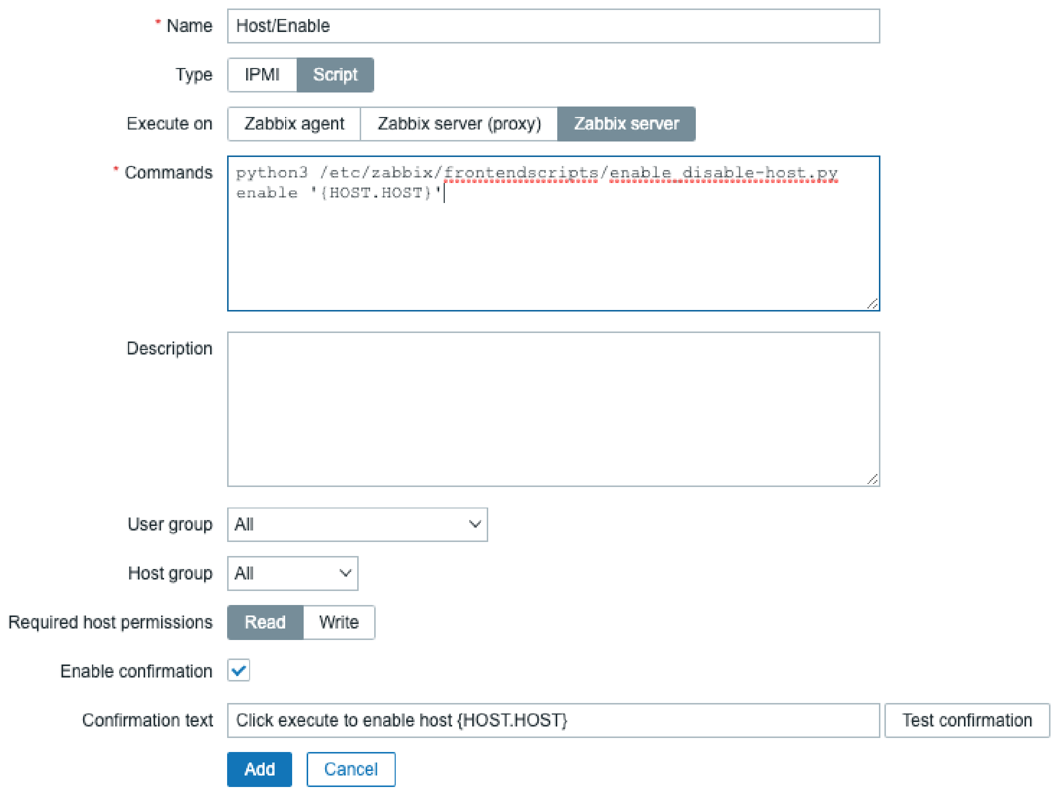
20. Now let's edit the script with the following command:

```
vim enable_disable-host.pyThere's more
```

21. In the section starting with `url`, we will need to enter our own setup information. I will fill in the following here, but be sure to enter your own information where relevant:

```
url = 'http://10.16.16.152/zabbix/api_jsonrpc.php?'
username = "API"
password = "password"
```

22. Now, we can move on to our Zabbix frontend to add a frontend script. Navigate to **Administration** | **Scripts** then click the blue **Create script** button in the top right.
23. Add the following script:

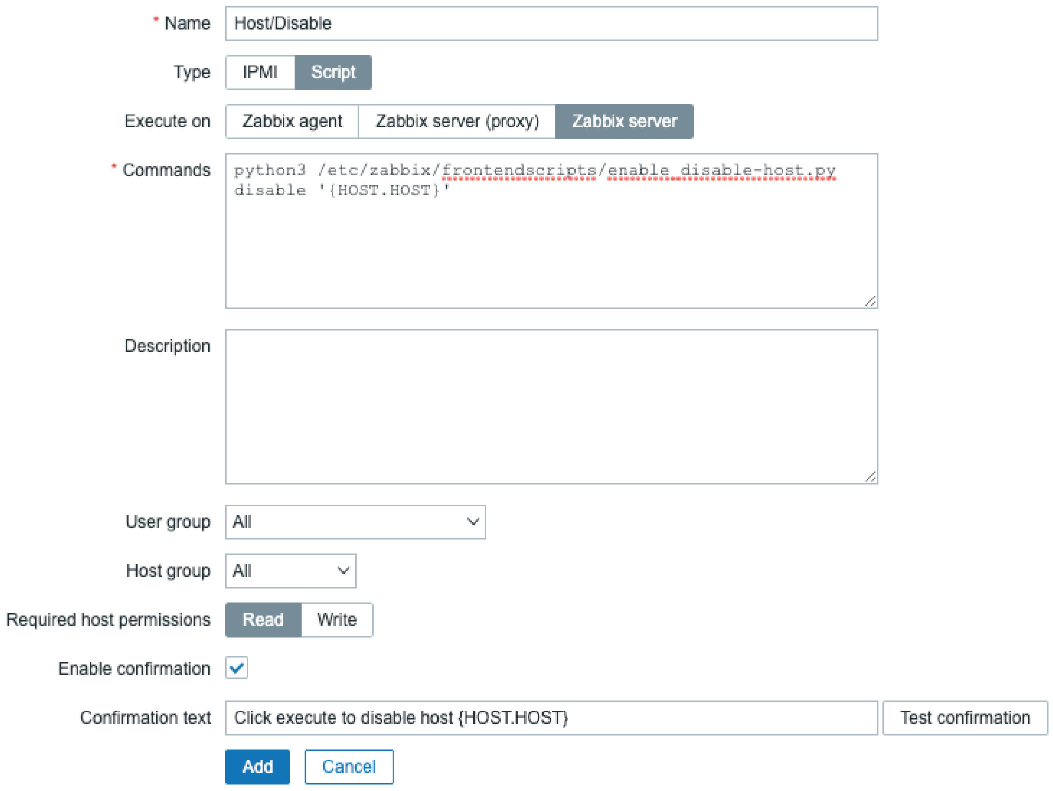


* Name	Host/Enable
Type	IPMI [✓ Script]
Execute on	Zabbix agent / Zabbix server (proxy) / [✓ Zabbix server]
* Commands	`python3 /etc/zabbix/frontendscripts/enable_disable-host.py enable '{HOST.HOST}'`
Description	
User group	All
Host group	All
Required host permissions	[✓ Read] Write
Enable confirmation	✓
Confirmation text	Click execute to enable host {HOST.HOST}

Figure 9.35 – Zabbix Administration | Scripts, create script page, Host/Enable

24. Click on the blue **Add** button and then, on the next page, click the blue **Create script** button in the top-right corner again.

25. Now add the second and final script as follows:



* Name	Host/Disable
Type	IPMI [✓ Script]
Execute on	Zabbix agent Zabbix server (proxy) [✓ Zabbix server]
* Commands	python3 /etc/zabbix/frontendscripts/enable_disable-host.py disable '{HOST.HOST}'
Description	
User group	All
Host group	All
Required host permissions	[✓ Read] Write
Enable confirmation	✓
Confirmation text	Click execute to disable host {HOST.HOST}

Figure 9.36 – Zabbix Administration | Scripts, Create script page, Host/Disable

26. Now navigate to **Monitoring** | **Maps**, and you should see a map called **Local network** here, as it's included with Zabbix by default. Click this map (or any other map with hosts in it).

27. Now, if you click on a host on the map, you will see a drop-down menu like this:

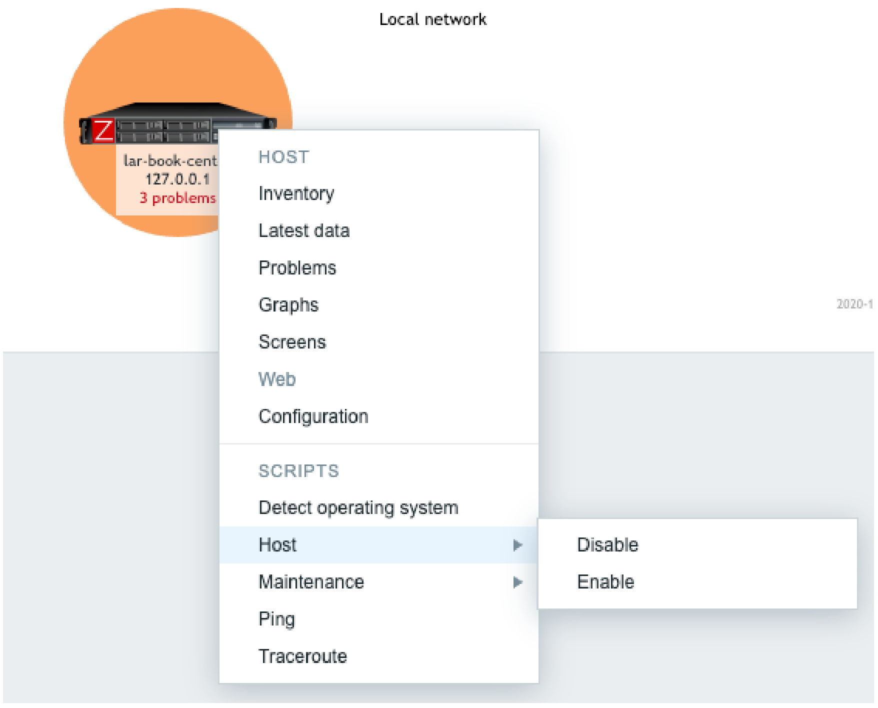
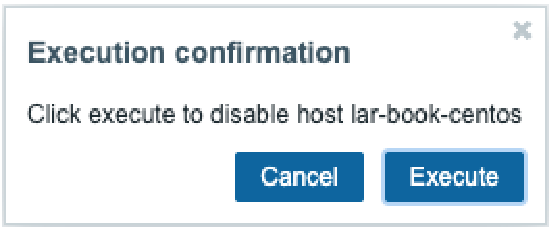


Figure 9.37 – Zabbix Monitoring | Maps, Local network map drop-down menu

28. If we click on **Disable** here, we will get a pop-up message as follows:


Execution confirmation

Click execute to disable host lar-book-centos

Figure 9.38 – Zabbix script confirmation window

29. Click on the blue **Execute** button and this host will be disabled. Navigate to **Monitoring** | **Hosts** to confirm if this worked. You should see that the host is **Disabled**.

30. Back at **Monitoring** | **Maps**, you can enable the host again with the same drop-down menu. This time, select **Enable**.

How it works

The script we just used was built in Python utilizing the Zabbix API. With this script, we can now enable and disable hosts from the Zabbix frontend as a Zabbix User.

This works because the **Monitoring** | **Maps** option is available even to Zabbix Users. This script uses the API user for execution though. Since our Zabbix API user has more user permissions, it can execute the script that gets host information from a Zabbix database and creates a maintenance period using the information. As we can see in the following diagram, our script follows roughly the same steps as the other Zabbix API utilities:

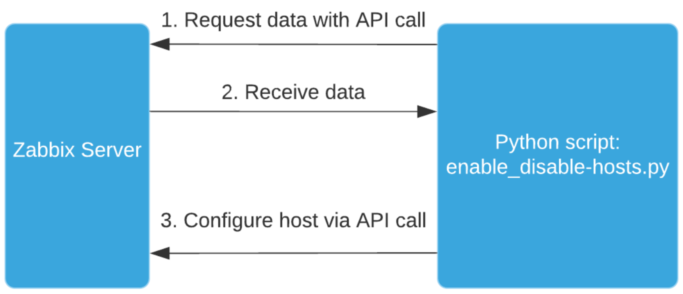

1. Request data with API call
2. Receive data
Zabbix Server
Python script: enable_disable-hosts.py
3. Configure host via API call

Figure 9.39 – Python script maintenance.py execution diagram

Because the Zabbix API is very flexible, we can pull data and write data to do almost anything we could do from the frontend.

We can now use this cool function from anywhere in the Zabbix frontend where we see a dotted line with the hostname, even from **Monitoring** | **Hosts**.

See also

Brian van Baekel created this script for a customer at *Opensource ICT Solutions* and then open sourced it. Because Zabbix has a very cool community that continues to extend the possibilities of Zabbix even further, we too upload some of our scripts. Sharing is caring, so check out the other open-sourced scripts at `https://github.com/OpensourceICTSolutions`.

10 Maintaining Your Zabbix Setup

Like any good piece of software, Zabbix needs to be maintained in order to keep working over the years. A lot of users have been running their setups since the days of Zabbix 2.0. It's perfectly viable to do this if you bring the right knowledge of Zabbix to the equation.

In this chapter, I am going to show you how to do some of the most important parts of Zabbix maintenance to make sure you can keep your setup available and running smoothly. We are going to cover creating maintenance periods, how to make backups, how to upgrade Zabbix and various Zabbix components, and how to do some performance maintenance.

We'll cover these in the following recipes:

- Setting Zabbix maintenance periods
- Backing up your Zabbix setup
- Upgrading the Zabbix backend from older PHP versions to PHP 7.2 or higher
- Upgrading a Zabbix database from older MariaDB versions to MariaDB 10.5
- Upgrading your Zabbix setup
- Maintaining Zabbix performance over time

Technical requirements

We are going to need several important servers for these recipes. First of all, we are going to need a running Zabbix 5 server for which to set up maintenance periods and do performance tuning.

We will need one of the following servers:

- A CentOS 7 server running Zabbix server 4.0 with PHP 5.4 and MariaDB 5.5
- An Ubuntu 16.04 server running Zabbix server 4.0 with PHP 7.0 and MariaDB 10.0

I will call this server `lar-book-zbx4`, which you can run with a distribution of your choice.

If you do not have any prior experience with Zabbix, this chapter may prove a good challenge, as we are going to go in depth into the more advanced Zabbix processes.

Setting Zabbix maintenance periods

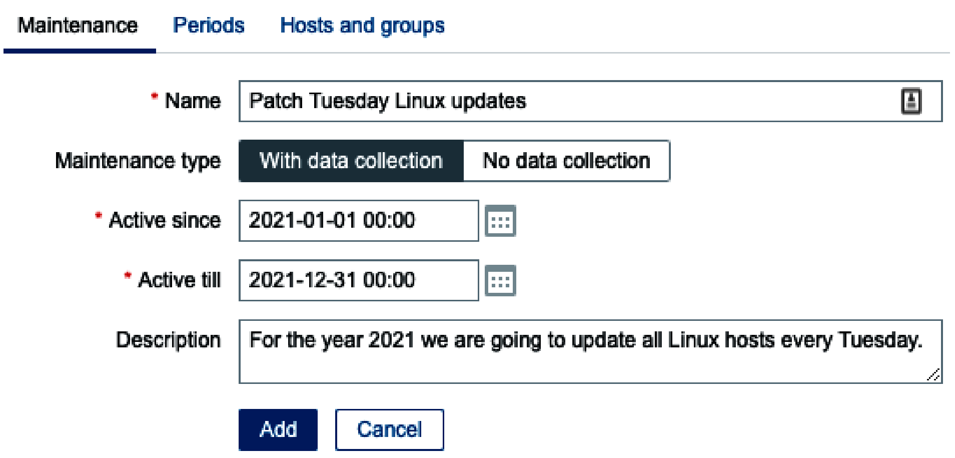
When we are working on our Zabbix server or on other hosts, it's super useful to set up maintenance periods in the Zabbix frontend. With maintenance periods we can make sure that our Zabbix users don't get alerts going off because of our maintenance. Let's see how we can schedule maintenance periods in this recipe.

Getting ready

All we are going to need in this recipe is our Zabbix server, for which I'll use `lar-book-centos`. The server will need at least some hosts and host groups to create maintenance periods on. Furthermore, we'll need to know how to navigate the Zabbix frontend.

How to do it…

1. Let's get started with this recipe by logging in to our frontend and navigating to **Configuration | Maintenance**.
2. We are going to click on the blue **Create maintenance period** button in the top-right corner.
3. This will bring us to the next page, where we can set up our maintenance period. On the first tab, **Maintenance**, fill in the following:

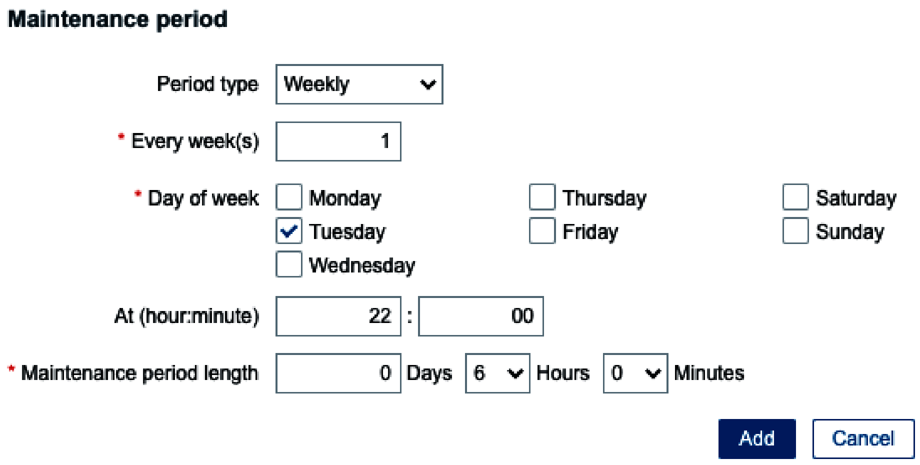


Maintenance | Periods | Hosts and groups

* Name	Patch Tuesday Linux updates
Maintenance type	[✓ With data collection] No data collection
* Active since	2021-01-01 00:00
* Active till	2021-12-31 00:00
Description	For the year 2021 we are going to update all Linux hosts every Tuesday.

Figure 10.1 – Zabbix Configuration | Maintenance, create maintenance page, Patch Tuesday

4. Now, click on the **Periods** tab to create a new maintenance period. We need to click on the underlined **Add** text.
5. This will bring us to a pop-up window where we can set the maintenance period. We need to fill in the following information:

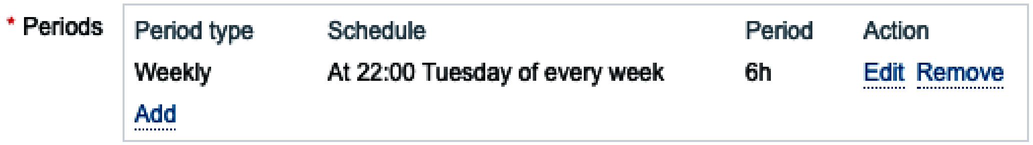


Maintenance period

Period type	Weekly
* Every week(s)	1
* Day of week	Monday, [✓ Tuesday], Wednesday, Thursday, Friday, Saturday, Sunday
At (hour:minute)	22 : 00
* Maintenance period length	0 Days 6 Hours 0 Minutes

Figure 10.2 – Zabbix Configuration | Maintenance, create maintenance period window, Patch Tuesday

6. Now press the blue **Add** button to continue. You should now see that our maintenance period is filled in:



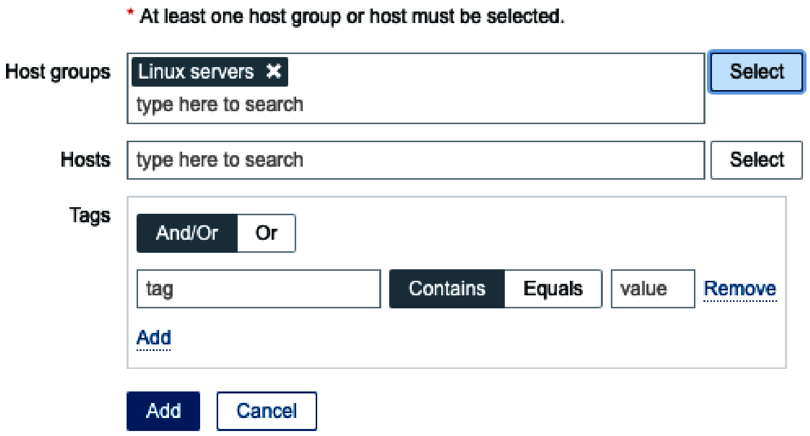
* Periods

Period type	Schedule	Period	Action
Weekly	At 22:00 Tuesday of every week	6h	Edit Remove
Add			

Figure 10.3 – Zabbix Configuration | Maintenance, create maintenance period page, Patch Tuesday

7. Now click on the **Hosts and groups** tab to fill out what hosts will be affected by the maintenance.
8. Next to **Host groups**, click on the **Select** button and select the host group **Linux servers**.

   Our page should look like this:



* At least one host group or host must be selected.

Host groups	Linux servers ✖ type here to search
Hosts	type here to search
Tags	And/Or, tag, Contains, value

Figure 10.4 – Zabbix Configuration | Maintenance, add hosts to maintenance page, Patch Tuesday

9. Now click on the blue **Add** button at the bottom of the page to finish creating the maintenance period. This will bring us back to our maintenance periods page, where we should see that our maintenance window has been created.

How it works

When configuring actions in Zabbix, we tell Zabbix to do a certain defined operation when a trigger is fired. Maintenance periods work by suppressing these Zabbix operations for the time period defined in the maintenance period. We do this to make sure that no Zabbix users are notified of any problems going on as maintenance is being done on a host. Of course, it's a good idea to only use this during the time that we are actually working on the hosts in question.

Tip

Like many configurations done from the Zabbix frontend, changes do not take effect until Zabbix server configuration cache reload. The maintenance period will not go into effect until the configuration cache is reloaded either automatically or manually.

