



Quick answers to common problems

Magento 2 Cookbook

Over 50 practical recipes that will help you realize the full potential of Magento in order to build a professional online store

Foreword by Guido Jansen, Magento Community Manager

Ray Bogman
Vladimir Kerkhoff

[PACKT] open source[®]
PUBLISHING
community experience distilled

Magento 2 Cookbook

Over 50 practical recipes that will help you realize the full potential of Magento in order to build a professional online store

Ray Bogman

Vladimir Kerkhoff



open source 
community experience distilled

BIRMINGHAM - MUMBAI

Magento 2 Cookbook

Copyright © 2016 Packt Publishing

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

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

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

First published: March 2016

Production reference: 1210316

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

ISBN 978-1-78588-706-2

www.packtpub.com

Credits

Authors

Ray Bogman
Vladimir Kerkhoff

Project Coordinator

Nikhil Nair

Reviewer

Kevin Schroeder

Proofreader

Safis Editing

Commissioning Editor

Wilson D'souza

Indexer

Monica Ajmera Mehta

Acquisition Editor

Reshma Raman

Graphics

Disha Haria

Content Development Editor

Parshva Sheth

Production Coordinator

Arvindkumar Gupta

Technical Editors

Pranil Pathare
Manthan Raja

Cover Work

Arvindkumar Gupta

Copy Editor

Tasneem Fatehi

Foreword

Ray and Vladimir have been THE BEST thing to happen to Magento ever since X.Commerce.

Ok, that may be a slight exaggeration..., but it remains true that besides the software itself, Magento has always thrived on the dedication and enthusiasm of its large community. Ray and Vladimir are prime examples of the literally hundreds of developers, merchants, project managers, extension providers, event organizers, and platform evangelists that contribute every day to help community members make the most of their Magento shop.

I met Ray and Vladimir in the early days of Magento and got to know them as two highly engaged Magento fanatics. They built shops, extended the functionality, and pushed Magento to its limits. Failed and succeeded more often than anyone can count. More importantly, they were always generous enough to share their learning with the community through blog posts, webinars, events, and as Magento Doctors!

After a not-to-be-specified number of days, the announcement—we finally have Magento 2! If there are two people in our community capable of taking a deep dive into this new amazing platform and giving us practical recipes, it will be Ray and Vladimir.

I look forward to seeing what we can make happen in this new chapter for Magento. A lot of things will change and this book will help you get started with over 50 recipes.

One thing, however, will never change: don't touch the core...

Guido Jansen
Magento Community Manager

About the Authors

Ray Bogman is an IT professional and Magento evangelist from the Netherlands. He started working with computers in 1983 as a hobby at first. In the past, he has worked for KPN, a large Dutch Telecom company, as a senior security officer.

He has been the CTO of Wild Hibiscus, Netherlands, since 2010, and cofounder and business creator of Yireo until 2011, Jira ICT since 2005, and CTO of SupportDesk B.V., which he cofounded in 2011.

At SupportDesk B.V., he is a Magento, Joomla, OroCRM, web / server / mobile performance specialist, and security evangelist. His focus during the day is business/product development and training webmasters and consultants about the power of Magento, from the basics up to an advanced level. He has trained over 1,000 Magento and 750 Joomla experts worldwide since 2005.

At Magento events such as Magento Developers Paradise, Meet Magento, and MagentoLive, he has been a regular speaker since 2009.

He has participated in reviewing *Mastering Magento* (2012), *Mastering Magento Theme Design* (2014), *Magento Administration Guide* (2014), *Learning Magento Theme Development* (2014), and the video, *Mastering Magento* (2013), all by Packt Publishing.

The writing of this book was a big thing for me. I spent every free minute with passion in writing this. I had many doubts to overcome this big challenge as a dyslexic person. But I never thought that writing this would give me lots of self-esteem, peace, and joy. This would not have happened without the loving support and patience of my wife, Mette, and daughter, Belize. Joining forces with my co-writer, Vladimir, was a lot of fun. Sharing thoughts and insights about our Magento passion was great.

Last but not least, I would like to thank all the people who made writing this possible.

Vladimir Kerkhoff is an experienced IT professional, 4-times Magento Certified Developer from the Netherlands, and working with Magento since late 2009.

Currently, Vladimir is the founder/owner of Genmato BV, a Magento extension development company. He is also available for cool freelance projects. Vladimir has had a long career in the IT industry. He started his career in 1992 in an IT company, managing customer corporate networks (Bionet, Novell Netware, and Windows Server). In 1996, he was the cofounder and CTO of Perfect InterNetworking Solutions (PINS), a large Dutch-managed hosting provider. After leaving PINS in 2009, he cofounded eFulFillers offering web shop fulfilment services based on the Magento platform.

The writing of this book was a nice new experience, forcing myself to dive into the new Magento 2 code and learn the new methods that are available.

About the Reviewer

Kevin Schroeder has more years of experience in software development and consulting than he would like to admit. He was worked at several notable companies, including Zend as a consultant and evangelist and Magento as a consultant with the Expert Consulting Group. After getting his first taste in browser testing, he realized that the current tooling amounted to cruel and unusual punishment and founded the Magium open source project under his company, 10n Software, with the purpose of making automated browser testing almost enjoyable and actually useful beyond just browser testing.

When he's not working on software, he is sleeping. When he's not sleeping, he is recording music (Coronal Loop Safari and Loudness Wars) or writing books (*The IBMi Programmer's Guide to PHP*, *You want to do WHAT with PHP?*, and *Advanced Guide to PHP on the IBMi*, all published by MC Press).

I would like to thank my wife, Laurie, and my three young 'uns who, after having me in their family, will have no problems coping with the rigors of adult life. Supreme thanks to life's Author, about whom many books have been written and none have been sufficient.

www.PacktPub.com

eBooks, discount offers, and more

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

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



<https://www2.packtpub.com/books/subscription/packtlib>

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

Why Subscribe?

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

Table of Contents

Preface	v
Chapter 1: Installing Magento 2 on Apache and NGINX	1
Introduction	1
Installing Apache	2
Installing NGINX	5
Installing PHP-FPM	7
Installing HHVM	13
Installing MySQL	18
Installing Magento 2	21
Installing Magento 2 on Hypernode	33
Managing Magento 2 on Docker	38
Chapter 2: Magento 2 System Tools	43
Introduction	43
Installing Magento 2 sample data via GUI	47
Installing Magento 2 sample data via the command line	53
Managing Magento 2 indexes via the command line	56
Managing Magento 2 cache via the command line	59
Managing Magento 2 backup via the command line	64
Managing Magento 2 set mode (MAGE_MODE)	68
Transferring your Magento 1 database to Magento 2	71
Chapter 3: Enabling Performance in Magento 2	79
Introduction	79
Configuring Redis for backend cache	80
Configuring Memcached for session caching	89
Configuring Varnish as the Full Page Cache	93
Configuring Magento 2 with CloudFlare	99

Configuring optimized images in Magento 2	107
Configuring Magento 2 with HTTP/2	111
Configuring Magento 2 performance testing	118
<u>Chapter 4: Creating Catalogs and Categories</u>	<u>125</u>
Introduction	125
Create a Root Catalog	127
Create subcategories	134
Manage attribute sets	137
Create products	141
Manage products in a catalog grid	150
<u>Chapter 5: Managing Your Store</u>	<u>155</u>
Introduction	155
Creating shipping and tax rules	156
Managing customer groups	168
Configuring inventories	170
Configuring currency rates	174
Managing advanced pricing	175
<u>Chapter 6: Creating a Magento 2 Theme</u>	<u>181</u>
Introduction	181
Creating a new theme	182
Changing a layout XML of a Magento 2 module	190
Adding CSS/JS to pages	196
Using Grunt for CSS changes	198
Adding static blocks to pages through layout XML	202
Adding static blocks to pages through widgets	205
Using a dynamic serving theme based on the client browser	209
Creating theme-specific translations	213
<u>Chapter 7: Creating Magento 2 Extensions – the Basics</u>	<u>217</u>
Introduction	217
Initializing extension basics	218
Working with database models	222
Creating tables using setup scripts	224
Creating a web route and controller to display data	235
Creating system configuration fields	241
Creating a backend data grid	246
Creating a backend form to add/edit data	256