In the case of this recipe, we've created a recurring maintenance period for the entire year, 2021. Let's say the organization we're working for has a lot of Linux hosts that need to be patched weekly. We set up the maintenance period to recur weekly every Tuesday between 22:00 and 04:00.

Now keep in mind that after December 31st, 2021, we will stop this maintenance period as it won't be active any longer. We have two time/date values to bear in mind when setting up scheduled maintenance. The **active since/active till** time/date value of the maintenance period and the period's time/date value of the maintenance period. This allows us to create more flexible periods and recurring ones like we just did.

Also, take note that this maintenance period is **With data collection**. We can also create a maintenance period with the option of **No data collection**. When we use the **With data collection** option, we will keep collecting data but won't send any problems to Zabbix Users. If we want to stop collecting the data, simply select the **No data collection** option.

Backing up your Zabbix setup

Before working on any Zabbix setup, it is vital to make a backup of everything important. In this recipe, I will take you through some of the most important steps you should always take before doing maintenance on your Zabbix setup.

Getting ready

We are going to need our Zabbix server, for which I'll use `lar-book-centos`. Make sure to get the CLI to the server ready, as this whole recipe will use the Linux CLI.

How to do it

1. Let's start by logging in to our Zabbix server via the Linux CLI and create some new directories that we are going to use for our Zabbix backups. Preferably, this directory would be on another hard disk:

```
mkdir /opt/zbx-backup/
mkdir /opt/zbx-backup/database/
mkdir /opt/zbx-backup/zbx-config/
mkdir /opt/zbx-backup/web-config/
mkdir /opt/zbx-backup/shared/
mkdir /opt/zbx-backup/shared/zabbix/
mkdir /opt/zbx-backup/shared/doc/
```

2. It's important to back up all of our Zabbix configuration data, which is located in `/etc/zabbix/`. We can manually copy the data from our current folder to our new backup folder by issuing the following command:

```
cp -r /etc/zabbix/ /opt/zbx-backup/zbx-config/
```

3. Now, let's do the same for our `httpd` configuration:

```
cp /etc/httpd/conf.d/zabbix.conf /opt/zbx-backup/
web-config/
```

Important note

Please note that if you are using NGINX or Apache on a Debian-based system, your web configuration location might be different. Adjust your command accordingly.

4. It's also important to keep our Zabbix PHP files and binaries backed up. We can do that using the following commands:

```
cp -r /usr/share/zabbix/ /opt/zbx-backup/shared/zabbix/
cp -r /usr/share/doc/zabbix-* /opt/zbx-backup/shared/doc/
```

5. We could also create a cronjob to automatically compress and back up these files for us every day at `00:00`. Simply issue the following command:

```
crontab -e
```

6. And add the following information:

   ```
   * 0 * * * tar -zcvf /opt/zbx-backup/zbx-config/zabbix.
   tar.gz /etc/zabbix/ >/dev/null 2>&1
   ```

   ```
   * 0 * * * tar -zcvf /opt/zbx-backup/web-config/zabbix-
   web.tar.gz /etc/httpd/conf.d/zabbix.conf >/dev/null 2>&1
   ```

   ```
   * 0 * * * tar -zcvf /opt/zbx-backup/shared/zabbix/zabbix_
   usr_share.tar.gz /usr/share/zabbix/ >/dev/null 2>&1
   ```

   ```
   * 0 * * * tar -zcvf /opt/zbx-backup/shared/doc/zabbix_
   usr_share_doc.tar.gz /usr/share/doc/ >/dev/null 2>&1
   ```

7. These are all of the most important files we need to back up from our Zabbix stack. Let's move on to our database. We could now additionally use a rotation tool such as `logrotate` to manage our files.

> **Important note**
>
> Using cronjobs to back up your files is a good setup and definitely better than nothing, but it's not the best solution. It's recommended to use a solid script or tool to do the backup procedures.

8. Backing up our database is quite easy. We can simply use the built-in tools provided by MySQL and PostgreSQL. Issue the following command for your respective database:

   For MySQL databases:

   ```
   mysqldump -u zabbixuser -p zabbixdb > /opt/zbx-backup/
   database/backup_zabbixDB_date.sql
   ```

   For PostgreSQL databases:

   ```
   pg_dump zabbixdb > /opt/zbx-backup/database/backup_
   zabbixDB_date.bak
   ```

9. Make sure to add the right location, as the database dump will be quite large if the database itself is large. Preferably, dump to another disk or even better, another machine.

10. We can also do this with a cronjob by issuing the following command:

    ```
    crontab -e
    ```

11. Then for MySQL, add the following line where `-u` is the username, `-p` is the password, and the database name is `zabbix`. This is the command for MySQL:

```
* * * * 6 mysqldump -u'zabbixuser' -p'password' zabbixdb
> /opt/zbx-backup/database/backup_zabbixDB.sql
```

12. If you want to back up a PostgreSQL database with a cronjob, we will need to create a file in our user's home directory:

```
vim /home/USERNAME/.pgpass
```

13. We add the following to this file, where `zabbixuser` is the username and `zabbixdb` is the database name:

```
#hostname:port:database:username:password
```

```
localhost:5432:zabbixuser:zabbixdb:password
```

14. Then we can add a cronjob for PostgreSQL as follows:

```
* * * * 6 pg_dump --no-password -U zabbixuser zabbixdb >
/opt/zbx-backup/database/backup_zabbixDB_date.bak
```

15. We can also add a cronjob to only keep a certain number of days' worth of backups. Issue:

```
crontab -e
```

16. Then add the following line, where `+60` is the number of days you want to keep backups for:

```
* * * * 6 find /opt/zbx-backup/database/ -mtime +60 -type
f -delete
```

17. That concludes our demonstration of backing up our Zabbix components the easy way.

Important note

Although using cronjobs to back up your MySQL or PostgreSQL database is a pretty good method for backing up the database, there's definitely a better way to make backups. Consider using a solid script or application that handles the backup procedure.

How it works

A Zabbix setup consists of several components. We have the Zabbix frontend, Zabbix server, and Zabbix database. These components in this setup require different pieces of software to run on, as shown in the following diagram:

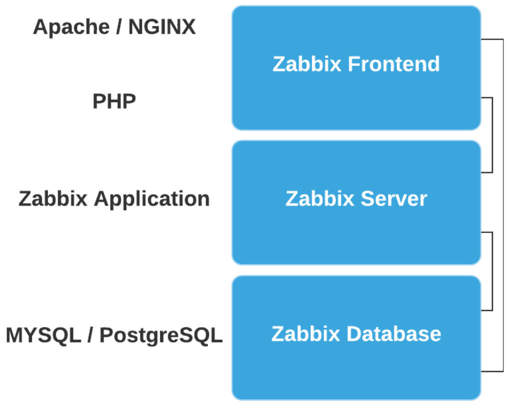


Figure 10.5 – Zabbix key components setup diagram

Looking at the preceding diagram, we can see that our Zabbix frontend runs on a web engine such as Apache or NGINX. We also need PHP to run our Zabbix web pages. This means that we have to back up two components:

- The web engine: Apache, NGINX, or another
- PHP

The Zabbix server is a proprietary application designed by Zabbix, so we only need to back up one thing here:

- The Zabbix server config files

Then last, but definitely not least, we need to make a backup of our database. The most common databases used are MySQL and PostgreSQL, so we only need to do one thing for this:

- Create a dump of the Zabbix database.

There's more

Backing up your Zabbix setup like this is one thing, but of course, it's not everything. Make sure you have the correct backups of your Linux system, using snapshots and other technologies.

When you follow standard backup implementations, you should be prepared against any unforeseen circumstances with your Zabbix setup.

Upgrading the Zabbix backend from older PHP versions to PHP 7.2 or higher

When we are upgrading a Zabbix setup from Zabbix 4 to Zabbix 5, you might run into the issue that PHP 7.1 or an earlier version is installed. One of the first steps we need to do to upgrade our Zabbix setup is to make sure PHP 7.2 or higher is installed. So, let's make sure we are ready by installing PHP 7.2.

Getting ready

For this recipe, we will need our server called `lar-book-zbx4`. At this point, the server installed with CentOS 7 will be running Zabbix server 4.0 with PHP version 5.4.

Another working option is to have a server running Ubuntu 16.04, which already includes PHP version 7 by default. As we need version 7.2 or higher, we will upgrade this one to PHP 7.2 as well.

Also, make sure to take backups of your system and read the release notes for the new version you're installing.

How to do it

1. First things first, let's log in to our host CLI to check out our versions. Issue the following commands:

   For Zabbix server:

   ```
   zabbix_server --version
   ```

   For PHP:

   ```
   php --version
   ```

   For MariaDB:

   ```
   mysql --version
   ```

2. Verify that our versions match the versions from the *Getting ready* section of this recipe.

With that done, let's move on to upgrade the PHP version on our host.

CentOS 7

1. First, remove your old PHP versions using the following command:

```
yum remove php php-fpm
```

2. Now install the correct repo on our machine using the following command:

```
yum install centos-release-scl
yum clean all
```

3. We can now update our system to PHP 7.2 with the following command:

```
yum install rh-php72
```

4. When we check the version of PHP, it should now be 7.2:

```
php --version
```

Ubuntu 16.04

1. Let's start by adding the PPA repository to our host with the following command:

```
sudo add-apt-repository ppa:ondrej/php
```

2. Now update the repositories with the following command:

```
sudo apt update
```

3. On some installations, the key of the repository might not be available, so you might see an error reading `key is not available`. You can fix this with the following command, where `PUB_KEY_HERE` is the key shown in the error:

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-
keys PUB_KEY_HERE
```

4. Now we can install PHP version 7.2 with the following command:

```
sudo apt install -y php7.4 php7.4-fpm php7.4-mysql
php7.4-bcmath php-7.4-mbstring php-7.4-gd php-7.4-xml
```

5. That's it, the version of PHP should now be the one we want. Check the version of PHP with the following command:

```
php --version
```

How it works

Because Zabbix server 5 requires us to install PHP version 7.2 or higher, we need to upgrade the PHP version first. Doing so will not break our current Zabbix server 4.0 installation, as PHP is backward-compatible.

Now that we have upgraded PHP, we are ready to move on to upgrading the Zabbix database engine.

Upgrading a Zabbix database from older MariaDB versions to MariaDB 10.5

Although not always a requirement, it is a good idea to keep your database version up to date. MariaDB regularly makes improvements to how MySQL handles certain aspects of performance.

In this recipe, I will show you how to upgrade MariaDB to the now-latest version 10.5.

Getting ready

For this recipe, we will need our server called `lar-book-zbx4`. At this point, the server installed with CentOS 7 will be running Zabbix server 4.0 with MariaDB version 5.5.

Another option is that you have a server running Ubuntu 16.04, which already includes MariaDB version 10.0 by default. We will be upgrading the MariaDB instance on this server to version 10.5.

If you've followed the *Upgrading the Zabbix backend from older PHP versions to PHP 7.2 or higher* recipe, your server will now be running PHP version 7.2. If not, it's wise to follow that recipe first.

Also, make sure to take backups of your system and read the release notes for the new version you're installing.

How to do it

1. First things first, let's log in to our Linux host CLI to check out our versions. Issue the following commands:

   For Zabbix server:

   ```
   zabbix_server --version
   ```

   For PHP:

   ```
   php --version
   ```

   For MariaDB:

   ```
   mysql --version
   ```

2. After verifying our versions match the versions mentioned in the *Getting ready* section of this recipe, let's move on to upgrade our version of MariaDB.

CentOS 7

1. On our CentOS 7 server, the first thing we'll do after checking the versions is stop our MariaDB server:

   ```
   systemctl stop mariadb.service
   ```

2. Now set up a repository file for MariaDB with the following command:

   ```
   vim /etc/yum.repos.d/MariaDB.repo
   ```

3. We will add the following code to this new file. Make sure to add the correct architecture after `baseurl` if using anything other than `amd64`:

   ```
   # MariaDB 10.5 CentOS repository list
   # http://downloads.mariadb.org/mariadb/repositories/
   [mariadb]

   name = MariaDB
   baseurl = http://yum.mariadb.org/10.5/centos7-amd64
   gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
   gpgcheck=1
   ```

4. Now upgrade your MariaDB server with the following command:

   ```
   yum install MariaDB-client MariaDB-server
   ```

5. Restart the MariaDB service with the following command:

   ```
   systemctl start mariadb
   ```

6. That's it, MariaDB should now be upgraded to the correct version. Check the version again with the following command to make sure:

   ```
   mysql --version
   ```

Ubuntu 16.04

1. On our Ubuntu 16.04 server, the first thing we'll do after checking the versions is stop our MariaDB server:

   ```
   systemctl stop mysql.service
   ```

2. We can use the MariaDB repository setup script to update to the right repository. Execute the following command:

   ```
   curl -sS https://downloads.mariadb.com/MariaDB/mariadb_
   repo_setup | sudo bash
   ```

3. This will add the latest MariaDB package to the `/etc/apt/sources.list.d/mariadb.list` file. To see if it is on version 10.5 check it with the following command:

   ```
   vim /etc/apt/sources.list.d/mariadb.list
   ```

4. The file should look like the following code block. If it doesn't look right, edit it to match. Make sure to add the correct architecture on the `deb` lines if using anything other than `amd64`:

   ```
   # MariaDB Server
   # To use a different major version of the server, or to
   pin to a specific minor version, change URI below.
   deb [arch=amd64] http://downloads.mariadb.com/MariaDB/
   mariadb-10.5/repo/ubuntu xenial main

   # MariaDB MaxScale
   # To use the latest stable release of MaxScale, use
   ```

```
"latest" as the version
# To use the latest beta (or stable if no current beta)
release of MaxScale, use "beta" as the version
deb [arch=amd64] http://downloads.mariadb.com/MaxScale/
latest/ubuntu xenial main

# MariaDB Tools
deb [arch=amd64] http://downloads.mariadb.com/Tools/
ubuntu xenial main
```

5. We need to remove our old MariaDB packages with the following command:

```
apt-get remove --purge mariadb-server mariadb-client
zabbix-server-mysql
```

6. Now upgrade the MariaDB server version with the following command:

```
sudo apt-get install mariadb-server mariadb-client
zabbix-server-mysql
```

7. That's it, MariaDB should now be upgraded to the correct version. Check the version again with the following command:

```
mysql --version
```

How it works

Now, while it might not be a requirement, it is a smart idea to upgrade your database version regularly. New versions of your database engine might include improvements to stability and performance, both of which could improve your Zabbix server greatly.

Do keep the release notes and bug reports on your scope though. MariaDB 10.5 is, at the moment of writing, the newest version on the market. You might want to stick back one or two releases as these are still in support and have been running in production for a while already. After all, nobody likes unforeseen issues such as bugs.

Upgrading your Zabbix setup

As we've seen throughout the book already, Zabbix 5 offers a great deal of cool new features. Zabbix 5.0 is a **Long-Term Support** (**LTS**) release, so just like 3.0 and 4.0, you will receive long-term support for it. Let's see how we can upgrade a Zabbix server from version 4.0 to version 5.0.

Getting ready

For this recipe, we will need our server called `lar-book-zbx4`. At this point, your server installed with either CentOS 7 or Ubuntu 16.04 will be running Zabbix server 4.0.

If you've followed the *Upgrading the Zabbix backend from older PHP versions to PHP 7.2 or higher* recipe, your server will now be running PHP version 7.2. If not, it's wise to follow that recipe first.

If you've followed the *Upgrading a Zabbix database from older MariaDB versions to MariaDB 10.5* recipe, it will now be running MariaDB version 10.5. If not, it's wise to follow that recipe first.

Also, make sure to take backups of your system and read the release notes for the new version you're installing.

How to do it

1. First things first, let's log in to our Linux host CLI to check out our versions. Issue the following commands to do so:

   For Zabbix server:

   ```
   zabbix_server --version
   ```

   For PHP:

   ```
   php --version
   ```

   For MariaDB:

   ```
   mysql --version
   ```

2. After verifying our versions match the versions mentioned in the *Getting ready* section of this recipe, let's move on to upgrade our Zabbix server.

CentOS 7

1. First, let's stop our Zabbix server components with the following command:

   ```
   systemctl stop zabbix-server zabbix-agent
   ```

2. On our server, let's issue the following command to add the new Zabbix 5.0 repository:

   ```
   rpm -Uvh https://repo.zabbix.com/zabbix/5.0/rhel/7/
   x86_64/zabbix-release-5.0-1.el7.noarch.rpm
   ```

3. Run the following command to clean the repositories:

```
yum clean all
```

4. Now upgrade the Zabbix setup with the following command:

```
yum upgrade zabbix-server-mysql zabbix-web-mysql zabbix-
agent
```

5. Start the Zabbix components with the following command:

```
systemctl restart zabbix-server zabbix-agent
```

6. When we check if the server is running, it should say `Active (running)` when we issue the following command:

```
systemctl status zabbix-server
```

7. If not, we check the logs with the following command, so we can see what is happening:

```
tail /var/log/zabbix/zabbix_server.log
```

8. Now, check whether you see an error that looks like this:

```
25615:20201029:155200.045 [Z3005] query failed: [1118]
Row size too large. The maximum row size for the used
table type, not counting BLOBs, is 8126. This includes
storage overhead, check the manual. You have to change
some columns to TEXT or BLOBs [alter table `media_type`
add `event_menu_name` varchar(255) default '' not null]
```

9. To fix this error, log in to MariaDB with the following command:

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

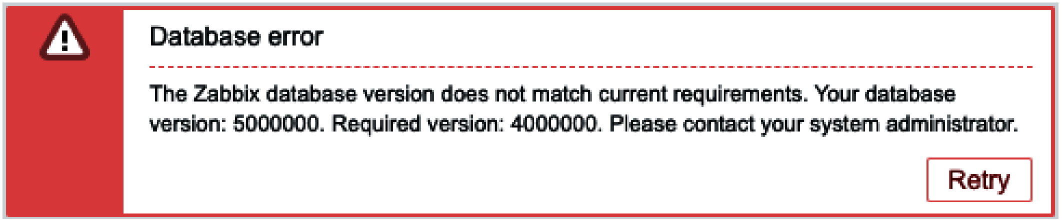
10. Then issue the following command:

```
SET GLOBAL innodb_strict_mode=OFF;
```

11. If we start the server again, this error should be gone and Zabbix server should keep running:

```
systemctl restart zabbix-server
```

12. Now when we navigate to the frontend, we might see this:


Database error

The Zabbix database version does not match current requirements. Your database version: 5000000. Required version: 4000000. Please contact your system administrator.

Figure 10.6 – Zabbix frontend error message, database version mismatch higher

13. This means our frontend version doesn't match the upgraded Zabbix database. Let's remove our old frontend by issuing the following command:

```
yum remove zabbix-web-*
```

14. Now edit the repository information to allow us to install the new frontend. Edit the repository information with the following command:

```
vim etc/yum.repos.d/zabbix.repo
```

15. Edit the `[zabbix-frontend]` code block to have `enabled=1`. It will look like this:

```
[zabbix-frontend]
name=Zabbix Official Repository frontend - $basearch
baseurl=http://repo.zabbix.com/zabbix/5.0/
rhel/7/$basearch/frontend
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-ZABBIX-
A14FE591
```

16. Then, upgrade the frontend with the following commands:

```
yum clean all
yum install zabbix-web-mysql-scl
```

17. Do not forget to update the time zone of the `php-fpm` module with the following command:

```
vim /etc/opt/rh/rh-php72/php-fpm.d/zabbix.conf
```

18. Edit the following line to match your time zone:

```
php_value[date.timezone] = Europe/Riga
```

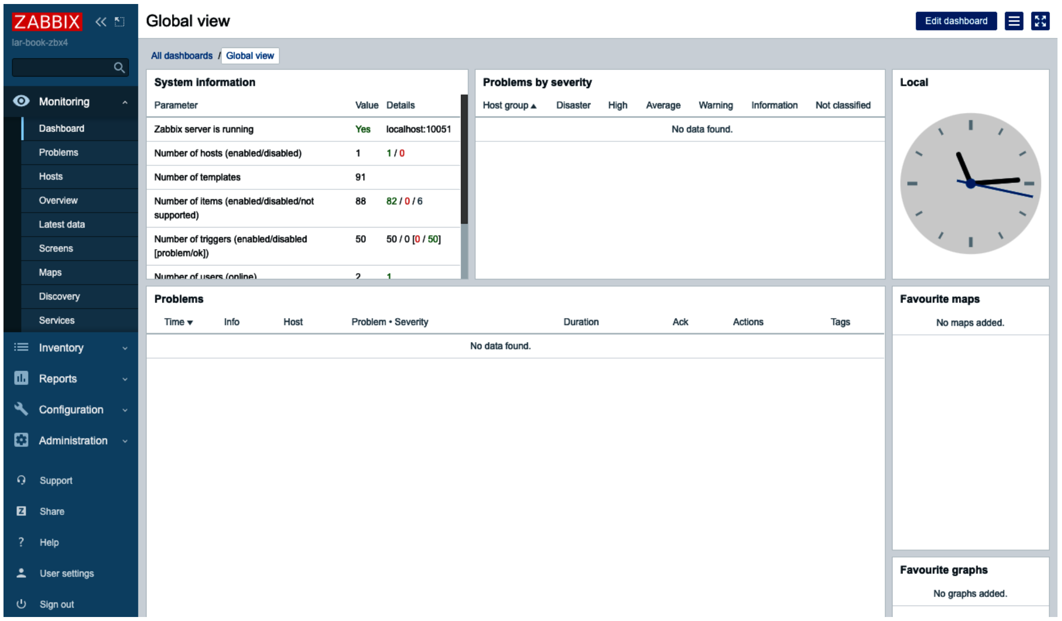
19. We also need to install a new package for the web config with the following command:

```
yum install zabbix-apache-conf-scl
```

20. Now restart the Zabbix components with the following command:

```
systemctl restart httpd rh-php72-php-fpm zabbix-server
mariadb
```

21. Now everything should be working as expected and we should see the new Zabbix 5 frontend, as shown in the following screenshot:



System information

Parameter	Value	Details
Zabbix server is running	Yes	localhost:10051
Number of hosts (enabled/disabled)	1	1 / 0
Number of templates	91	
Number of items (enabled/disabled/not supported)	88	82 / 0 / 6
Number of triggers (enabled/disabled [problem/ok])	50	50 / 0 [0 / 50]
Number of users (online)	2	1

Problems by severity

Host group ▲	Disaster	High	Average	Warning	Information	Not classified

No data found.

Problems

Time ▾	Info	Host	Problem • Severity	Duration	Ack	Actions	Tags

No data found.

Favourite maps

No maps added.

Favourite graphs

No graphs added.

Figure 10.7 – Zabbix 5 frontend after upgrade Centos

Ubuntu 16.04

1. First, let's stop our Zabbix server components with the following command:

```
systemctl stop zabbix-server zabbix-agent
```

2. Now add the new repository for Zabbix 5.0 on Ubuntu 16.04 with the following commands:

```
wget https://repo.zabbix.com/zabbix/5.0/ubuntu/pool/
main/z/zabbix-release/zabbix-release_5.0-1+xenial_all.
deb
```

```
dpkg -i zabbix-release_5.0-1+xenial_all.deb
```

3. Update the repository information with the following command:

```
apt-get update
```

4. Now upgrade the Zabbix server components with the following command:

```
apt-get install --only-upgrade zabbix-server-mysql
zabbix-frontend-php zabbix-agent
```

5. Make sure to not overwrite your Zabbix server configuration. If you do overwrite your configuration file, you can restore it from the backup taken in the *Backup up your Zabbix setup* recipe.
6. Now remove the old Zabbix Apache configuration with the following command:

```
apt-get remove zabbix-frontend-php
```

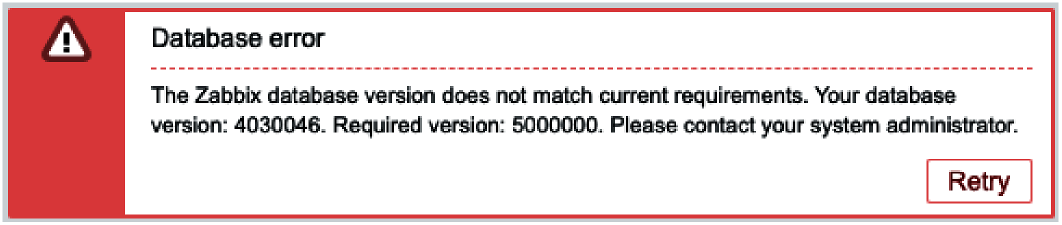
7. Then install the new Zabbix Apache configuration for older Ubuntu versions with the following command:

```
apt-get install zabbix-frontend-php-deprecated
```

8. Restart the Zabbix server components with the following command and you should be done:

```
systemctl restart zabbix-server httpd
```

9. You might still run into a database upgrade issue as shown in the following screenshot:


Database error

The Zabbix database version does not match current requirements. Your database version: 4030046. Required version: 5000000. Please contact your system administrator.

Figure 10.8 – Zabbix frontend error message, database version mismatch lower

10. We check the logs with the following command so we can see what is happening:

```
tail /var/log/zabbix/zabbix_server.log
```

11. We are looking for a message that looks like the following message:

```
17201:20201030:111123.704 [Z3005] query failed: [1118]
Row size too large. The maximum row size for the used
table type, not counting BLOBs, is 8126. This includes
storage overhead, check the manual. You have to change
some columns to TEXT or BLOBs [alter table `media_type`
add `event_menu_name` varchar(255) default '' not null]
```

12. To fix this error, log in to MariaDB with the following command:

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

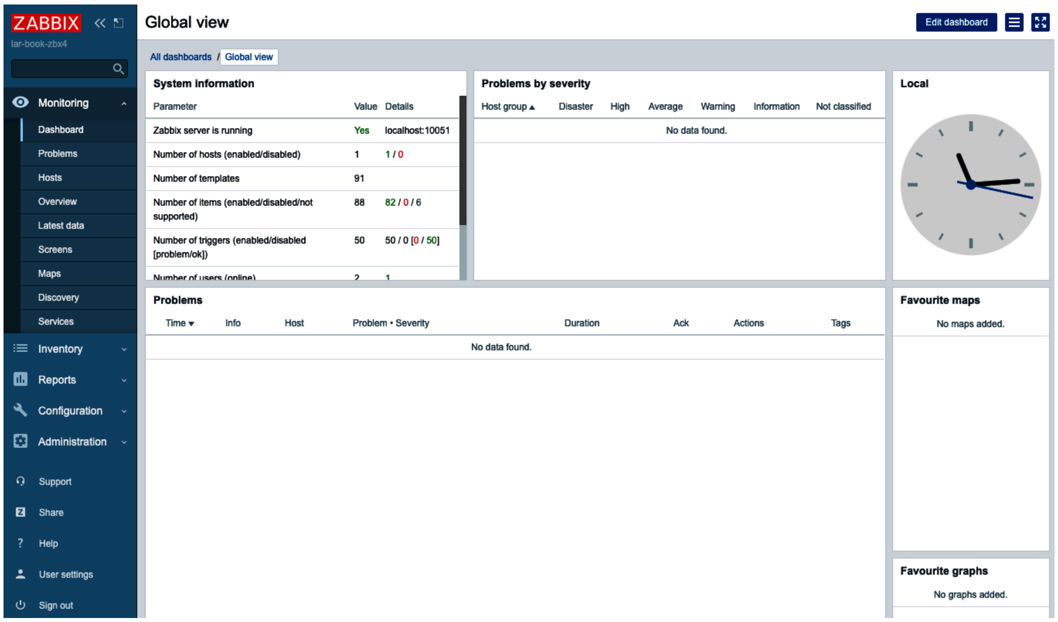
13. Then issue the following command:

```
SET GLOBAL innodb_strict_mode=OFF;
```

14. If we start the server again, this error should be gone and Zabbix server should keep running:

```
systemctl restart zabbix-server
```

15. This should conclude the upgrade process, and if we go to the frontend, we should see the new Zabbix 5 frontend:



System information

Parameter	Value	Details
Zabbix server is running	Yes	localhost:10051
Number of hosts (enabled/disabled)	1	1 / 0
Number of templates	91	
Number of items (enabled/disabled/not supported)	88	82 / 0 / 6
Number of triggers (enabled/disabled [problem/ok])	50	50 / 0 [0 / 50]
Number of users (online)	2	1

Problems by severity

Host group ▲	Disaster	High	Average	Warning	Information	Not classified

No data found.

Problems

Time ▼	Info	Host	Problem • Severity	Duration	Ack	Actions	Tags

No data found.

Favourite maps

No maps added.

Favourite graphs

No graphs added.

Figure 10.9 – Zabbix 5 frontend after upgrade Ubuntu

How it works

Upgrading Zabbix can be an easy task when we are running the latest version of Linux. When we are running something older though, we might run into some issues. Companies are still running CentOS 7 and Ubuntu 16.04, though, and some companies just aren't comfortable upgrading yet.

The recipe we've just followed shows us the upgrade process for these two older versions, along with the most common issues we might run into.

> **Important note**
>
> While upgrading, make sure to keep an eye on your `zabbix_server.log` file, as this file will tell you if something is wrong during the upgrade process.

Once you've moved away from the PHP versions lower than 7.2 though, the upgrade process to Zabbix 5 is quite straightforward. This requirement for the upgrade process is the biggest change coming from Zabbix 4 to Zabbix 5.

Now that you've upgraded all the components, you should be ready to work with Zabbix 5 and your setup will be futureproof for a while – of course, until Zabbix 6 comes out, when we might see some new requirements coming up.

See also

Make sure to check out the Zabbix documentation for the versions you are upgrading from and to. Zabbix always includes detailed descriptions of the requirements and processes to make it as easy as possible for you to upgrade. Check out the right documentation for your version at `https://www.zabbix.com/documentation/current/manual/installation/upgrade`.

Maintaining Zabbix performance over time

It's important to make sure that your Zabbix setup keeps performing well over time. There are several key components that are important to keep your Zabbix setup performing optimally over time. Let's see how to work on some of these components and keep your Zabbix setup running smoothly.

Getting ready

All we are going to need for this recipe is a Zabbix 5 server configured with either a MySQL or PostgreSQL database backend.

How to do it

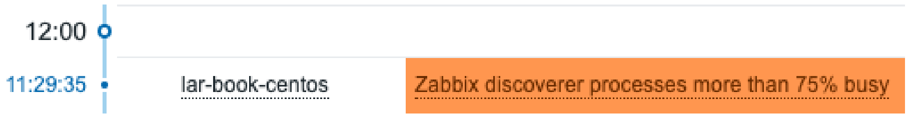
We will go through three of the main problems people face whilst maintaining Zabbix server performance. First things first, let's look at the Zabbix processes and how to edit them.

Zabbix processes

A regular problem people face is a Zabbix process being too busy. Let's log in to our Zabbix frontend and check out how this problem might look.

First, let's start by logging in to our Zabbix server frontend and check out some messages:

1. When we navigate to **Monitoring** | **Dashboard** and then select the default dashboard **Global view**, we might see something like this:



12:00

11:29:35 lar-book-centos Zabbix discoverer processes more than 75% busy

Figure 10.10 – Zabbix problem from our Zabbix server, discoverer process 75% busy

2. Then we navigate to **Monitoring** | **Hosts** and click on **Latest data** for the Zabbix server host (in my case, called `lar-book-centos`). This will take us to the latest data for our host.
3. For the filters, type `discoverer` in the **Name** field, then click on **Graph**. This will show you the following graph:

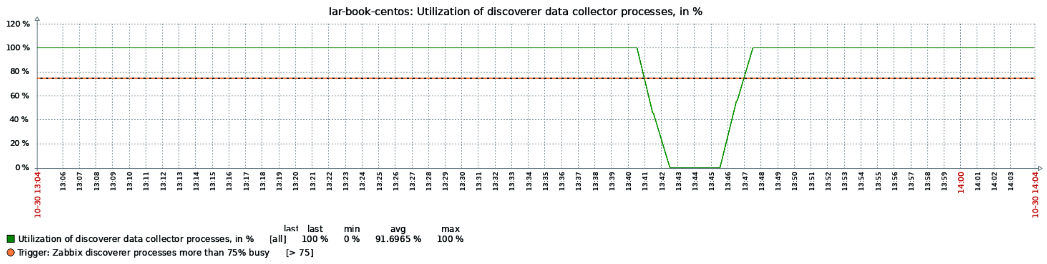


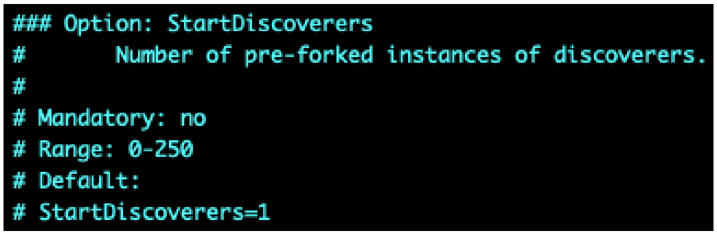
Figure 10.11 – Zabbix server discoverer graph, Utilization of discover data collector in %

This graph is at 100% almost all the time, which explains why we see the problem shown in *Figure 10.10* on our dashboard.

4. Let's log in to the Linux CLI of our Zabbix server to edit this process.
5. Edit the following file on your Zabbix server:

```
vim /etc/zabbix/zabbix_server.conf
```

6. Now, if we want to give our Zabbix server's `discoverer` process more room, we need to edit the correct parameter. Scroll down until you see the following:



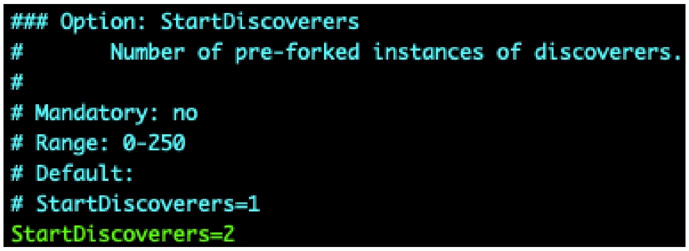
```
### Option: StartDiscoverers
#       Number of pre-forked instances of discoverers.
#
# Mandatory: no
# Range: 0-250
# Default:
# StartDiscoverers=1
```

Figure 10.12 – Zabbix server configuration file, StartDiscoverers default

7. Now add a new line under this and add the following line:

```
StartDiscoverers=2
```

8. If your file now looks like the following screenshot, you can save and exit the file:



```
### Option: StartDiscoverers
#       Number of pre-forked instances of discoverers.
#
# Mandatory: no
# Range: 0-250
# Default:
# StartDiscoverers=1
StartDiscoverers=2
```

Figure 10.13 – Zabbix server configuration file, StartDiscoverers 2

9. For the changes to take effect, we will need to restart the Zabbix server with the following command:

```
systemctl restart zabbix-server
```

10. Now if we go back to our Zabbix frontend, we should still be at our graph so we can see the following:

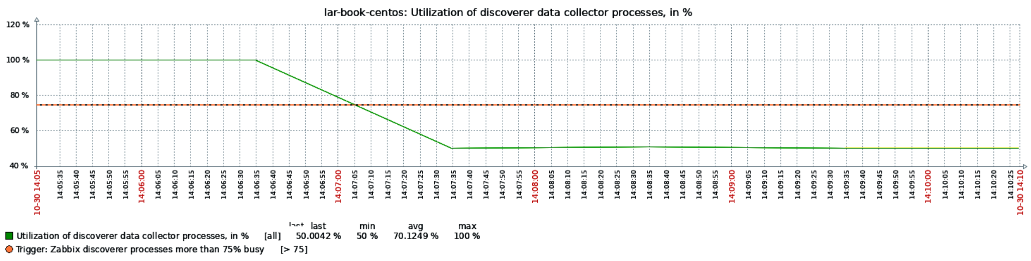


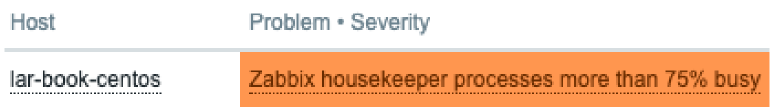
Figure 10.14 – Zabbix server discoverer graph

The utilization of our discoverer process has gone down, which means our problem won't show up anymore. That's how we edit Zabbix server processes.

Zabbix housekeeper

Another very common problem people face is the Zabbix housekeeper process being too busy. Let's log in to our Zabbix frontend and check out the problem:

1. When we navigate to **Monitoring** | **Dashboard**, and then select the default dashboard **Global view**, we might see something like this:



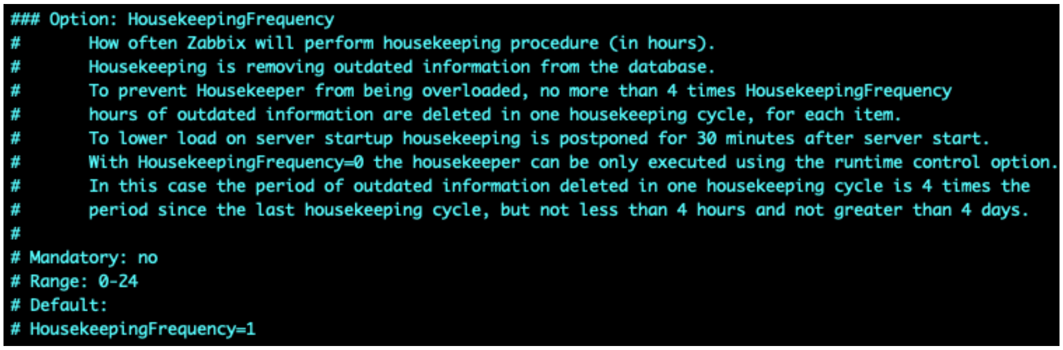
Host	Problem • Severity
lar-book-centos	Zabbix housekeeper processes more than 75% busy

Figure 10.15 – Zabbix problem for Zabbix housekeeper

2. Similar to editing any Zabbix process, we can also edit the Zabbix housekeeper process. Let's log in to the Linux CLI of our Zabbix server to edit our process.
3. Edit the following file on your Zabbix server:

```
vim /etc/zabbix/zabbix_server.conf
```

4. Now, if we want to edit this process, we need to edit the correct parameters. Scroll down until you see the following:



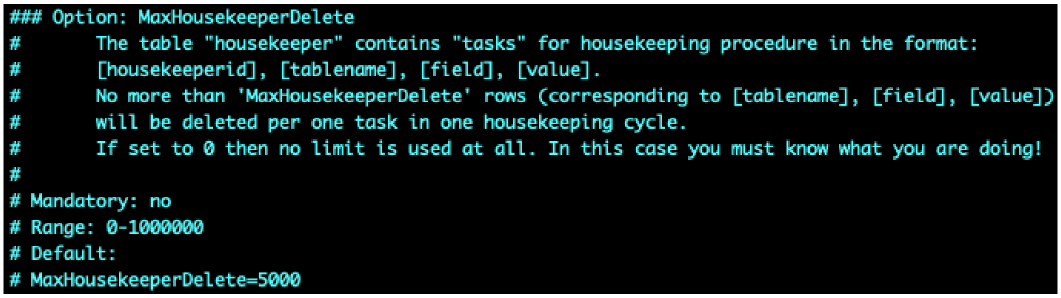
```
### Option: HousekeepingFrequency
#       How often Zabbix will perform housekeeping procedure (in hours).
#       Housekeeping is removing outdated information from the database.
#       To prevent Housekeeper from being overloaded, no more than 4 times HousekeepingFrequency
#       hours of outdated information are deleted in one housekeeping cycle, for each item.
#       To lower load on server startup housekeeping is postponed for 30 minutes after server start.
#       With HousekeepingFrequency=0 the housekeeper can be only executed using the runtime control option.
#       In this case the period of outdated information deleted in one housekeeping cycle is 4 times the
#       period since the last housekeeping cycle, but not less than 4 hours and not greater than 4 days.
#
# Mandatory: no
# Range: 0-24
# Default:
# HousekeepingFrequency=1
```

Figure 10.16 – Zabbix configuration file, HousekeepingFrequency 1

5. This is our first housekeeper parameter. Let's edit this parameter by adding the following line under this block:

```
HousekeepingFrequency=2
```

6. Now scroll down until you see the following:



```
### Option: MaxHousekeeperDelete
#       The table "housekeeper" contains "tasks" for housekeeping procedure in the format:
#       [housekeeperid], [tablename], [field], [value].
#       No more than 'MaxHousekeeperDelete' rows (corresponding to [tablename], [field], [value])
#       will be deleted per one task in one housekeeping cycle.
#       If set to 0 then no limit is used at all. In this case you must know what you are doing!
#
# Mandatory: no
# Range: 0-1000000
# Default:
# MaxHousekeeperDelete=5000
```

Figure 10.17 – Zabbix configuration file, HousekeepingDelete 5000

7. The preceding screenshot shows our second housekeeper parameter. Let's edit this parameter by adding the following line under this code block:

```
MaxHousekeeperDelete=20000
```

8. For the changes to take effect, we will need to restart the Zabbix server with the following command:

```
systemctl restart zabbix-server
```

Tuning a MySQL database

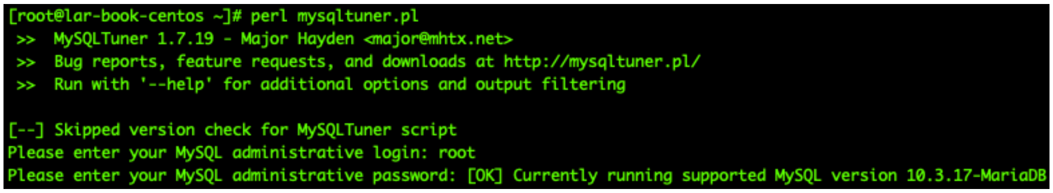
1. Let's see how we can tune a MySQL database with ease. First off, let's go to the following link in our browser: `https://github.com/major/MySQLTuner-perl`.
2. This link brings us to an open source GitHub project started by *Major Hayden*. Be sure to follow the repository and do all you can to help out. Let's download the script from the GitHub repository or simply use the following command:

```
wget https://raw.githubusercontent.com/major/
MySQLTuner-perl/master/mysqltuner.pl
```

3. Now we can execute this script with the following command:

```
perl mysqltuner.pl
```

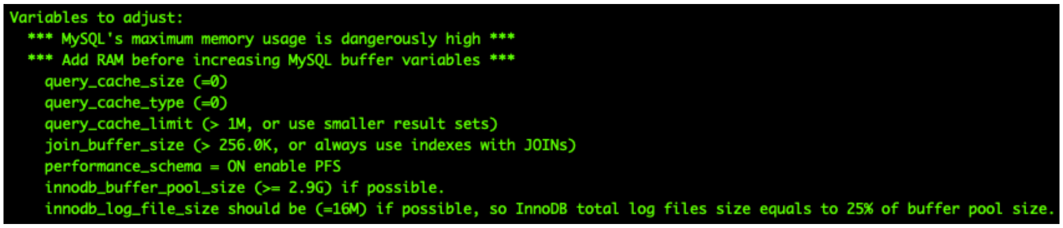
4. This will bring us to a prompt for our MySQL database credentials. Fill them out and continue:



```
[root@lar-book-centos ~]# perl mysqltuner.pl
 >>  MySQLTuner 1.7.19 - Major Hayden <major@mhtx.net>
 >>  Bug reports, feature requests, and downloads at http://mysqltuner.pl/
 >>  Run with '--help' for additional options and output filtering

[--] Skipped version check for MySQLTuner script
Please enter your MySQL administrative login: root
Please enter your MySQL administrative password: [OK] Currently running supported MySQL version 10.3.17-MariaDB
```

Figure 10.18 – MySQL tuner script execution

5. Now, the script will output a lot of information that you will need to read carefully, but the most important part is at the end – everything after `Variable to adjust`:



```
Variables to adjust:
  *** MySQL's maximum memory usage is dangerously high ***
  *** Add RAM before increasing MySQL buffer variables ***
    query_cache_size (=0)
    query_cache_type (=0)
    query_cache_limit (> 1M, or use smaller result sets)
    join_buffer_size (> 256.0K, or always use indexes with JOINs)
    performance_schema = ON enable PFS
    innodb_buffer_pool_size (>= 2.9G) if possible.
    innodb_log_file_size should be (=16M) if possible, so InnoDB total log files size equals to 25% of buffer pool size.
```

Figure 10.19 – MySQL tuner script output

6. We can edit these variables in the MySQL `my.cnf` file. In my case, I edit it with the following command:

```
vim etc/my.cnf
```

7. Now, you simply edit or add the variables that are suggested in the script and then restart your MySQL server:

```
systemctl restart mariadb
```

How it works

We've just done three of the main performance tweaks we can do for a Zabbix server, but there's a lot more to do. Let's take a look at what we've just edited, consider why we've edited it, and find out whether it's really that simple.

Zabbix processes

Zabbix processes are a big part of your Zabbix server setup and must be edited with care. In this recipe, we've only just edited the discoverer process on a small installation. This problem was easy as the server had more than enough resources to account for another process running.

Now if we look at the following diagram, we can see the situation as it was before we added a new discoverer process:

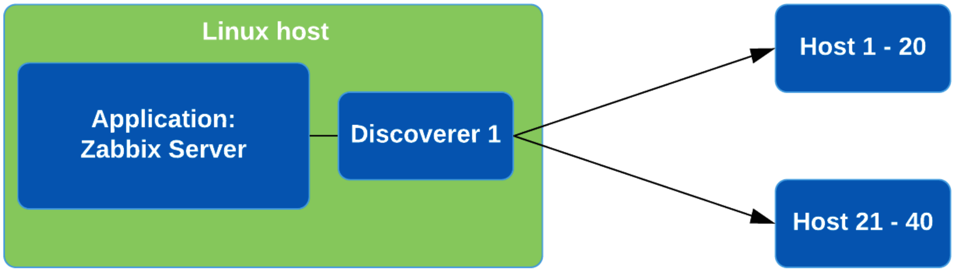


Figure 10.20 – Zabbix server single process setup diagram

We can see our `Linux host` running our `Zabbix server` application and we can see our `Discoverer 1` process discovering host 1 through 20. As we've seen, this is apparently too heavy for our system, so we added another discoverer:

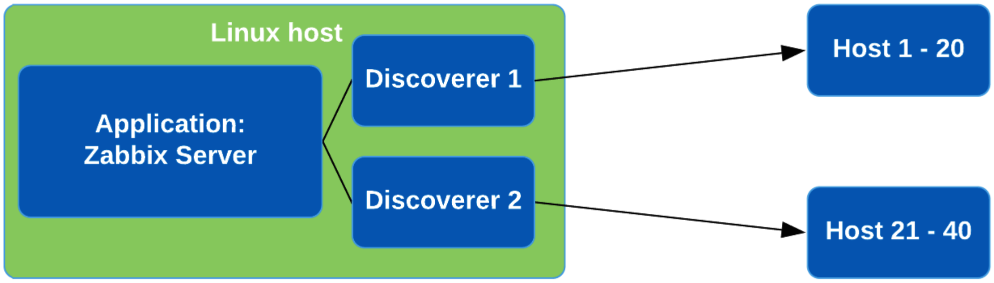


Figure 10.21 – Zabbix server multiple process setup diagram

Our new setup distributes the load of the process evenly by using several discoverers for discovering hosts. It works the same for the other processes – more processes mean better distributions of tasks.

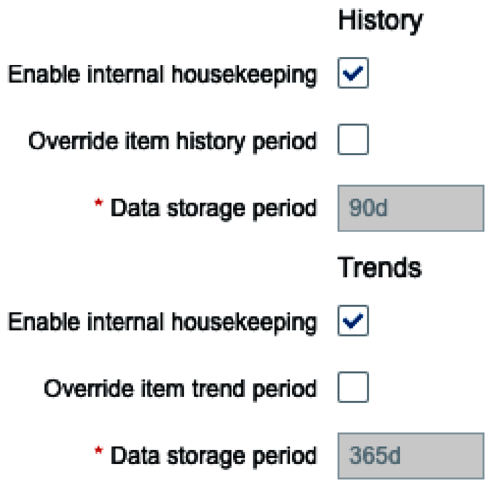
However, there are several things to be careful of here. First of all, not all issues can be solved by simply throwing more resources at them. Some Zabbix setups are configured poorly, where there's something in the configuration making our processes unnecessarily busy. If we deal with the poor configuration aspect, we can take away the high load, thus we need fewer processes.

The second thing I'd like to stress is that we can keep adding processes to our Zabbix server configuration – within limits. Before we reach those limits though, you are definitely going to reach the roof of what our Linux host hardware is capable of. Make sure you have enough RAM and CPU power to actually run all these processes or use Zabbix proxies for offloading.

Zabbix housekeeper

Now for the housekeeper, a very important process for Zabbix administrators that haven't set up MySQL partitioning or PostgreSQL TimescaleDB partitioning yet. The Zabbix housekeeper process is a process that connects to our database and then drops information line by line that has expired. You might think, how do you mean *expired*? Well, we can set limits in the Zabbix server for how long an item should be kept in the database.

If we look at **Administration** | **General** and then use the drop-down menu to go to **Housekeeping**, part of what we will see is shown in the following screenshot:


History

Enable internal housekeeping	✓
Override item history period	☐
* Data storage period	90d

Trends

Enable internal housekeeping	✓
Override item trend period	☐
* Data storage period	365d

Figure 10.22 – Zabbix server history and trends housekeeping setup

These are our global **History** and **Trends** housekeeping parameters. This defines how long an item's data should be kept in our database. If we look at an item on a template or host, we can also see these parameters:

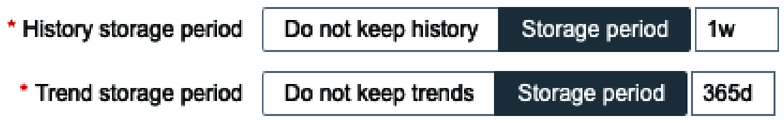

* History storage period	Do not keep history	Storage period	1w
* Trend storage period	Do not keep trends	Storage period	365d

Figure 10.23 – Zabbix item history and trends housekeeping parameters

These settings override the global settings so you can tweak the housekeeper further. That's how the housekeeper keeps your database in check.

But now, let's look at the tweaks we made in our Zabbix server configuration file, the first of which is `HousekeepingFrequency`. Housekeeping frequency is how often the housekeeper process is started. We've lowered this from every hour to every two hours. Now you might think that's worse, but it doesn't have to be. A lot of the time, we see that housekeeping is not done after one hour and then it just keeps going on and on.

We also changed the `MaxHousekeeperDelete` parameter, which is something completely different. This determines how many database rows our Zabbix housekeeper is allowed to delete in each run. The default settings determined that every hour, we can delete 5,000 database rows. With our new settings, we can now delete 20,000 database rows every two hours.

How does this change anything at all? Well, it might not. It completely depends on your setup. Tweaking the Zabbix housekeeper is different for every setup, and you will have to determine your optimum settings for yourself. Try to balance what you see in your graphs with the two settings we've discussed here to see how well you can optimize it.

However, at one point, your Zabbix setup might grow big enough and Zabbix housekeeping won't be able to keep up. This is when you'll need to look at MySQL partitioning or PostgreSQL TimescaleDB. There's no predefined point where the Zabbix housekeeper won't be able to keep up, so it might be smarter to just start with MySQL partitioning or PostgreSQL TimescaleDB right from the start. After all, any setup might grow larger than expected, right?

Tuning a MySQL database

Now for tuning your MySQL database with the `mysqltuner.pl` script. This script does a lot in the background, but we can summarize it as follows: it looks at what the current utilization of your MySQL database is, and then outputs what it thinks the correct tuning variables would be.

Do not take the script output as a given, as like with Zabbix housekeeping, there is no way to give you a definitive setup for your database. Databases are simply more complicated than just putting in some tweaks and being done with it.

The script will definitely help you tweak your MySQL database to an extent, especially for the smaller setups. But make sure to extend your knowledge by reading blogs, guides, and books about databases regularly.

There's more

We went over how to tune a MySQL database, but we didn't go over how to tune a PostgreSQL instance. There's a wide variety of options out there to do this, so for more on that I recommend checking out the PostgreSQL wiki at `https://wiki.postgresql.org/wiki/Performance_Optimization`.

There are different varieties and different preferences at play here. Make sure to check them all out well and pick the one that works the best for you.

11 Advanced Zabbix Database Management

Whether you've been using Zabbix for a while or you are looking toward setting up your first production instance, database management is important right from the start. A lot of the time, people set up their Zabbix database and don't know yet that it will be a big database. The Zabbix housekeeper just can't keep up when your database grows beyond a certain size and that's where we need to look toward different options.

In this chapter, we'll look into keeping our Zabbix database from using up 100% disk space when the Zabbix housekeeper is not keeping up. For MySQL users, we'll look into using database partitioning to keep our database in check. For PostgreSQL users, we'll look toward the brand-new TimescaleDB support. Last but not least, we'll also check out how to secure our connection between the Zabbix server and database.

We'll do all this in the following recipes:

- Setting up MySQL partitioning for your Zabbix database
- Using the new PostgreSQL TimescaleDB functionality
- Securing your Zabbix MySQL database

Without further ado, let's get started on these recipes and learn all about managing our database.

Technical requirements

We are going to need some new servers for these recipes. One Linux server needs to run Zabbix server 5 with MySQL (MariaDB) set up; we'll call this host `lar-book-mysql-mgmt`. We will also need a Linux server running Zabbix server 5 with PostgreSQL, which we'll call `lar-book-postgresql-mgmt`.

We'll also need two servers for creating a secure Zabbix database setup. One server will be running the MySQL (MariaDB) database; let's call this server `lar-book-secure-db`. Then, connecting externally to a Zabbix database, we'll have our Zabbix server, which we'll call `lar-book-secure-zbx`.

Setting up MySQL partitioning for your Zabbix database

When working with a MySQL database, the biggest issue we face is how MySQL stores its data by default. There is no real order to the data that we can use if we want to drop large chunks of data. MySQL partitioning solves this issue; let's see how we can configure it to use for our Zabbix database.

Getting ready

For this recipe, we are going to need a running Zabbix server with a MySQL database. I'll be using MariaDB in my example, but any MySQL flavor should be the same. The Linux host I'll be using is called `lar-book-mysql-mgmt`, which already meets the requirements.

How to do it...

1. First things first, let's log in to our Linux CLI to execute our commands.
2. It's a good idea to open a screen because partitioning can take several days for big databases. If a screen is not installed, install it first as you will need `epel-release` on CentOS 8.

   The RHEL-based command is as follows:

   ```
   dnf install epel-release
   dnf install screen
   ```

The Debian-based command is as follows:

```
apt-get install screen
```

3. Run a new screen by issuing the following command:

```
screen -S mysql-part
```

Important note

It's not required to run partitioning on a screen, but definitely smart. Partitioning takes a long time. Also, make sure to move your database to another machine with ample resources to partition or stop the Zabbix server process for several days.

4. Now, let's log in to the MySQL application as the root user with the following command:

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

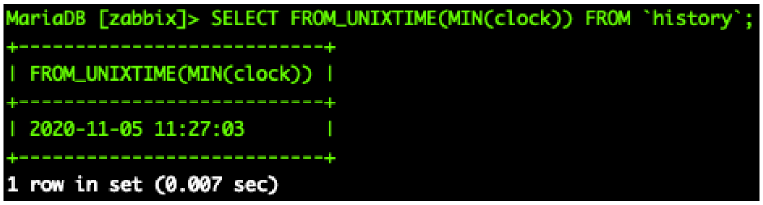
5. Now, move to use the Zabbix database with the following command:

```
USE zabbix;
```

6. We are going to need to partition some tables here, but to do this, we need to know the UNIX timestamp on our tables:

```
SELECT FROM_UNIXTIME(MIN(clock)) FROM `history`;
```

You will receive an output like this:



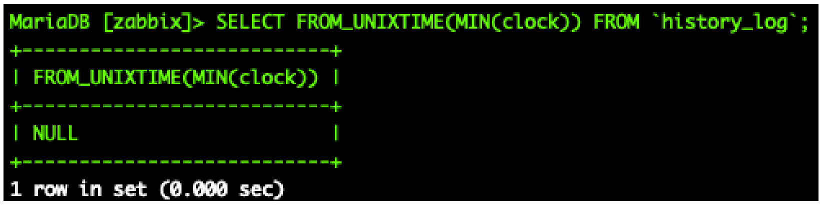
```
MariaDB [zabbix]> SELECT FROM_UNIXTIME(MIN(clock)) FROM `history`;
+---------------------------+
| FROM_UNIXTIME(MIN(clock)) |
+---------------------------+
| 2020-11-05 11:27:03       |
+---------------------------+
1 row in set (0.007 sec)
```

Figure 11.1 – MySQL returning a timestamp on the table history

7. This timestamp should be about the same for every single table we are going to partition. Verify this by running the same query for the remaining history tables:

```
SELECT FROM_UNIXTIME(MIN(clock)) FROM `history`;
SELECT FROM_UNIXTIME(MIN(clock)) FROM `history_uint`;
SELECT FROM_UNIXTIME(MIN(clock)) FROM `history_str`;
SELECT FROM_UNIXTIME(MIN(clock)) FROM `history_text`;
SELECT FROM_UNIXTIME(MIN(clock)) FROM `history_log`;
```

8. A table might return a different value or even no value at all. We need to take this into account when creating our partitions. A partition showing `NULL` has no data, but an earlier date means we need an earlier partition:



```
MariaDB [zabbix]> SELECT FROM_UNIXTIME(MIN(clock)) FROM `history_log`;
+---------------------------+
| FROM_UNIXTIME(MIN(clock)) |
+---------------------------+
| NULL                      |
+---------------------------+
1 row in set (0.000 sec)
```

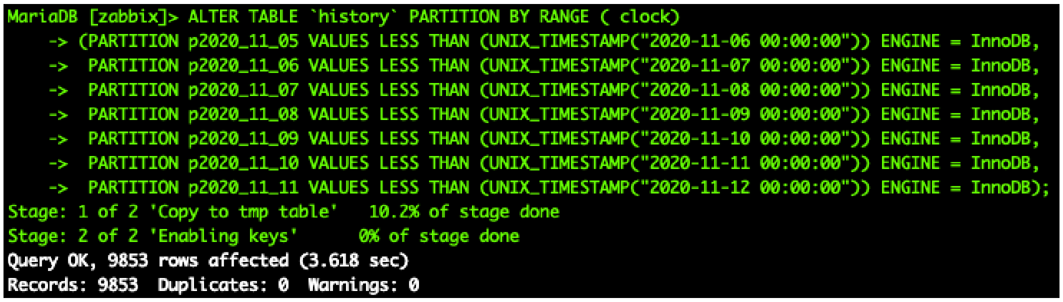
Figure 11.2 – MySQL returning a timestamp on the history_log table

9. Let's start with the `history` table. We are going to partition this table by day, and we are going to do this up until the date it is today; for me, it is `11-11-2020`. Let's prepare the following MySQL query (for example, in a notepad):

```
ALTER TABLE `history` PARTITION BY RANGE ( clock)
(PARTITION p2020_11_05 VALUES LESS THAN (UNIX_
TIMESTAMP("2020-11-06 00:00:00")) ENGINE = InnoDB,
 PARTITION p2020_11_06 VALUES LESS THAN (UNIX_
TIMESTAMP("2020-11-07 00:00:00")) ENGINE = InnoDB,
 PARTITION p2020_11_07 VALUES LESS THAN (UNIX_
TIMESTAMP("2020-11-08 00:00:00")) ENGINE = InnoDB,
 PARTITION p2020_11_08 VALUES LESS THAN (UNIX_
TIMESTAMP("2020-11-09 00:00:00")) ENGINE = InnoDB,
 PARTITION p2020_11_09 VALUES LESS THAN (UNIX_
TIMESTAMP("2020-11-10 00:00:00")) ENGINE = InnoDB,
 PARTITION p2020_11_10 VALUES LESS THAN (UNIX_
TIMESTAMP("2020-11-11 00:00:00")) ENGINE = InnoDB,
 PARTITION p2020_11_11 VALUES LESS THAN (UNIX_
TIMESTAMP("2020-11-12 00:00:00")) ENGINE = InnoDB);
```

10. Make sure that the oldest partition here matches the timestamp we collected in *Step 9*. In my case, the oldest data was from November 5, so this is my oldest partition. Also, make sure that your newest partition matches the date you are partitioning on.

11. Copy and paste the prepared MySQL query from *Step 9* and click *Enter*. This might take a while, as your table might be quite large. After you're done, you will see the following:



```
MariaDB [zabbix]> ALTER TABLE `history` PARTITION BY RANGE ( clock)
    -> (PARTITION p2020_11_05 VALUES LESS THAN (UNIX_TIMESTAMP("2020-11-06 00:00:00")) ENGINE = InnoDB,
    ->  PARTITION p2020_11_06 VALUES LESS THAN (UNIX_TIMESTAMP("2020-11-07 00:00:00")) ENGINE = InnoDB,
    ->  PARTITION p2020_11_07 VALUES LESS THAN (UNIX_TIMESTAMP("2020-11-08 00:00:00")) ENGINE = InnoDB,
    ->  PARTITION p2020_11_08 VALUES LESS THAN (UNIX_TIMESTAMP("2020-11-09 00:00:00")) ENGINE = InnoDB,
    ->  PARTITION p2020_11_09 VALUES LESS THAN (UNIX_TIMESTAMP("2020-11-10 00:00:00")) ENGINE = InnoDB,
    ->  PARTITION p2020_11_10 VALUES LESS THAN (UNIX_TIMESTAMP("2020-11-11 00:00:00")) ENGINE = InnoDB,
    ->  PARTITION p2020_11_11 VALUES LESS THAN (UNIX_TIMESTAMP("2020-11-12 00:00:00")) ENGINE = InnoDB);
Stage: 1 of 2 'Copy to tmp table'   10.2% of stage done
Stage: 2 of 2 'Enabling keys'      0% of stage done
Query OK, 9853 rows affected (3.618 sec)
Records: 9853  Duplicates: 0  Warnings: 0
```

Figure 11.3 – MySQL returning a successful query result for the history table

12. Do the same partitioning for the remaining history tables; make sure to use the other UNIX timestamps for the earliest partition:

    a. `history_uint`

    b. `history_str`

    c. `history_text`

    d. `history_log`

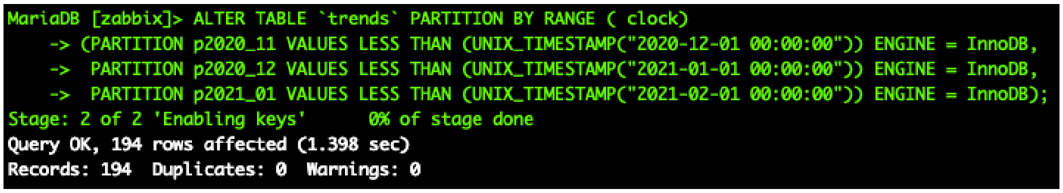
13. Once you've partitioned all the history tables, let's partition the `trends` tables. We have two of these called `trends` and `trends_uint`.
14. We are going to check the timestamps again with the following:

```
SELECT FROM_UNIXTIME(MIN(clock)) FROM `trends`;
SELECT FROM_UNIXTIME(MIN(clock)) FROM `trends_uint`;
```

15. For these tables, it's important to focus on what the earliest month is. For my tables, this is month 11 of year 2020.
16. Now, let's prepare and execute the partitioning for this table. Let's do three extra partitions starting from the earliest date seen in the timestamp at *Step 14*:

```
ALTER TABLE `trends` PARTITION BY RANGE ( clock)
(PARTITION p2020_11 VALUES LESS THAN (UNIX_
TIMESTAMP("2020-12-01 00:00:00")) ENGINE = InnoDB,
 PARTITION p2020_12 VALUES LESS THAN (UNIX_
TIMESTAMP("2021-01-01 00:00:00")) ENGINE = InnoDB,
 PARTITION p2021_01 VALUES LESS THAN (UNIX_
TIMESTAMP("2021-02-01 00:00:00")) ENGINE = InnoDB);
```

17. Again, we partition from the earliest collected UNIX timestamp, up until the current month. But there's no harm in creating some new partitions for future data:



```
MariaDB [zabbix]> ALTER TABLE `trends` PARTITION BY RANGE ( clock)
    -> (PARTITION p2020_11 VALUES LESS THAN (UNIX_TIMESTAMP("2020-12-01 00:00:00")) ENGINE = InnoDB,
    ->  PARTITION p2020_12 VALUES LESS THAN (UNIX_TIMESTAMP("2021-01-01 00:00:00")) ENGINE = InnoDB,
    ->  PARTITION p2021_01 VALUES LESS THAN (UNIX_TIMESTAMP("2021-02-01 00:00:00")) ENGINE = InnoDB);
Stage: 2 of 2 'Enabling keys'      0% of stage done
Query OK, 194 rows affected (1.398 sec)
Records: 194  Duplicates: 0  Warnings: 0
```

Figure 11.4 – MySQL returning a successful query result for the trends table

18. Do the same thing for the `trends_uint` table.
19. That concludes the actual partitioning of the database. Let's make sure our partitions remain managed. On your Linux host, download the partitioning script with the following command:

```
wget https://raw.githubusercontent.com/PacktPublishing/
Zabbix-5-Network-Monitoring-Cookbook/master/chapter11/
mysql_zbx_part.pl
```

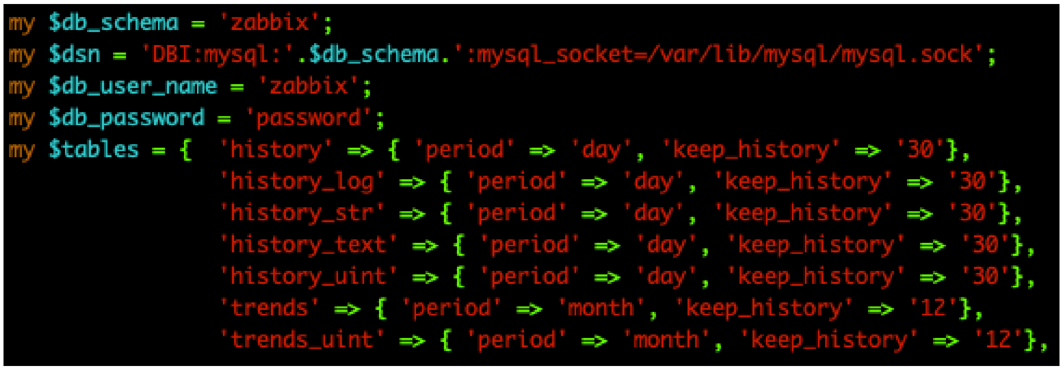
20. If you can't use `wget`, simply download the script from the following link: `https://github.com/PacktPublishing/Zabbix-5-Network-Monitoring-Cookbook/blob/master/chapter11/mysql_zbx_part.pl`.
21. Now, move the script to the `/usr/lib/zabbix/` folder with the following command:

```
mv mysql_zbx_part.pl /usr/lib/zabbix/
```

22. We are going to customize some details in the script. Edit the script with the following:

```
vim /usr/lib/zabbix/mysql_zbx_part.pl
```

We need to edit some text at the following part:



```
my $db_schema = 'zabbix';
my $dsn = 'DBI:mysql:'.$db_schema.':mysql_socket=/var/lib/mysql/mysql.sock';
my $db_user_name = 'zabbix';
my $db_password = 'password';
my $tables = {  'history' => { 'period' => 'day', 'keep_history' => '30'},
                'history_log' => { 'period' => 'day', 'keep_history' => '30'},
                'history_str' => { 'period' => 'day', 'keep_history' => '30'},
                'history_text' => { 'period' => 'day', 'keep_history' => '30'},
                'history_uint' => { 'period' => 'day', 'keep_history' => '30'},
                'trends' => { 'period' => 'month', 'keep_history' => '12'},
                'trends_uint' => { 'period' => 'month', 'keep_history' => '12'},
```

Figure 11.5 – MySQL Zabbix partitioning script user parameters

23. Edit `$db_schema` to match your Zabbix database name.
24. Edit `$db_user_name` to match your Zabbix database username.
25. Edit `$db_password` to match your Zabbix database password.
26. Now, at the `$tables` variable, we are going to add some of the most important details. This is where we'll add how much `days of history` data we want to keep and how much `months of trends` data. Add your values; the default settings keep 30 days of history data and 12 months of trends data.
27. Also, make sure to edit the `my $curr_tz = 'Europe/Amsterdam';` line to match your own time zone.

> **Tip**
> If you are using a version of Zabbix before 2.2 or a MySQL version before 5.6, there are some extra lines of configuration that need to be commented and uncommented in the script. If this applies to you, read the comments in the `mysql_zbx_part.pl` script file and edit it.

28. Before executing the script, we are going to need to install some Perl dependencies. On CentOS 8, we will need the `PowerTools` repo. Install the dependencies with the following.

The RHEL-based commands are as follows:

```
dnf config-manager --set-enabled PowerTools
dnf update
dnf install perl-Sys-Syslog
dnf install perl-DateTime
```

The Debian-based commands are as follows:

```
apt-get install liblogger-syslog-perl
apt-get install libdatetime-perl
```

29. Make the script executable with the following command:

```
chmod +x /usr/lib/zabbix/mysql_zbx_part.pl
```

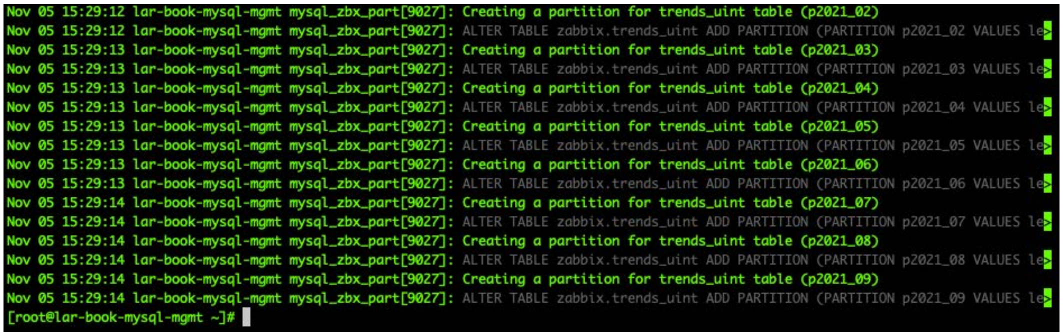
30. Then, this is the moment where we should be ready to execute the script to see whether it is working. Let's execute it:

```
/usr/lib/zabbix/mysql_zbx_part.pl
```

31. Once your script has finished running, let's see whether it was successful with the following command:

```
journalctl -t mysql_zbx_part
```

32. You should see an output like this:



```
Nov 05 15:29:12 lar-book-mysql-mgmt mysql_zbx_part[9027]: Creating a partition for trends_uint table (p2021_02)
Nov 05 15:29:12 lar-book-mysql-mgmt mysql_zbx_part[9027]: ALTER TABLE zabbix.trends_uint ADD PARTITION (PARTITION p2021_02 VALUES le>
Nov 05 15:29:13 lar-book-mysql-mgmt mysql_zbx_part[9027]: Creating a partition for trends_uint table (p2021_03)
Nov 05 15:29:13 lar-book-mysql-mgmt mysql_zbx_part[9027]: ALTER TABLE zabbix.trends_uint ADD PARTITION (PARTITION p2021_03 VALUES le>
Nov 05 15:29:13 lar-book-mysql-mgmt mysql_zbx_part[9027]: Creating a partition for trends_uint table (p2021_04)
Nov 05 15:29:13 lar-book-mysql-mgmt mysql_zbx_part[9027]: ALTER TABLE zabbix.trends_uint ADD PARTITION (PARTITION p2021_04 VALUES le>
Nov 05 15:29:13 lar-book-mysql-mgmt mysql_zbx_part[9027]: Creating a partition for trends_uint table (p2021_05)
Nov 05 15:29:13 lar-book-mysql-mgmt mysql_zbx_part[9027]: ALTER TABLE zabbix.trends_uint ADD PARTITION (PARTITION p2021_05 VALUES le>
Nov 05 15:29:13 lar-book-mysql-mgmt mysql_zbx_part[9027]: Creating a partition for trends_uint table (p2021_06)
Nov 05 15:29:13 lar-book-mysql-mgmt mysql_zbx_part[9027]: ALTER TABLE zabbix.trends_uint ADD PARTITION (PARTITION p2021_06 VALUES le>
Nov 05 15:29:14 lar-book-mysql-mgmt mysql_zbx_part[9027]: Creating a partition for trends_uint table (p2021_07)
Nov 05 15:29:14 lar-book-mysql-mgmt mysql_zbx_part[9027]: ALTER TABLE zabbix.trends_uint ADD PARTITION (PARTITION p2021_07 VALUES le>
Nov 05 15:29:14 lar-book-mysql-mgmt mysql_zbx_part[9027]: Creating a partition for trends_uint table (p2021_08)
Nov 05 15:29:14 lar-book-mysql-mgmt mysql_zbx_part[9027]: ALTER TABLE zabbix.trends_uint ADD PARTITION (PARTITION p2021_08 VALUES le>
Nov 05 15:29:14 lar-book-mysql-mgmt mysql_zbx_part[9027]: Creating a partition for trends_uint table (p2021_09)
Nov 05 15:29:14 lar-book-mysql-mgmt mysql_zbx_part[9027]: ALTER TABLE zabbix.trends_uint ADD PARTITION (PARTITION p2021_09 VALUES le>
[root@lar-book-mysql-mgmt ~]#
```

Figure 11.6 – MySQL Zabbix partitioning script results

33. Now, execute the following command:

```
crontab -e
```

34. To automate the execution of the script, add the following line to the file:

```
* 0 * * * /usr/lib/zabbix/mysql_zbx_part.pl
```

35. The last thing we are going to need to do is to go to the Zabbix frontend. Navigate to **Administration** | **General**.
36. Now, use the drop-down menu to go to **Housekeeping**:

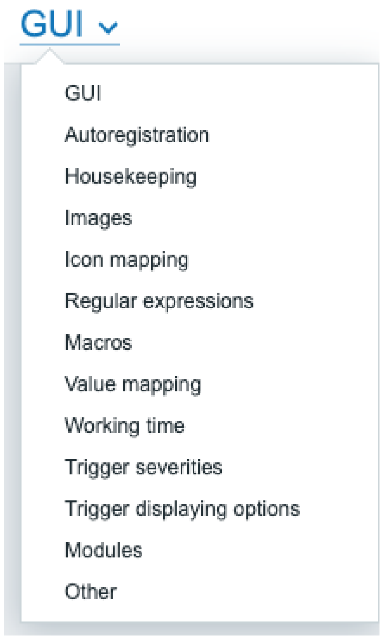
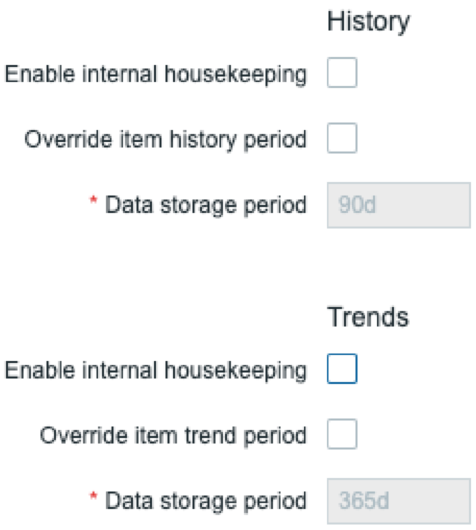


Figure 11.7 – Zabbix Administration | General drop-down menu, Housekeeping option

37. Disable the housekeeping for the **History** and **Trends** tables. It will look like the following:



History

Enable internal housekeeping	☐
Override item history period	☐
* Data storage period	90d

Trends

Enable internal housekeeping	☐
Override item trend period	☐
* Data storage period	365d

Figure 11.8 – Zabbix Administration | General | Housekeeping disabled for History and Trends

That concludes our Zabbix database partitioning setup.

How it works...

Database partitioning seems like a daring task at first, but once you break it down into chunks, it is not that hard to do. It is simply the process of breaking down our most important Zabbix database tables into time-based partitions. Once these partitions are set up, we simply need to manage these tables with a script and we're ready.

Look at the following figure and let's say today is **07-11-2020**. We have a lot of partitions managed by the script. All of our **history** data today is going to be written to the partition for this day and all of our **trends** data is going to be written into the partition for this month:

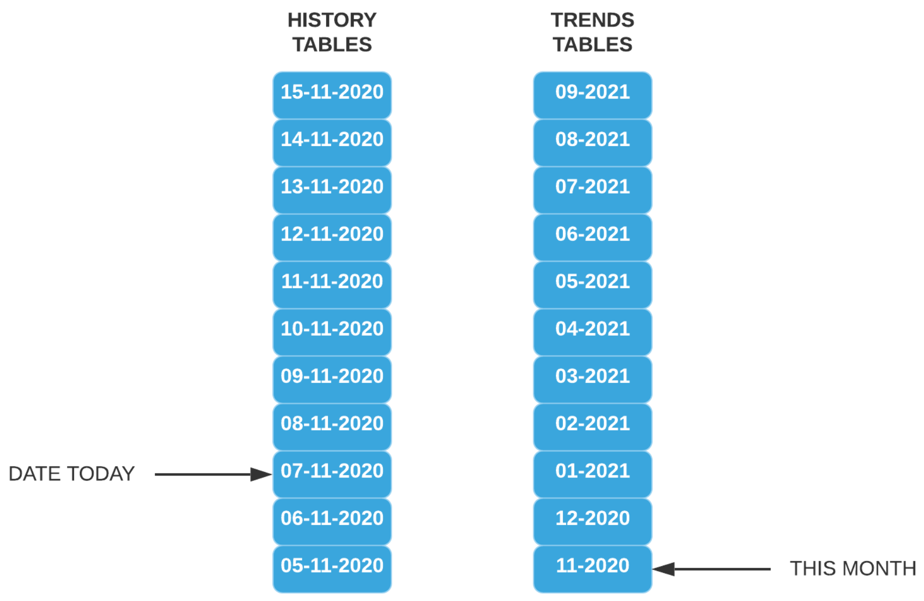


Figure 11.9 – Zabbix partitioning illustration

The actual script does only two things. It creates new partitions and it deletes old partitions.

For deleting partitions, once a partition reaches an age older than specified in the `$tables` variable, it drops the entire partition.

For creating partitions, every time the script is run, it creates 10 partitions into the future starting from today. Except of course when a partition already exists.

This is better than using the housekeeper for one clear reason. It's simply faster! The Zabbix housekeeper goes through our database data line by line to check the UNIX timestamp and then it drops that line when it reaches data older than specified. This takes time and resources. Dropping a partition, though, is almost instant.

One downside of partitioning a Zabbix database, though, is that we can no longer use the frontend item history and trend configuration. This means we can't specify different history and trends for different items; it's all global now.

See also

When I first started using Zabbix, I did not have a book like this one. Instead, I relied heavily on the resources available online and my own skillset. A great partitioning guide is found at the following link, which contains loads of information about partitioning. Once again, sharing is caring!

`https://zabbix.org/wiki/Docs/howto/mysql_partitioning`

Using the new PostgreSQL TimescaleDB functionality

TimescaleDB is an open source relational PostgreSQL database for time-based series data. Using PostgreSQL TimescaleDB is a solid way to work around using the Zabbix housekeeper to manage your PostgreSQL database. In this recipe, we will go over the installation of PostgreSQL TimescaleDB on a new server and how to set it up with Zabbix.

Getting ready

We will need an empty Linux server. I'll be using my server called `lar-book-postgresql-mgmt`.

How to do it...

Debian-based installation

1. Let's log in to our Linux CLI and add the PostgreSQL repo with the following commands:

```
echo "deb http://apt.postgresql.org/pub/repos/apt/ $(lsb_
release -c -s)-pgdg main" | sudo tee /etc/apt/sources.
list.d/pgdg.list
```

```
wget --quiet -O - https://www.postgresql.org/media/keys/
ACCC4CF8.asc | sudo apt-key add -
```

```
sudo apt-get update
```

2. Now, add the TimescaleDB repository:

```
add-apt-repository ppa:timescale/timescaledb-ppa
```

```
apt-get update
```

3. Now, install TimescaleDB with the installation command:

```
apt install timescaledb-postgresql-12
```

4. Start and enable PostgreSQL 12:

```
systemctl enable postgresql-12
systemctl start postgresql-12
```

5. Now, continue with the *TimescaleDB configuration* section of this recipe.

RHEL-based installation

1. Let's start by logging in to our Linux CLI. We will need PostgreSQL version 11 or higher. Let's install version 12; first, disable AppStream:

```
dnf -qy module disable postgresql
```

2. Add the correct repository:

```
dnf install https://download.postgresql.org/pub/repos/
yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.
rpm
```

3. Then, install PostgreSQL:

```
dnf install postgresql12 postgresql12-server
```

4. Make sure to initialize the database:

```
/usr/pgsql-12/bin/postgresql-12-setup initdb
```

5. Now, edit the following file:

```
vim /etc/yum.repos.d/timescale_timescaledb.repo
```

6. Add the repo information to the file and save it:

```
[timescale_timescaledb]
name=timescale_timescaledb
baseurl=https://packagecloud.io/timescale/timescaledb/
el/7/$basearch
repo_gpgcheck=1
gpgcheck=0
enabled=1
```

```
gpgkey=https://packagecloud.io/timescale/timescaledb/
gpgkey
sslverify=1
sslcacert=/etc/pki/tls/certs/ca-bundle.crt
metadata_expire=300
```

7. Install TimescaleDB with the installation command:

```
dnf install timescaledb-postgresql-12
```

8. Now, continue with the *TimescaleDB configuration* section of this recipe.

TimescaleDB configuration

In this section, we'll go over how to set up TimescaleDB after finishing the installation process. There's a lot more to configure, so let's check it out:

1. Let's start by running the following command:

```
timescaledb-tune
```

2. Sometimes this does not work, and you want to specify the PostgreSQL location like this:

```
timescaledb-tune --pg-config=/usr/pgsql-12/bin/pg_config
```

3. Go through the steps and answer the questions with `yes` or `no` accordingly. For a first-time setup, `yes` on everything is good.
4. Now, restart PostgreSQL:

```
systemctl restart postgresql-12.service
```

5. If you haven't already, download and install Zabbix with the following.

   The RHEL-based commands are as follows:

```
rpm -Uvh https://repo.zabbix.com/zabbix/5.0/rhel/8/
x86_64/zabbix-release-5.0-1.el8.noarch.rpm
dnf clean all
dnf install zabbix-server-pgsql zabbix-web-pgsql zabbix-
apache-conf zabbix-agent
```

The Debian-based commands are as follows:

```
wget https://repo.zabbix.com/zabbix/5.0/ubuntu/pool/
main/z/zabbix-release/zabbix-release_5.0-1+focal_all.deb
```

```
dpkg -i zabbix-release_5.0-1+focal_all.deb
```

```
apt update
```

```
apt install zabbix-server-pgsql zabbix-frontend-php
php7.4-pgsql zabbix-apache-conf zabbix-agent
```

6. Create the initial database with the following:

```
sudo -u postgres createuser --pwprompt zabbix
```

```
sudo -u postgres createdb -O zabbix zabbix
```

7. Import the database schema for PostgreSQL:

```
zcat /usr/share/doc/zabbix-server-pgsql*/create.sql.gz |
sudo -u zabbix psql zabbix
```

8. Add the database password to the Zabbix configuration file by editing it:

```
vim /etc/zabbix/zabbix_server.conf
```

9. Add the following lines, where `password` is your password and `DBHost` is empty:

```
DBHost=
```

```
DBPassword=password
```

10. Do not forget to edit your timezone with the following:

```
vim /etc/php-fpm.d/zabbix.conf
```

11. Then, edit the line and remove `;`:

```
; php_value[date.timezone] = Europe/Riga
```

12. Now, enable the TimescaleDB extension with the following command:

```
echo "CREATE EXTENSION IF NOT EXISTS timescaledb
CASCADE;" | sudo -u postgres psql zabbix
```

13. Unpack the `timescale.sql` script located in your Zabbix share folder:

```
gunzip /usr/share/doc/zabbix-server-pgsql/timescaledb.
sql.gz
```

14. Now, let's run `timescale.sql`:

```
cat /usr/share/doc/zabbix-server-pgsql/timescaledb.sql|
sudo -u zabbix psql zabbix
```

15. One more thing before moving to the frontend. We need to edit the `pg_hba.conf` file to allow our Zabbix frontend to connect. Edit the following file:

```
vim /var/lib/pgsql/12/data/pg_hba.conf
```

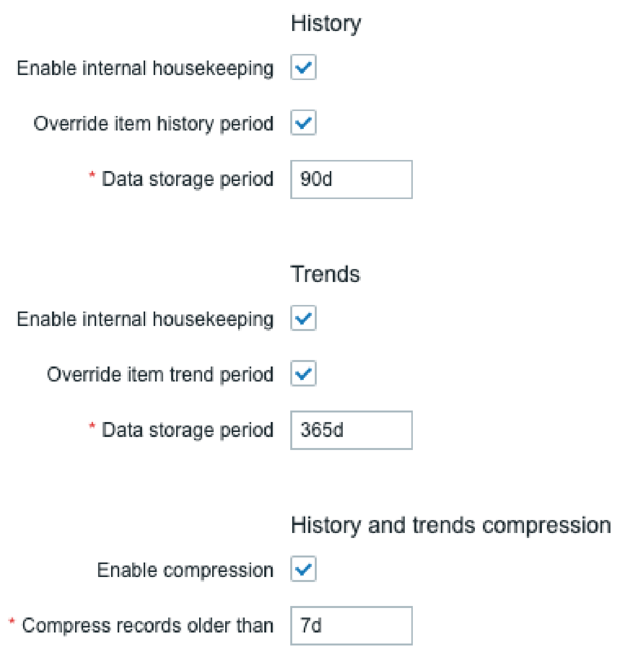
16. Make sure the following lines match in your file; they need to end with `md5`:

```
# "local" is for Unix domain socket connections only
local   all             all
md5
# IPv4 local connections:
host    all             all             127.0.0.1/32
md5
# IPv6 local connections:
host    all             all             ::1/128
md5
```

17. Now, start Zabbix and finish the frontend setup using the following commands:

```
systemctl restart zabbix-server zabbix-agent httpd
php-fpm
systemctl enable zabbix-server zabbix-agent httpd php-fpm
```

18. Once we navigate to the frontend and we've logged in to our setup, navigate to **Administration | General**. Use the drop-down menu to select **Housekeeping**.
19. We can now edit the following parameters to match our preferences and TimescaleDB will take care of maintaining the data retention period:



	History
Enable internal housekeeping	✓
Override item history period	✓
* Data storage period	90d
	Trends
Enable internal housekeeping	✓
Override item trend period	✓
* Data storage period	365d
	History and trends compression
Enable compression	✓
* Compress records older than	7d

Figure 11.10 – Zabbix Administration | General | Housekeeping, TimescaleDB specific options

How it works...

Using the TimescaleDB functionality with your Zabbix setup is a solid integration to your PostgreSQL database. The extension is supported by Zabbix and you can expect it to only get better in the near future.

Now, how TimescaleDB works is by dividing up your PostgreSQL hypertable into time-based chunks. If we look at the following figure, we can see how that looks:

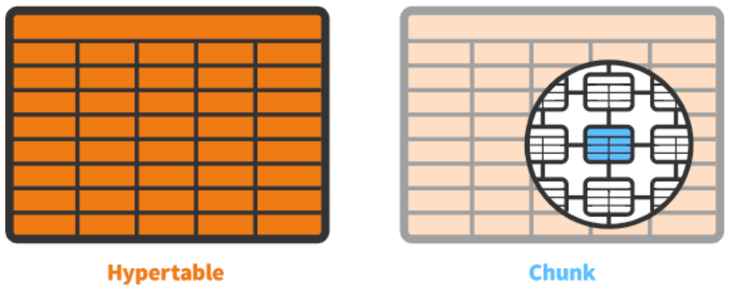


Figure 11.11 – TimescaleDB hypertable chunks diagram

These time-based chunks are a lot faster to drop from the database than using the Zabbix housekeeper. The Zabbix housekeeper goes through our database data line by line to check the UNIX timestamp and then it drops the line when it reaches data older than specified. This takes time and resources. Dropping a chunk though is almost instant.

Another great thing about using TimescaleDB with a Zabbix database is that we can still use the frontend item history and trend configuration. On top of that, TimescaleDB also compresses our data, to keep databases smaller.

The downside is that we can't specify different history and trends for different items; it's all global now.

See also

This recipe details the installation of PostgreSQL TimescaleDB. As this process is constantly changing, you might need to include some new information from the official TimescaleDB documentation. Check out their documentation here:

`https://docs.timescale.com/latest/getting-started/installation/rhel-centos/installation-yum`

Securing your Zabbix MySQL database

Another great added feature for the Zabbix server is the ability to encrypt data between the database and Zabbix components. This is particularly useful when you are running a split database and the Zabbix server over the network. A **Man in the Middle** (**MITM**) or other attacks can be executed on the network to gain access to your monitoring data.

In this recipe, we'll set up MySQL encryption between Zabbix components and the database to add another layer of security.

Getting ready

We are going to need a Zabbix setup that uses an external database. I'll be using the Linux `lar-book-secure-db` and `lar-book-secure-zbx` hosts.

The new server called `lar-book-secure-zbx` will be used to connect externally to the `lar-book-secure-db` database server. The database servers won't run our Zabbix server; this process will run on `lar-book-secure-zbx`.

Make sure that MariaDB is already installed on the `lar-book-secure-db` host and that you are running a recent supported version that is able to use encryption. If you don't know how to upgrade your database, check out, in *Chapter 10*, *Maintaining Your Zabbix Setup*, the recipe named *Upgrading Zabbix database from older MariaDB versions to MariaDB 10.5*, or check the documentation online.

How to do it...

1. Make sure your host files on both hosts from the *Getting ready* section contain the hostname and IP for your Linux hosts and edit the file with the following:

```
vim /etc/hosts
```

2. Then, fill in the file with your hostnames and IPs. It will look like this:

```
10.16.16.170 lar-book-secure-db
10.16.16.171 lar-book-secure-zbx
```

3. On the `lar-book-secure-db` MySQL server, if you haven't already, create the Zabbix database by logging in to MySQL:

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

4. Then, issue the following command to create the database:

```
create database zabbix character set utf8 collate utf8_
bin;
```

5. Also, make sure to create a user that will be able to access the database securely. Make sure the IP matches the IP from the Zabbix server:

```
create user 'zabbix'@'*' identified BY 'password';
grant all privileges on zabbix.* to 'zabbix'@*';
flush privileges;
```

6. Quit MySQL and then make sure to run the secure `mysql` script with the following:

```
mysql_secure_installation
```

7. Log in to `lar-book-secure-zbx` and install the Zabbix server repo with the following command:

```
rpm -Uvh https://repo.zabbix.com/zabbix/5.0/rhel/8/
x86_64/zabbix-release-5.0-1.el8.noarch.rpm
dnf clean all
```

8. Then, install the Zabbix server and its required components.

   Use the following RHEL-based command:

```
dnf install zabbix-server-mysql zabbix-web-mysql zabbix-
apache-conf zabbix-agent2 mariadb
```

   Use the following Debian-based command:

```
apt install zabbix-server-mysql zabbix-frontend-php
zabbix-apache-conf zabbix-agent mariadb
```

9. From the Zabbix server, connect to the remote database server and import the database scheme with the following command:

```
zcat /usr/share/doc/zabbix-server-mysql*/create.sql.gz |
mysql -h 10.16.16.170 -uzabbix -p zabbix
```

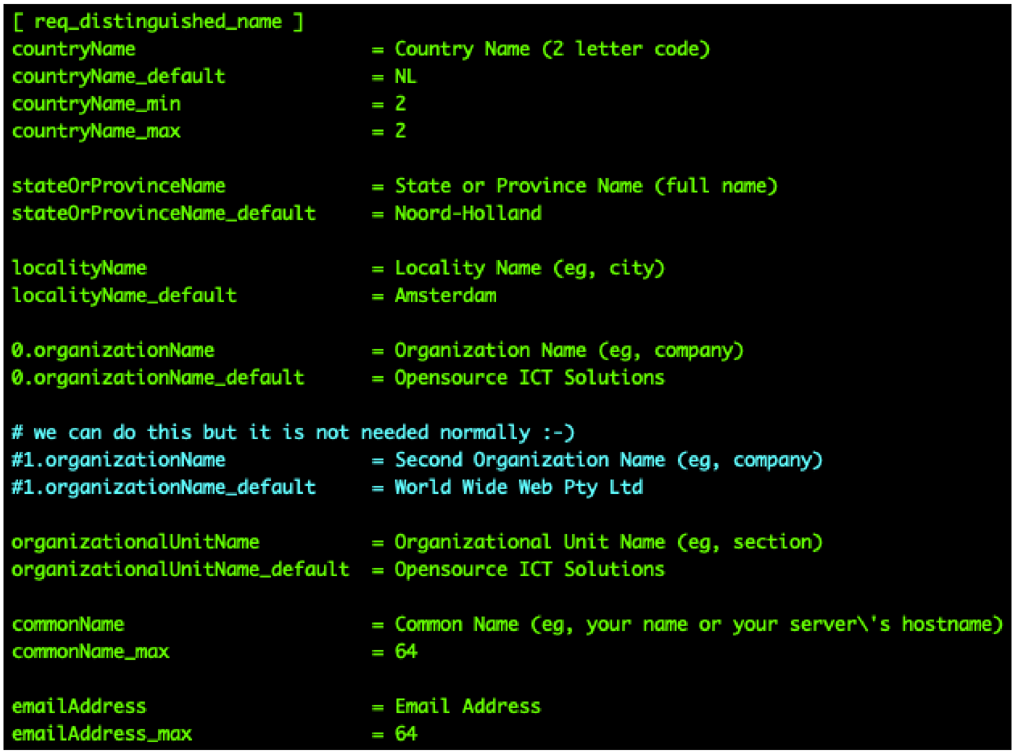
10. Now we are going to open the file called `openssl.cnf` and edit it by issuing the following command:

```
vim /etc/pki/tls/openssl.cnf
```

11. In this file, we need to edit the following lines:

```
countryName_default             = XX
stateOrProvinceName_default     = Default Province
localityName_default            = Default City
0.organizationName_default      = Default Company Ltd
organizationalUnitName_default. =
```

It will look like this filled out completely:



```
[ req_distinguished_name ]
countryName                     = Country Name (2 letter code)
countryName_default             = NL
countryName_min                 = 2
countryName_max                 = 2

stateOrProvinceName             = State or Province Name (full name)
stateOrProvinceName_default     = Noord-Holland

localityName                    = Locality Name (eg, city)
localityName_default            = Amsterdam

0.organizationName              = Organization Name (eg, company)
0.organizationName_default      = Opensource ICT Solutions

# we can do this but it is not needed normally :-)
#1.organizationName             = Second Organization Name (eg, company)
#1.organizationName_default     = World Wide Web Pty Ltd

organizationalUnitName          = Organizational Unit Name (eg, section)
organizationalUnitName_default  = Opensource ICT Solutions

commonName                      = Common Name (eg, your name or your server\'s hostname)
commonName_max                  = 64

emailAddress                    = Email Address
emailAddress_max                = 64
```

Figure 11.12 – OpenSSL config file with our personal defaults

12. We can also see this line:

```
dir      = /etc/pki/CA       # Where everything is kept
```

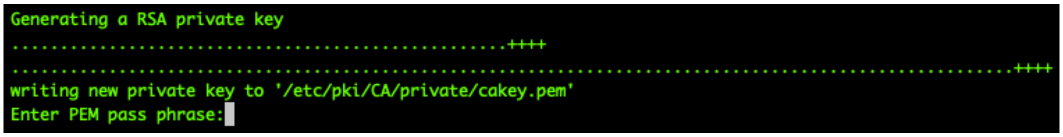
13. This means the default directory is `/etc/pki/CA`; if yours is different, act accordingly. Close the file by saving and continue.
14. Let's create a new folder for our private certificates using the following command:

```
mkdir -p /etc/pki/CA/private
```

15. Now, let's create our key pair in the new folder. Issue the following command:

```
openssl req -new -x509 -keyout /etc/pki/CA/private/
cakey.pem -out /etc/pki/CA/cacert.pem -days 3650 -newkey
rsa:4096
```

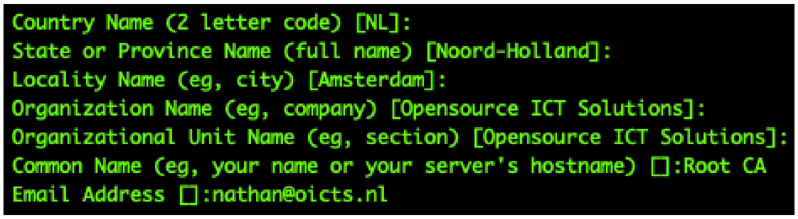
16. You will be prompted for a password now:



```
Generating a RSA private key
.............................................++++
.....................................................................................................++++
writing new private key to '/etc/pki/CA/private/cakey.pem'
Enter PEM pass phrase:
```

Figure 11.13 – Certificate generation response asking for a password

17. You might also be promoted to enter some information about your company. It will use the default we filled in earlier, so you can just press *Enter* up until `Common Name`.
18. Fill in `Root CA` for `Common Name` and add your email address like this:



```
Country Name (2 letter code) [NL]:
State or Province Name (full name) [Noord-Holland]:
Locality Name (eg, city) [Amsterdam]:
Organization Name (eg, company) [Opensource ICT Solutions]:
Organizational Unit Name (eg, section) [Opensource ICT Solutions]:
Common Name (eg, your name or your server's hostname) []:Root CA
Email Address []:nathan@oicts.nl
```

Figure 11.14 – Certificate generation response asking for information, Root CA

19. Next up is creating the actual signed certificates that our Zabbix server will use. Let's make sure that OpenSSL has the right files to keep track of signed certificates:

```
touch /etc/pki/CA/index.txt
echo 01 > /etc/pki/CA/serial
```

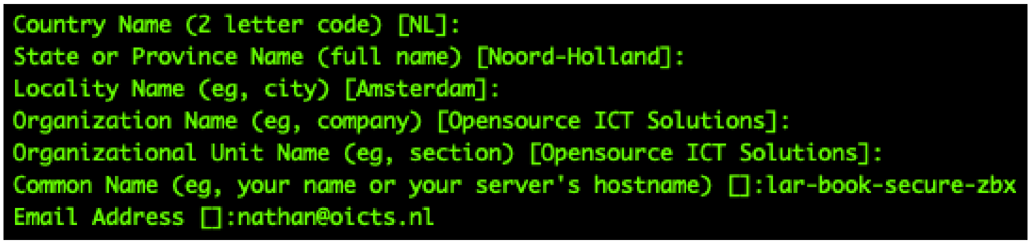
20. Then, create the folders to keep our certificates in:

```
mkdir /etc/pki/CA/unsigned
mkdir /etc/pki/CA/newcerts
mkdir /etc/pki/CA/certs
```

21. Now, let's create our certificate signing request for the `lar-book-secure-zbx` Zabbix server with the following command:

```
openssl req -nodes -new -keyout /etc/pki/CA/private/
zbx-srv_key.pem -out /etc/pki/CA/unsigned/zbx-srv_req.pem
-newkey rsa:2048
```

22. You will be prompted to add a password and your company information again. Use the default up until `Common Name`. We will fill out our `Common Name`, which will be the server hostname, and we'll add our email address like this:



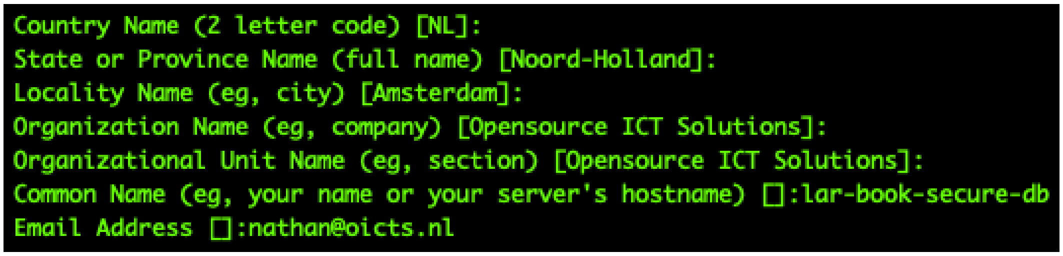
```
Country Name (2 letter code) [NL]:
State or Province Name (full name) [Noord-Holland]:
Locality Name (eg, city) [Amsterdam]:
Organization Name (eg, company) [Opensource ICT Solutions]:
Organizational Unit Name (eg, section) [Opensource ICT Solutions]:
Common Name (eg, your name or your server's hostname) []:lar-book-secure-zbx
Email Address []:nathan@oicts.nl
```

Figure 11.15 – Certificate generation response asking for information, lar-book-secure-zbx

23. Let's do the same for our `lar-book-secure-db` server:

```
openssl req -nodes -new -keyout /etc/pki/CA/private/
mysql-srv_key.pem -out /etc/pki/CA/unsigned/mysql-srv_
req.pem -newkey rsa:2048
```

The response will look like this:



```
Country Name (2 letter code) [NL]:
State or Province Name (full name) [Noord-Holland]:
Locality Name (eg, city) [Amsterdam]:
Organization Name (eg, company) [Opensource ICT Solutions]:
Organizational Unit Name (eg, section) [Opensource ICT Solutions]:
Common Name (eg, your name or your server's hostname) []:lar-book-secure-db
Email Address []:nathan@oicts.nl
```

Figure 11.16 – Certificate generation response asking for information, lar-book-secure-db

Important note

Our certificates need to be created without a password; otherwise, our MariaDB and Zabbix applications won't be able to use them. Make sure to specify the `-nodes` option.

24. Now, sign the certificate for `lar-book-secure-zbx` with the following command:

```
openssl ca -policy policy_anything -days 365 -out /
etc/pki/CA/certs/zbx-srv_crt.pem -infiles /etc/pki/CA/
unsigned/zbx-srv_req.pem
```

25. You will be promoted with the question `Sign the certificate? [y/n]`. Answer this and all the following questions with `Y`.
26. Now, let's do the same thing for the `lar-book-secure-db` certificate:

```
openssl ca -policy policy_anything -days 365 -out /etc/
pki/CA/certs/mysql-srv_crt.pem -infiles /etc/pki/CA/
unsigned/mysql-srv_req.pem
```

27. Let's log in to the `lar-book-secure-db` MySQL server and create a directory for our newly created certificates:

```
mkdir /etc/my.cnf.d/certificates/
```

28. Add the right permissions to the folder:

```
chown -R mysql. /etc/my.cnf.d/certificates/
```

29. Now, back at the new `lar-book-secure-zbx` Zabbix server, copy over the files to the database server with the following commands:

```
scp /etc/pki/CA/private/mysql-srv_key.pem
root@10.16.16.170:/etc/my.cnf.d/certificates/mysql-srv.
key
```

```
scp /etc/pki/CA/certs/mysql-srv_crt.pem
root@10.16.16.170:/etc/my.cnf.d/certificates/mysql-srv.
crt
```

```
scp /etc/pki/CA/cacert.pem root@10.16.16.170:/etc/
my.cnf.d/certificates/cacert.crt
```

30. Now, back at the `lar-book-secure-db` MySQL server, add the right permissions to the files:

```
chown -R mysql:mysql /etc/my.cnf.d/certificates/
```

```
chmod 400 /etc/my.cnf.d/certificates/mysql-srv.key
```

```
chmod 444 /etc/my.cnf.d/certificates/mysql-srv.crt
```

```
chmod 444 /etc/my.cnf.d/certificates/cacert.pem
```

31. Edit the MariaDB configuration file with the following command:

```
vim /etc/my.cnf.d/mariadb-server.cnf
```

32. Add the following lines to the configuration file under the `[mysqld]` block:

```
bind-address=lar-book-secure-db
ssl-ca=/etc/my.cnf.d/certificates/cacert.crt
ssl-cert=/etc/my.cnf.d/certificates/mysql-srv.crt
ssl-key=/etc/my.cnf.d/certificates/mysql-srv.key
```

33. Log in to MySQL with the following command:

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

34. Make sure our Zabbix MySQL user requires SSL encryption with the following:

```
alter user 'zabbix'@'10.16.16.171' require ssl;
flush privileges;
```

35. Quit and then restart MariaDB with the following command:

```
systemctl restart mariadb
```

36. Now, back on the `lar-book-secure-zbx` Zabbix server, create a new folder for our certificates:

```
mkdir -p /var/lib/zabbix/ssl/
```

37. Copy the certificates over to this folder with the following:

```
cp /etc/pki/CA/cacert.pem /var/lib/zabbix/ssl/
cp /etc/pki/CA/certs/zbx-srv_crt.pem /var/lib/zabbix/ssl/
zbx-srv.crt
cp /etc/pki/CA/private/zbx-srv_key.pem /var/lib/zabbix/
ssl/zbx-srv.key
```

38. Edit the Zabbix server configuration file to use these certificates:

```
vim /etc/zabbix/zabbix_server.conf
```

39. Make sure the following lines match our `lar-book-secure-db` database server's setup:

```
DBHost=lar-book-secure-db
DBName=zabbix
DBUser=zabbix
DBPassword=password
```

40. Now, make sure our SSL-related configuration matches our new files:

```
DBTLSConnect=verify_full
DBTLSCAFile=/var/lib/zabbix/ssl/cacert.pem
DBTLSCertFile=/var/lib/zabbix/ssl/zbx-srv.crt
DBTLSKeyFile=/var/lib/zabbix/ssl/zbx-srv.key
```

41. Also, make sure to add the right permissions to the SSL-related files:

```
chown -R zabbix:zabbix /var/lib/zabbix/ssl/
chmod 400 /var/lib/zabbix/ssl/zbx-srv.key
chmod 444 /var/lib/zabbix/ssl/zbx-srv.crt
chmod 444 /var/lib/zabbix/ssl/cacert.pem
```

42. Before restarting, make sure to edit the PHP timezone with the following command:

```
vim /etc/php-fpm.d/zabbix.conf
```

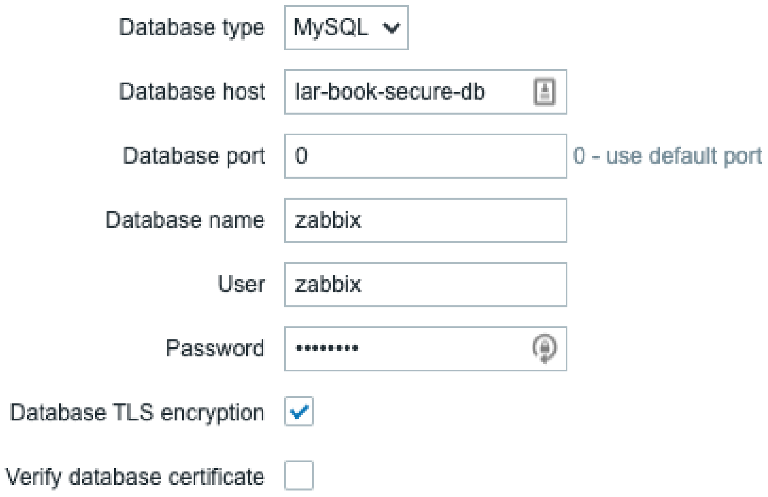
43. Edit the following line to match your timezone:

```
; php_value[date.timezone] = Europe/Riga
```

44. Start and enable the Zabbix server with the following:

```
systemctl restart zabbix-server zabbix-agent2 httpd
php-fpm
systemctl enable zabbix-server zabbix-agent2 httpd
php-fpm
```

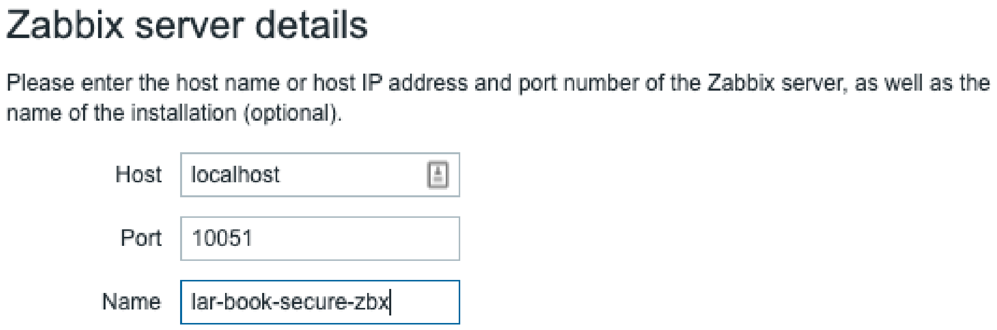
45. Then, navigate to the Zabbix frontend and fill in the right information as shown in the following screenshot:



Database type	MySQL
Database host	lar-book-secure-db
Database port	0 — 0 - use default port
Database name	zabbix
User	zabbix
Password	•••••••••
Database TLS encryption	✓
Verify database certificate	

Figure 11.17 – Zabbix frontend configuration, database step

46. When we press **Next step**, we need to fill out some more information:



Zabbix server details

Please enter the host name or host IP address and port number of the Zabbix server, as well as the name of the installation (optional).

Host	localhost
Port	10051
Name	lar-book-secure-zbx

Figure 11.18 – Zabbix frontend configuration, server details step

47. Then, after clicking **Next step**, **Next step**, and **Finish**, the frontend should now be configured and working.

How it works…

This was quite a long recipe, so let's break it down quickly:

- In *Steps 1* through *9*, we prepared our servers
- In *Steps 10* through *37*, we executed everything needed to create our certificates
- In *Steps 38* through *47*, we set up our Zabbix frontend for encryption

Going through all these steps, setting up your Zabbix database securely can seem like quite a daunting task, and it can be. Certificates, login procedures, loads of settings, and more can all add up to become very complicated, which is why I'd always recommend diving deeper into encryption methods before trying to set this up yourself.

If your setup requires encryption, though, this recipe is a solid starting point for your first-time setup. It works very well in an internal setting, as we are using private certificates. Make sure to renew them yearly, as they are only valid for 365 days.

All Zabbix components, except for communication between the Zabbix server and Zabbix frontend, can be encrypted as shown in the following diagram:

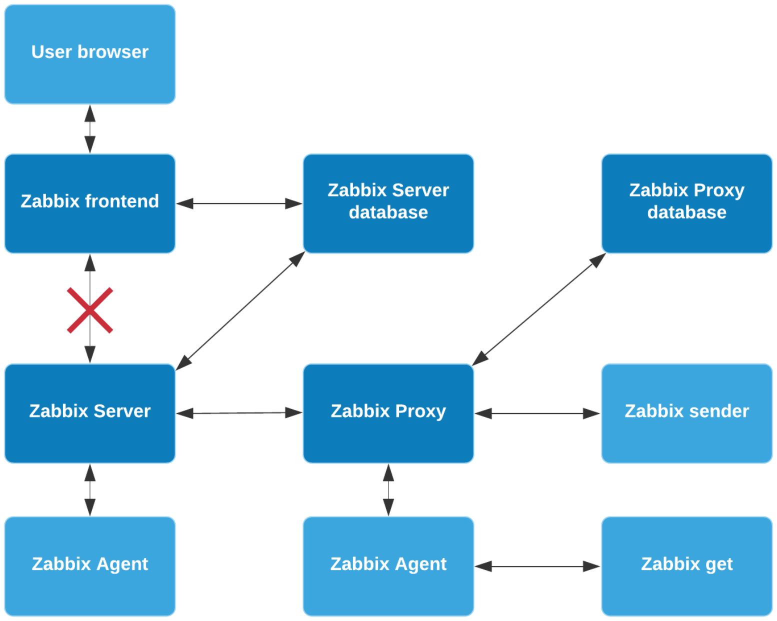


Figure 11.19 – Zabbix encryption scheme possibilites

We've set up encryption between the following:

- The Zabbix server and MariaDB
- The Zabbix frontend and MariaDB

This means that when our Zabbix server or frontend requests or writes data to our database, it will be encrypted. Because our Zabbix applications are running on a different server than our Zabbix database, this might be important. For example, our setup might look like this:

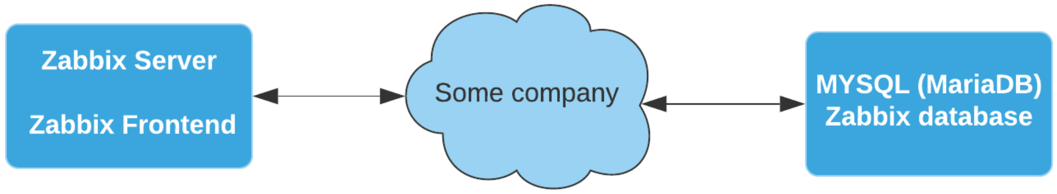


Figure 11.20 – Zabbix setup with external network diagram

Let's say the cloud is called **Some company** in a network that isn't managed by us. There are several switches and routers in this network that are used for numerous clients with their own VLANs. If one of these devices gets compromised somehow, all of our Zabbix data could be seen by others.

Even if the network equipment is ours, there might still be a compromised device in the network and our data can be seen. This is why you might want to add encryption, to add that extra layer of security. Whether it's against breaches in other companies and their network that you want to secure against or whether it's against your own breaches, securing your database as we did in this recipe might just save you from leaking all that data.

12
Bringing Zabbix to the Cloud with Zabbix Cloud Integration

For the last chapter, we have prepared something special. As a long-time Zabbix user, the importance of cloud integration for tools such as Zabbix has not gone unnoticed. For some people, the cloud can be daring, and so with this chapter, I want to show you just how easy it can be to start working with the most popular cloud providers and Zabbix.

We are going to start by talking about monitoring the **Amazon Web Services** (**AWS**) cloud with Zabbix. Then we will also see how the same things are done using Microsoft Azure so we can clearly see the differences.

After going through these cloud products, we'll also check out container monitoring with Docker, a very popular product that can also benefit greatly from setting up Zabbix monitoring. Follow these recipes closely and you will be able to monitor all of these products easily and work to extend the products using Zabbix. This chapter comprises the following recipes:

- Setting up AWS monitoring
- Setting up Microsoft Azure monitoring
- Building your Zabbix Docker monitoring

Technical requirements

As this chapter focuses on AWS, Microsoft Azure, and Docker monitoring, we are going to need a working AWS, Microsoft Azure, or Docker setup. The recipe does not cover how to set these up, so make sure to have your own infrastructure at the ready.

Furthermore, we are going to need our Zabbix server running Zabbix 5. We will call this server `zbx-home` in this chapter.

You can download the code files for this chapter from the following GitHub link: `https://github.com/PacktPublishing/Zabbix-5-Network-Monitoring-Cookbook/tree/master/chapter12`.

Setting up AWS monitoring

A lot of infrastructure is moving toward the cloud these days and it's important to keep an eye on this infrastructure as much as when it is your own hardware. In this recipe, we are going to discover how to monitor RDS instances and S3 buckets with our Zabbix setup.

Getting ready

For this recipe, we are going to need our AWS cloud with some S3 buckets and/or RDS instances in it already. Of course, we will also need our Zabbix server, which we'll call `zbx-home` in this recipe.

Then, last but not least, we will require some templates and hosts, which we can import. We can download the XML files here: `https://github.com/PacktPublishing/Zabbix-5-Network-Monitoring-Cookbook/tree/master/chapter12`.

Important note

Using Amazon CloudWatch is not free, so you will incur costs. Make sure you check out the Amazon pricing for AWS CloudWatch before proceeding: `https://aws.amazon.com/cloudwatch/pricing/`.

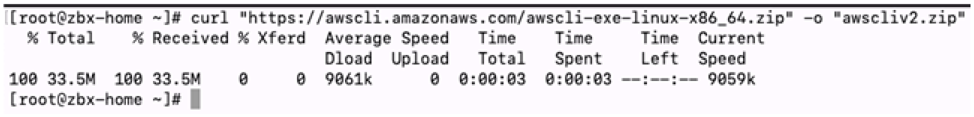
How to do it…

Setting up AWS monitoring might seem like a daunting task at first, but once we get a hold of the technique it's not that difficult. Let's waste no more time and check out one of the methods we could use:

1. Let's start by logging in to our Zabbix server with the hostname `zbx-home`.
2. On the Zabbix server, download the AWS CLI installation package with the following command:

   ```
   curl https://awscli.amazonaws.com/awscli-exe-
   linux-x86_64.zip -o "awscliv2.zip"
   ```

   The result will look like this:



```
[root@zbx-home ~]# curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 33.5M  100 33.5M    0     0  9061k      0  0:00:03  0:00:03 --:--:-- 9059k
[root@zbx-home ~]#
```

Figure 12.1 – Curl command execution for AWS CLI

3. If you haven't already done so, install `Unzip` with the following command:

   For RHEL-based systems:

   ```
   dnf install unzip
   ```

   For Debian-based systems:

   ```
   apt-get install unzip
   ```

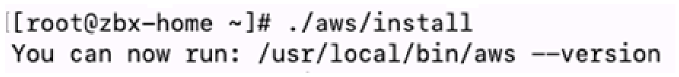
4. After installing the `Unzip` application, we can use it to unzip our AWS CLI installation package with the following command:

   ```
   unzip -q awscliv2.zip
   ```

5. We are now ready to install our AWS CLI with the following command:

   ```
   ./aws/install
   ```

The output will look like this:



```
[[root@zbx-home ~]# ./aws/install
 You can now run: /usr/local/bin/aws --version
```

Figure 12.2 – AWS CLI install command execution

6. Next up, log in to your AWS account by navigating to the following URL in your browser: `https://signin.aws.amazon.com/`.
7. Once logged in, we can navigate to **My Security Credentials** listed in your user profile in the top-right corner:

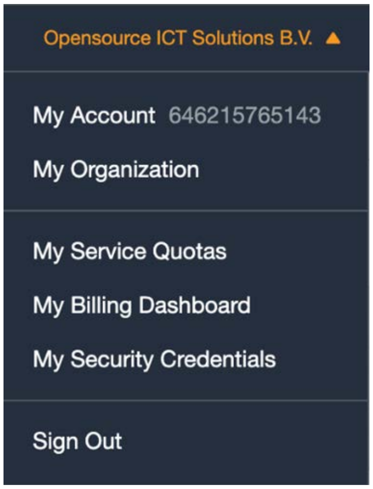
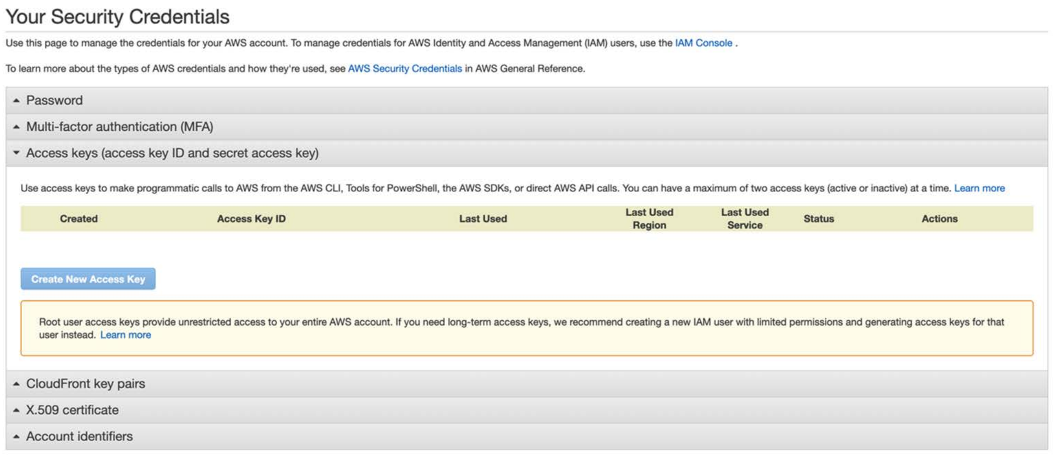


Figure 12.3 – AWS web frontend user profile

8. At the next page, click on **Access keys (access key ID and secret access key)**. This will open up the following screen:



Your Security Credentials

Use this page to manage the credentials for your AWS account. To manage credentials for AWS Identity and Access Management (IAM) users, use the IAM Console .

To learn more about the types of AWS credentials and how they're used, see AWS Security Credentials in AWS General Reference.

- Password
- Multi-factor authentication (MFA)
- Access keys (access key ID and secret access key)

Use access keys to make programmatic calls to AWS from the AWS CLI, Tools for PowerShell, the AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time. Learn more

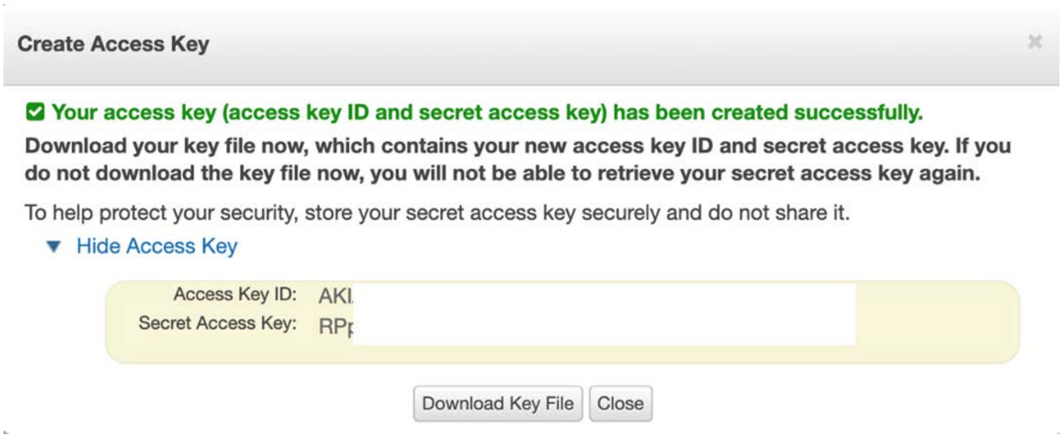
Created	Access Key ID	Last Used	Last Used Region	Last Used Service	Status	Actions

Root user access keys provide unrestricted access to your entire AWS account. If you need long-term access keys, we recommend creating a new IAM user with limited permissions and generating access keys for that user instead. Learn more

- CloudFront key pairs
- X.509 certificate
- Account identifiers

Figure 12.4 – AWS access keys page

9. Click on **Create New Access Key** to create a new access key. You should see the following:



Create Access Key

Your access key (access key ID and secret access key) has been created successfully.

Download your key file now, which contains your new access key ID and secret access key. If you do not download the key file now, you will not be able to retrieve your secret access key again.

To help protect your security, store your secret access key securely and do not share it.

Access Key ID: AKI[redacted]
Secret Access Key: RPr[redacted]

Figure 12.5 – AWS access key details

10. Make sure to note down the access key ID and the secret access key as we will need these later.

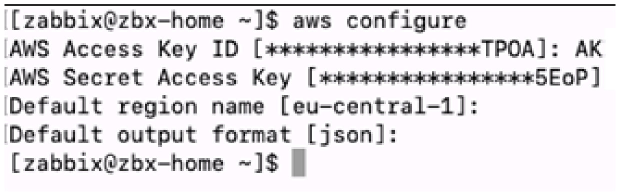
11. Back at the Linux CLI, let's log in to our `zabbix` user. It is important to execute the following commands on the correct Linux users, so switch to the `zabbix` user with the help of the following command:

```
sudo su -s /bin/bash zabbix
```

12. Under the `zabbix` user, enter the following command:

```
aws configure
```

13. Fill in the **Access Key ID** and **Secret Access Key** fields we noted down earlier. Also, make sure to change your default region to your preferred region and set the output format to `json`. It will look like this:



```
[zabbix@zbx-home ~]$ aws configure
AWS Access Key ID [****************TPOA]: AK
AWS Secret Access Key [****************5EoP]
Default region name [eu-central-1]:
Default output format [json]:
[zabbix@zbx-home ~]$
```

Figure 12.6 – AWS configure command on the Linux CLI

14. Configuration under the **zabbix** user is now complete and you can switch back with the following command:

```
su - root
```

15. Now, switch directories with the help of the following command:

```
cd /usr/lib/zabbix/externalscripts
```

16. Then, create a new script file called `aws_script.sh` in this directory with the help of the following command:

```
vim aws_script.sh
```

17. Add the following lines to the script and save the file:

```
#!/bin/bash
instance=$1
metric=$2
now=$(date +%s)
aws cloudwatch get-metric-statistics --metric-name
$metric --start-time "$(echo "$now - 300" | bc)"
```

```
--end-time "$now" --period 300  --namespace AWS/RDS
--dimensions Name=DBInstanceIdentifier,Value="$instance"
--statistics Average
```

18. Make sure the `zabbix` Linux user can execute this script by setting the permissions with the following command:

```
chown zabbix:zabbix aws_script.sh
chmod 700 aws_script.sh
```

19. We also need to add functionality to the Zabbix agent. Do this by adding user parameters. On the Zabbix server Linux CLI, switch directory with the help of the following command:

```
cd etc/zabbix/zabbix_agentd.d/
```

20. Now, create a new file with the following command:

```
vim userparameter_aws.conf
```

21. Then, insert the following lines:

```
#Buckets
UserParameter=bucket.discovery,aws s3api list-buckets
--query "Buckets[]"
UserParameter=bucket.get[*], aws s3api list-objects
--bucket "$1" --output json --query "[sum(Contents[].
Size), length(Contents[])]"

#RDS
UserParameter=rds.discovery,aws rds describe-db-instances
--output json --query "DBInstances"
UserParameter=rds.metrics.discovery[*],aws cloudwatch
list-metrics --namespace AWS/RDS --dimensions
Name=DBInstanceIdentifier,Value="$1" --output json
--query "Metrics"
```

22. Restart your Zabbix agent as follows:

```
systemctl restart zabbix-agent
```

Tip

Using user parameters will work with both Zabbix agent and Zabbix agent 2. This means that irrespective of whether you are required to run the old or new agent, you can set up AWS monitoring without any issues.

23. Let's now confirm whether it is working. Navigate to your Zabbix frontend and go to **Configuration | Templates**.
24. Click on the blue **Import** button and add the file `template_aws.xml`, which we can download from Packt's GitHub repository at the following link: `https://github.com/PacktPublishing/Zabbix-5-Network-Monitoring-Cookbook/blob/master/chapter12/template_aws.xml`.
25. Then, navigate to **Configuration | Hosts**, click on the blue **Import** button, and import the `aws_hosts.xml` file, which we can download from Packt's GitHub repository at the following link: `https://github.com/PacktPublishing/Zabbix-5-Network-Monitoring-Cookbook/blob/master/chapter12/aws_hosts.xml`.
26. This will import an AWS template called **Template AWS discovery** and add two AWS hosts – **Bucket discovery** and **RDS discovery**.

That's the final step for this recipe. Check out the new templates, hosts, and how it all fits together in the *How it works…* section.

How it works…

Now that we've done all the setup at the Linux CLI level and have imported the templates and hosts, let's see what they do. Under **Configuration | Hosts**, we can now see two new hosts.

First, let's look at the **AWS Bucket discovery** host. This host will discover our AWS buckets, such as the S3 bucket. We can see that the host only has one configuration, which is a discovery rule. If we go to this discovery rule, called **Bucket discovery**, we can see that it uses the item key `bucket.discovery`. This item key is defined by us in the user parameters and executes the following command:

```
aws s3api list-buckets --query "Buckets[]"
```

We do this to get every single AWS bucket and put it in the `{#NAME}` LLD macro.

Furthermore, there are three item prototypes on the discovery rule. The most important item prototype is the one called `{#NAME}`, which will use the Zabbix agent to execute the user parameter with the `bucket.get` item key for every bucket found for the `{#NAME}` LLD macro we just discovered in the discovery rule.

The `bucket.get` item key then executes the command we defined, which is the following:

```
aws s3api list-objects --bucket "$1" --output json --query
"[sum(Contents[].Size), length(Contents[])]"
```

This command gets all of our information from the AWS buckets found and adds them to an item on our **AWS Bucket discovery** host. We can then use dependent items as in the two examples cited to extract the information from the item and put them in different items. Check out *Chapter 2*, *Setting up Zabbix monitoring*, for more information on dependent items.

We also have our **AWS RDS discovery** host, which will discover AWS RDS instances. If we check out the host, we can see only one configuration, which is the discovery rule **Instance discovery**. This discovery rule has one host prototype on it to create new hosts from every RDS found in our AWS setup. It uses the `rds.discovery` item key, which executes the following command:

```
aws rds describe-db-instances --output json --query
"DBInstances"
```

This puts every RDS instance in the `{#NAME}` LLD macro and creates a new Zabbix host for it. After creating the host, it will also make sure that the `AWS RDS discovery` template is linked to the new host. The template has a discovery rule to get some RDS metrics from the RDS instance using the item key `rds.metrics.discovery`, which executes the following command:

```
aws cloudwatch list-metrics --namespace AWS/RDS --dimensions
Name=DBInstanceIdentifier,Value="$1" --output json --query
"Metrics"
```

What we are doing here is using the AWS CLI to execute commands on our Zabbix server through the Zabbix agent User parameters. The Zabbix agent runs the AWS CLI command and retrieves data from AWS CloudWatch. These metrics are stored in the database and then used in our items.

Now that we've seen how to use AWS CloudWatch, we can extend the Zabbix agent further by adding extra user parameter parameters or by creating more dependent items and using this information.

There's more...

It takes time to start monitoring with AWS CloudWatch as we need a good understanding of the AWS CLI commands with the use of CloudWatch. When you use the templates provided in this recipe as a basis, you have a solid foundation on which to build.

Make sure to check out the AWS documentation for more information on the commands that we can use at the following link: `https://docs.aws.amazon.com/cli/latest/reference/#available-services`.

Setting up Microsoft Azure monitoring

Microsoft Azure Cloud is a big player in the cloud market these days and it's important to keep an eye on this infrastructure as much as when it is your own hardware. In this recipe, we are going to discover how to monitor Azure instances with our Zabbix setup.

Getting ready

For this recipe, we are going to need our Azure Cloud with an Azure DB instance in it already. The recipe does not cover how to set up an Azure DB instance, so make sure to have this in advance. We will also need our Zabbix server, which we'll call `zbx-home` in this recipe.

We have split up the Azure CLI installation aspect into RHEL-based and Debian-based systems. Make sure to use the guide that is appropriate for you.

Then, last but not least, we will require some templates and hosts, which we can import. We can download the XML files here: `https://github.com/PacktPublishing/Zabbix-5-Network-Monitoring-Cookbook/tree/master/chapter12`.

How to do it...

For Azure monitoring we face some of the same techniques as we do for AWS monitoring. It is a bit daring, but easier than it looks. Let's check out one of the techniques we can use for Azure monitoring:

1. First things first, log in to the Zabbix server Linux CLI.

2. We are going to install the Azure CLI on our Zabbix server. Let's cover RHEL-based systems first.

RHEL-based Azure CLI installation

First things first, let's check out the installation process on a RHEL based host.

1. Let's import the Azure repository key:

```
rpm –import https://packages.microsoft.com/keys/
microsoft.asc
```

2. Then, create the repository file with the following command:

```
echo -e "[azure-cli]
name=Azure CLI
baseurl=https://packages.microsoft.com/yumrepos/azure-cli
enabled=1
gpgcheck=1
gpgkey=https://packages.microsoft.com/keys/microsoft.asc"
| sudo tee /etc/yum.repos.d/azure-cli.repo
```

3. Now, run the installation command to install Azure CLI with the following command:

```
dnf -y install azure-cli
```

Debian-based Azure CLI Installation:

Now, let's check out the installation process on a Debian-based host:

1. Install the required dependent packages with the following command:

```
sudo apt-get install ca-certificates curl apt-transport-
https lsb-release gnupg
```

2. Now, download and install the Microsoft signing key with the help of the following command:

```
curl -sL https://packages.microsoft.com/keys/microsoft.
asc |
    gpg --dearmor |
    sudo tee /etc/apt/trusted.gpg.d/microsoft.gpg > /dev/
null
```

3. Then, add the Azure CLI repository with the following command:

```
AZ_REPO=$(lsb_release -cs)
echo "deb [arch=amd64] https://packages.microsoft.com/
repos/azure-cli/ $AZ_REPO main" |
    sudo tee /etc/apt/sources.list.d/azure-cli.list
```

4. Then, update the repository information and install the Azure CLI with the following command:

```
sudo apt-get update
sudo apt-get install azure-cli
```

Setting up Azure monitoring

Now that we've installed the Azure CLI we can start setting up the Azure monitoring:

1. We will need to log in as the zabbix user. Make sure to use a bash terminal with the zabbix user. Use the following command:

```
sudo su -s /bin/bash zabbix
```

2. As the zabbix user, we can now use the following command to log in to the Azure CLI:

```
az login
```

3. You will be asked to provide Azure with the key and a token shown in the Azure CLI. Use the **Key** and **Token** options on the following web page: `https://aka.ms/devicelogin`.
4. Once you've logged in successfully, log out from the zabbix user with the help of the following command:

```
su - root
```

5. Now, let's add some functionality to the Zabbix agent on our Zabbix server by using user parameters.
6. Still on the Zabbix server CLI, switch directory to `/etc/zabbix/zabbix_agentd.d/` with the help of the following command:

```
cd /etc/zabbix/zabbix_agentd.conf
```

7. Create a new file with the following command:

```
vim userparameter_azure.conf'
```

8. In this file, insert the following lines:

```
UserParameter=azure.db.discovery,az resource list
--resource-type "Microsoft.DBforMySQL/servers"
```

9. We will also require a Zabbix `externalscript`. Let's switch directory with the following command:

```
cd /usr/lib/zabbix/externalscripts/
```

10. Then, create an `azure_script.sh` script by executing the following command:

```
vim azure_script.sh
```

11. We then need to add the following lines:

```
#!/bin/bash
id=$1
metric=$2
curr=$(date --utc +%Y-%m-%dT%H:%M:%SZ)
new=$(date --utc -d "($curr) -5minutes" +%Y-%m-
%dT%H:%M:%SZ)

az monitor metrics list --resource $id --metric "$metric"
--start-time "$new" --end-time "$curr" --interval PT5M
```

12. Save the file and make sure to set the correct permissions with the help of the following command:

```
chown zabbix:zabbix azure_script.sh
chmod 700 azure_script.sh
```

13. We can now restart our Zabbix agent with the following command:

```
systemctl restart zabbix-agent
```

14. We can now configure our Zabbix frontend by importing a template and a host.

15. Open your Zabbix frontend and navigate to **Configuration | Templates**. Here we need to click on the blue **Import** button and import the `template_azure.xml` file, which we can download at `https://github.com/PacktPublishing/Zabbix-5-Network-Monitoring-Cookbook/blob/master/chapter12/template_azure.xml`.
16. We will also need to import a host. Go to **Configuration | Hosts** and click on the blue **Import** button to import the `azure_host.xml` file, which can be downloaded here: `https://github.com/PacktPublishing/Zabbix-5-Network-Monitoring-Cookbook/blob/master/chapter12/azure_host.xml`.

That's the final step for this recipe. Let's stay with the frontend and check out the new templates, hosts, and how it all fits together in the *How it works…* section.

How it works…

If you've followed the recipe regarding AWS monitoring, you might think that Azure monitoring works the same. In fact, we employ a different Zabbix monitoring technique here, so let's check it out.

After adding the template and the host to our Zabbix server, we can go to **Configuration | Hosts** and check out our new host. We can see one new host here called **Discover Azure DBs**. When we examine this host, we can see that it only has one configuration, which is a discovery rule called **Discover Azure DBs**, like the host.

Under this discovery rule, we find a single host prototype called `{#NAME}`. This host prototype uses the discovery rule's `azure.db.discovery` item key to execute the following command:

```
az resource list --resource-type "Microsoft.DBforMySQL/servers"
```

With this command, the `{#NAME}` LLD macro is filled with our Azure DB instances and we create a new host for every single Azure DB instance found. The new hosts then get the `Azure DB` template added to it.

When we check out the template under **Configuration | Templates**, we can see that it has 12 items. Let's check out the item **CPU Load**. This item is of the **External check** type, which uses its `azure_script.sh[{$ID},cpu_percent]` item key to execute the external script `azure_script.sh` and feed it the parameters `{$ID}` and `cpu_perecent`. The script that is executed uses the Azure CLI to retrieve the values and we put this value in the Zabbix database.

There's more...

We can discover way more from Azure using the method applied in this recipe. The script we employed is used to get metrics from Azure, which we can put in items by feeding them the correct parameters.

Check out the Azure CLI for more information on the metrics retrieved using the script at the following link: `https://docs.microsoft.com/en-us/cli/azure/monitor/metrics?view=azure-cli-latest`.

Building your Zabbix Docker monitoring

Since the release of Zabbix 5, monitoring your Docker containers became a lot easier with the introduction of Zabbix agent 2 and plugins. Using Zabbix agent 2, we are now able to monitor our Docker containers out of the box.

In this recipe, we are going to see how to set this up and how it works.

Getting ready

For this recipe, we require some Docker containers. We won't go over the setup of Docker containers, so make sure to do this yourself. Furthermore, we are going to need Zabbix agent 2 installed on those Docker containers. Zabbix agent does not work in relation to this recipe; Zabbix agent 2 is required.

We also need our Zabbix server to actually monitor the Docker containers. We will call our Zabbix server `zbx-home`.

How to do it...

Let's waste no more time and dive right into the process of monitoring your Docker setup with Zabbix:

1. First things first, log in to the Linux CLI of your Docker container.
2. Add the repository for installing Zabbix components:

   For RHEL-based systems:

   ```
   rpm -Uvh https://repo.zabbix.com/zabbix/5.2/rhel/8/
   x86_64/zabbix-release-5.2-1.el8.noarch.rpm
   ```

   ```
   dnf clean all
   ```

For Debian-based systems:

```
wget https://repo.zabbix.com/zabbix/5.2/ubuntu/pool/
main/z/zabbix-release/zabbix-release_5.2-1+ubuntu20.04_
all.deb
dpkg -i zabbix-release_5.2-1+ubuntu20.04_all.deb
apt update
```

3. Now, install Zabbix agent 2 with the following command:

   For RHEL-based systems:

   ```
   dnf install zabbix-agent2
   ```

   For Debian-based systems:

   ```
   apt-get install zabbix-agent2
   ```

4. Following installation, make sure to edit the configuration file of the newly installed Zabbix agent 2 with the help of the following command:

   ```
   vim /etc/zabbix/zabbix_agent2.conf
   ```

5. Find the line that says `Server` and add your Zabbix server IP address to the file as follows:

   ```
   Server=10.16.16.102
   ```

6. Make sure to save the file and then restart Zabbix agent 2 with the following command:

   ```
   systemctl restart zabbix-agent2
   ```

7. Now, we need to add the zabbix user to the Docker group by executing the following command:

   ```
   gpasswd -a zabbix docker
   ```

8. Now, navigate to your Zabbix server frontend. Go to **Configuration | Hosts** and create a new host called Docker containers.
9. Make sure to link the `Docker` template to the host.

That's all there is to monitoring Docker containers with Zabbix server. Let's now see how it works.

How it works…

Docker monitoring in Zabbix these days is easy, due to the new Zabbix agent 2 support and default templates. On occasion though, a default template does not cut it, so let's break down the items used.

Almost all the items we can see on our host are dependent items, most of which are dependent on the master item, `Docker: Get info`. This master item is the most important item on our Docker template. It executes the `docker.info` item key, which is built into the new Zabbix agent 2. This item retrieves a list with all kinds of information from our Docker setup. We use the dependent items and preprocessing to get values we want from this master item.

The Docker template also contains two Zabbix discovery rules, one to discover Docker images, and one to discover Docker containers. If we check out the discovery rule for Docker containers called `Containers discovery`, we can see what happens. Our Zabbix Docker host will use the `docker.containers.discovery` item key to find every container and put this in the `{#NAME}` LLD macro. In the item prototypes, we then use this `{#NAME}` LLD macro to discover statistics with another master item, such as `docker.container_info`. From this master item, we then use the dependent items and preprocessing again to include this information in other item prototypes as well. We are now monitoring a bunch of statistics straight from our Docker setup.

If you want to get values from Docker that aren't in the default template, check out the information collected with the master items on the template. Use a new dependent item (prototype) and then use preprocessing to get the correct data from the master item.

There's more…

If you want to learn more about the Zabbix agent 2 Docker item keys, check out the supported item key list for Zabbix agent 2 under the Zabbix documentation:

`https://www.zabbix.com/documentation/current/manual/config/items/itemtypes/zabbix_agent/zabbix_agent2?s[]=docker.s`

Other Books You May Enjoy

Leave a review - let other readers know what you think

Index

A

Active agent autoregistration
  about 222
  working with 187-192
active Zabbix proxies
  hosts, monitoring with 216
  working with 212-214
advanced user authentication
  with SAML 37-41
alerts
  customizing 109-111
  keeping, effective 105-108
  setting up 97-105
application prototypes 121
Atlassian Opsgenie
  integrating, with Zabbix 262-271
AWS CloudWatch
  monitoring 378
AWS documentation
  reference link 378
AWS monitoring
  setting up 370-376
  working 376, 377

B

Bucket discovery 376
built-in macro
  defining 129
  working 130, 131

C

calculated items
  working with 62-64, 67
CentOS 7
  Zabbix backend, upgrading on 317
  Zabbix database, upgrading on 319
  Zabbix setup, upgrading on 322-324
Cloud Hoster 32
command-line interface
    (CLI) 90, 150, 206

D

dashboards
  creating, to obtain right
    overview 162-167

database monitoring
  setting up 73-75
  working 76
Debian-based Azure CLI
  installation process 379, 380
Debian-based installation
  of PostgreSQL TimescaleDB 350
dependent items
  working with 64-68
Discover Azure DBs 382
Discovery
  using, with Zabbix proxies 220, 222
Discovery rules 220, 221
Distributed Denial of Service (DDoS) 162
Docker containers 384

E

external checks
  creating 68, 69
  working 70

G

general application 120
Global view dashboard 14
graphs
  creating, to access visual data 146-153

H

host
  disabling, from Zabbix maps 299-306
  enabling, from Zabbix maps 299-306
  monitoring, with active proxy 216
  monitoring, with passive proxy 216
  monitoring, with Zabbix proxy 215
HTTP agent monitoring
  setting up 76-78
  working 78, 79

I

Intelligent Platform Management
    Interface (IPMI) 218
Internet Protocol (IP) 181
item values
  altering, with Zabbix
    preprocessing 79-85

J

Java Management Extensions (JMX) 218
JMX monitoring
  reference link 73
  setting up 70-72
  working 72
JMX objects
  discovering 199-202
  working 202, 203
jumphost
  building, with Python 282-290
  building, with Zabbix API 282-290

L

LLD
  using, on templates 131-140
Long-Term Support (LTS) 321
Low-Level Discovery (LLD) macro 153

M

macros
  built-in macro 129
  reference link 131
  setting up 127
  types 129
  user macro 127, 128
  working 129, 130
maintenance periods
  creating, as Zabbix User 291-298
Managed Service Provider (MSP) 169, 219
Management Information Base (MIB) 122
Man in the Middle (MITM) 356
maps
  creating, to monitor
    infrastructure 153-162
MBeans
  reference link 203
Microsoft Azure monitoring
  Debian-based Azure CLI
    installation 379, 380
  RHEL-based Azure CLI installation 379
  setting up 378-381
  working 382
Microsoft Teams alerting
  setting up, with Zabbix 243-250
MySQL database
  tuning, for maintaining Zabbix
    performance 333, 334, 337
MySQL partitioning
  setting up, for Zabbix database 340-348
  working 348, 349

N

New Values Per Second (NVPS) 15, 171

O

Object Identifier (OID) 182

P

page visitor count
  triggering on 90-95
passive proxy
  hosts, monitoring with 216
  working with 209-211
PostgreSQL TimescaleDB
  configuration 352-354
  Debian-based installation 350
  RHEL-based installation 351
PostgreSQL TimescaleDB documentation
  reference link 356
PostgreSQL TimescaleDB functionality
  using 350
  working 355, 356
proxy
  monitoring, from Zabbix
    frontend 227, 229
  monitoring, remotely from
    Zabbix server 225-228
  monitoring, with Zabbix 223-228
Python
  jumphost, building with 282-290

R

regex101
  URL 85
RHEL-based Azure CLI
  installation process 379
RHEL-based installation
  of PostgreSQL TimescaleDB 351

S

SAML
  using, for advanced user authentication 37-41
Secure Shell (SSH) 89
service level agreement (SLA) 22
setup, Zabbix agent 2 monitoring
  Zabbix agent 2, installing 44, 45
  Zabbix agent, using in Active mode 47-49
  Zabbix agent, using in Passive mode 45-47
Simple Mail Transfer Protocol (SMTP) 102
Simple Network Management Protocol (SNMP) 217
Slack alerting
  setting up, with Zabbix 232-242
Slack API
  URL 243
SNMP monitoring
  working with 51-55
SSH service monitoring 89, 93

T

Telegram bots
  reference link 262
  using, with Zabbix 251-262
template applications
  setting up 118, 119
template items
  creating 121-125
templates
  LLD, using on 131-140
template triggers
  creating 125-127
trappers
  creating 56, 59, 60
  working 62
trigger expressions
  reference link 97
triggers
  about 97
  multiple items, using in 92, 96
  setting up 88

U

Ubuntu 16.04
  Zabbix backend, upgrading on 317
  Zabbix database, upgrading on 320, 321
  Zabbix setup, upgrading on 326, 327
user groups
  creating, for login into Zabbix frontend 26-31
user macro
  defining 127, 128
users
  creating, for Zabbix frontend 31-36

V

virtual private network (VPN) 219

W

Windows performance counter discovery
  Low-Level Discovery (LLD) 198
  using 193-197
  working 198, 199

Z

Zabbix
  Atlassian Opsgenie, integrating with 262-271
  download link 206
  installation link 206
  Microsoft Teams alerting, setting up with 243-250
  Slack alerting, setting up with 232-242
  Telegram bots, using with 251-262
Zabbix 5.2, upgrade information
  reference link 329
Zabbix agent 2
  reference link 51
Zabbix agent 2 Docker item keys
  reference link 385
Zabbix agent 2 monitoring
  active agent, working 50
  passive agent, working 50
  setting up 44
  working 50
Zabbix Agent discovery
  setting up 176-180
  working 181
Zabbix and Opsgenie integration
  reference link 271
Zabbix, and SNMP OIDs/MIBs
  reference link 125
Zabbix API
  jumphost, building with 282-290
  reference link 282
  used, for extending functionality 274-282
Zabbix API libraries
  reference link 282
Zabbix backend
  upgrading, from older PHP versions to PHP 7.2 or higher 316, 318
  upgrading, on CentOS 7 317
  upgrading, on Ubuntu 16.04 317
Zabbix database
  MySQL partitioning, setting up for 340-348
  upgrading, from older MariaDB versions to MariaDB 10.5 318, 321
  upgrading, on CentOS 7 319
  upgrading, on Ubuntu 16.04 320, 321
Zabbix discovery action, HostMetadata
  reference link 193
Zabbix, discovery rules
  reference link 181
Zabbix Docker monitoring
  building 383, 384
  working 385
Zabbix documentation
  reference link 42
Zabbix encryption
  reference link 211
Zabbix frontend
  setting up 6-12
  using 12-19
  working 12
Zabbix housekeeper
  maintaining 332, 333
  working 336, 337
Zabbix inventory
  setting up 167, 168
  working 169
Zabbix LLD documentation
  reference link 140
Zabbix maintenance periods
  setting 308-310
  working 311

Zabbix maps
  host, disabling from 299-306
  host, enabling from 299-306
Zabbix MySQL database
  securing 356-365
  working 366, 367
Zabbix performance
  maintaining, over time 329, 334
  maintaining, with tuning MySQL database 333-337
Zabbix preprocessing
  reference link 85
  used, for altering item values 79-85
Zabbix processes
  maintaining 329-331
  working 335, 336
Zabbix proxies
  Discovery, using with 220-222
  hosts, monitoring with 215-218
  installation link 209
  monitoring 223
  reference link 219
  setting up 206-208
  versions 208
  working 208, 209
Zabbix raw items 120
Zabbix reporting
  Action log 172
  Audit 172
  Availability report 171
  Notifications 173
  System information 170
  Trigger top 100 171
  working through 169
Zabbix server
  default states 16
  installing 2-5
  navigating 19-25
  working 6
Zabbix setup
  backing up 311-314
  upgrading 321, 328, 329
  upgrading, on CentOS 7 322-324
  upgrading, on Ubuntu 16.04 326, 327
  working 315
Zabbix simple checks
  creating 56-59
  reference link 61
  working 61
Zabbix SNMP discovery
  setting up 182-185
  working 186
Zabbix template
  creating 114-117
  nesting 140-143
  reference link 117
Zabbix trigger expression
  host 93
  item key 93
  trigger function 94
  value 94
Zabbix User
  maintenance periods, creating as 291-298
Zabbix webhook 243
Zabbix webhook connector
  reference link 251