Chapter 8: Creating Magento 2 Extensions – Advanced	267
Introduction	267
Using dependency injection to pass classes to your own class	268
Modifying functions with the use of plugins – Interception	271
Creating your own XML module configuration file	275
Creating your own product type	282
Working with service layers/contracts	288
Creating a Magento CLI command option	308
Index	317

Preface

Magento is one of the most successful open source e-commerce platforms out there today. It is powerful enough to build small or enterprise businesses.

The first stable version of Magento was released in 2008. Since the beginning, lots of merchants have chosen Magento to be their platform with great success.

The current global ecosystem of Magento includes 240+ merchants, 300+ partners, and 4500+ certified developers, and this number is growing on a daily basis.

Over the last few years, e-commerce has changed a lot, so Magento announced a new platform that will support this. This new era of commerce is called Magento 2.

Magento 2 offers enhanced performance and scalability. The new platform is built from scratch to enable lots of new features. It supports both B2C and B2B businesses, and will serve the best omnichannel shopping experiences to their shoppers.

This book will provide you with the necessary insights to get a better understanding on what is needed to build such a powerful commerce platform.

The book is divided into several recipes, which show you which steps to take to complete a specific action. In each recipe, we have a section that explains how everything works.

It will cover installing a Magento 2 website, configuring your categories and products, performance tuning, creating a theme, developing a module, and much more.

At the end of this book, you will gain the knowledge to start building a success website.

What this book covers

Chapter 1, Installing Magento 2 on Apache and NGINX, is a totally different ballgame compared to Magento 1. Where Magento 1 could be installed through FTP or SSH, Magento 2 is installable only via the command-line interface for an experienced webmaster.

Chapter 2, Magento 2 System Tools, explains how to install Magento 2 via the command shell. Magento released a new powerful tool to manage and install sample data, reindex your database, back up your site, or flush your caches, which are just a few of the options.

Chapter 3, Enabling Performance in Magento 2, explains how to configure different types of caching options. In Magento 2, the Full Page Cache (FPC) can be handled by Varnish to give your store a performance boost. There are also external services that you can use as a cache.

Chapter 4, Creating Catalogs and Categories, shows you one of the major elements of a Magento store before creating products. Creating the correct product type including attributes is an important step in setting up a Magento store.

Chapter 5, Managing Your Store, covers setting up the correct tax rules, configuring an inventory, and creating customer groups.

Chapter 6, Creating a Magento 2 Theme, discusses the Magento 2 blank theme and how to use the fallback to create seasonal variations. It also explains how the new theme is set up and where files are stored.

Chapter 7, Creating Magento 2 Extensions – the Basics, contains the basic functions required to use extensions in a Magento 2 installation. It contains a brief introduction to new methods introduced in the Magento 2 framework and examples on how to create basic functions.

Chapter 8, Creating Magento 2 Extensions – Advanced, explains how to use advanced features in extensions for Magento 2. It also includes how to add unit/functional tests as this is a new requirement for extensions listed on the new Magento Connect.

What you need for this book

The following setup is recommended for maximum enjoyment:

- ▶ Magento 2 source code
- ▶ A virtual Linux server (Ubuntu 15.10 or higher; DigitalOcean)
- ▶ Hypernode (<https://hypernode.com/>)
- ▶ CloudFlare (<https://www.cloudflare.com/>)
- ▶ IDE (PhpStorm, NetBeans, and Sublime)
- ▶ A Git account

Who this book is for

This book is intended primarily for intermediate to professional merchants, webmasters, DevOps, solution integrators, and PHP developers who are interested in Magento 2.

It will cover the basic and advanced features of Magento 2 DevOps, performance, theming, development, and system configuration.

Sections

In this book, you will find several headings that appear frequently (Getting ready, How to do it, How it works, There's more, and See also).

To give clear instructions on how to complete a recipe, we use these sections as follows:

Getting ready

This section tells you what to expect in the recipe, and describes how to set up any software or any preliminary settings required for the recipe.

How to do it...

This section contains the steps required to follow the recipe.

How it works...

This section usually consists of a detailed explanation of what happened in the previous section.

There's more...

This section consists of additional information about the recipe in order to make the reader more knowledgeable about the recipe.

See also

This section provides helpful links to other useful information for the recipe.

Conventions

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

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows:
"Go to your favorite browser and search using your `yourdomain.com`."

A block of code is set as follows:

```
/usr/local/bin/php -f install.php -- \
--license_agreement_accepted "yes" \
--locale "en_US" \
--timezone "America/Los_Angeles" \
--default_currency "USD" \
--db_host "mysql.example.com" \
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
docker run --rm --name magento2 -it -p 80:80 --link mysql:mysql -e
MYSQL_USER=root -e MYSQL_PASSWORD=admin -e PUBLIC_HOST=yourdomain.com
raybogman/mage2cookbook-docker $*
```

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

```
service apache2 status
netstat -anp | grep apache2
```

New terms and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "Check on the second like for the **Server API**; this should be **FPM/FastCGI**."



Warnings or important notes appear in a box like this.



Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail feedback@packtpub.com, and mention the book's title in the subject of your message.

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

Customer support

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

Downloading the example code

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

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box.
5. Select the book for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this book from.
7. Click on **Code Download**.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- ▶ WinRAR / 7-Zip for Windows
- ▶ Zipeg / iZip / UnRarX for Mac
- ▶ 7-Zip / PeaZip for Linux

Downloading the color images of this book

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from https://www.packtpub.com/sites/default/files/downloads/Magento2Cookbook_ColorImages.pdf.

Errata

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

To view the previously submitted errata, go to <https://www.packtpub.com/books/content/support> and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

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

Please contact us at copyright@packtpub.com with a link to the suspected pirated material. We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at questions@packtpub.com, and we will do our best to address the problem.

1

Installing Magento 2 on Apache and NGINX

In this chapter, we will cover the basic tasks related to installing Magento 2 on Apache and NGINX. You will learn the following recipes:

- ▶ Installing Apache
- ▶ Installing NGINX
- ▶ Installing PHP-FPM
- ▶ Installing HHVM
- ▶ Installing MySQL
- ▶ Installing Magento 2
- ▶ Installing Magento 2 on Hypernode
- ▶ Managing Magento 2 on Docker

Introduction

This chapter explains how to install Magento 2 on a hosting environment. When installing a new Magento 2 instance, we can use either a **Linux, Apache, MySQL, PHP (LAMP)** or **Linux, NGINX, MySQL, PHP (LEMP)** setup. Currently, options such as MariaDB or HHVM are equivalent to MySQL or PHP. Only HHVM will be converted in this chapter.

We will install a clean Magento 2 setup on a hosted **virtual private server (VPS)** for more advanced users and an easy-to-use installation on **Hypernode**.

The recipes in this chapter will primarily focus on a basic setup of how to install Magento 2. However, in some situations, we will dive in deeper related to the subject.

While Magento requirements differ from Magento 1, we will be using the latest and finest version from PHP, HHVM, NGINX, Apache, Redis, MySQL, and Ubuntu.

[ Creating a new Magento 2 stack could bring up minor issues. Always update to the latest available version, if possible.]

Installing Apache

Throughout the following recipes, we will install the latest Apache 2.4.x version on a **Software as a Service (SaaS)** platform hosted by **DigitalOcean**. The current Apache version supports HTTP/2. This recipe will show you how to do this.

[ HTTP/2 is an optimized version of the **Hypertext Transfer Protocol (HTTP)**, also known as **HTTP version 2 (HTTP/2)**. Some of the new features of HTTP/2 are a more efficient usage of network resources and network latency. HTTP/2 allows multiple concurrent connections over a single **Transmission Control Protocol (TCP)** connection at once, which optimizes TCP load. Merging CSS and JS is not necessary anymore. HTTP/2 service also provides the possibility to use server push, which allows a proactive push of resources to the client.]

Getting ready

For this recipe, you will need to create an account at DigitalOcean <https://www.digitalocean.com/>. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to create an Apache hosting environment. The following steps will guide you through this:

1. The first step is creating an account at <https://cloud.digitalocean.com/registrations>. All the steps to create an account are straightforward. Confirm your mail and update your billing profile.
2. Start creating your first Droplet. Choose your hostname, and select your size (2GB CPU's or 40GB SSD Disk). Next, pick your region (DigitalOcean has many regions available worldwide). Select your image (we will use the latest Ubuntu). Select extra available settings such as **Private Networking** to run multiple servers including an Internet network or backup and IPv6 options. If you already have an SSH key, you can upload this:

Create Droplet

Droplet Hostname

mage2cookbook.com

Select Size

\$5/mo \$0.007/hour 512 MB / 1 CPU 20 GB SSD Disk 1000 GB Transfer	\$10/mo \$0.015/hour 1 GB / 1 CPU 30 GB SSD Disk 2 TB Transfer	\$20/mo \$0.030/hour 2 GB / 2 CPUs 40 GB SSD Disk 3 TB Transfer	\$40/mo \$0.060/hour 4 GB / 2 CPUs 60 GB SSD Disk 4 TB Transfer	\$80/mo \$0.119/hour 8 GB / 4 CPUs 80 GB SSD Disk 5 TB Transfer
\$160/mo \$0.238/hour 16 GB / 8 CPUs 160 GB SSD Disk 6 TB Transfer	\$320/mo \$0.476/hour 32 GB / 12 CPUs 320 GB SSD Disk 7 TB Transfer	\$480/mo \$0.714/hour 48 GB / 16 CPUs 480 GB SSD Disk 8 TB Transfer	\$640/mo \$0.952/hour 64 GB / 20 CPUs 640 GB SSD Disk 9 TB Transfer	

Your Droplet
Hostname
mage2cookbook....
Size
\$20/mo
Region
New York 3
Image
15.10 x64
Settings
none
SSH Keys
none

- The Droplet takes approximately 60 seconds to get created. You will get an e-mail right after including all the additional login details such as the hostname, IP address, username, and password.
- Configure your domain name, `yourdomain.com`, to the addressed IP address from the Droplet:

`yourdomain.com A 123.123.123.123`

`www CNAME yourdomain.com`

- Connect your Droplet with SSH using your favorite SSH client (putty or terminal) and change your password:

`sudo ssh yourdomain.com`

- First, we will update your server to the latest updates available:

`apt-get update && apt-get -y upgrade`

7. Next, we will install Apache 2 using a third-party package. Currently, Ubuntu doesn't include the latest Apache 2.4.x with HTTP/2 support. The third-party vendor that we will use is <https://launchpad.net/~ondrej/+archive/ubuntu/apache2>. Run the following command:

```
echo "deb http://ppa.launchpad.net/ondrej/apache2/ubuntu wily main" | sudo tee -a /etc/apt/sources.list.d/apache2.list  
echo "deb-src http://ppa.launchpad.net/ondrej/apache2/ubuntu wily main" | sudo tee -a /etc/apt/sources.list.d/apache2.list
```

8. Before we can install Apache 2, we need to authorize the package by installing a signed key:

```
apt-key adv --keyserver keyserver.ubuntu.com --recv-keys  
0x4F4EA0AAE5267A6C
```

9. Now, we will update our Ubuntu system using the latest Apache2 packages:

```
apt-get update && apt-get -y install apache2
```

10. Once every Apache 2 package is installed, we can check whether everything is in order by running the following command on the shell:

```
apache2 -v
```

The output of this command is as follows:

```
root@mage2cookbook:~# apache2 -v  
Server version: Apache/2.4.17 (Ubuntu)
```

If you have followed steps 1 to 9, you will be able to see if the web server is running. Go to your favorite browser and search using yourdomain.com.



In the DigitalOcean control panel, you can create multiple snapshots at all times during the recipes. This will help you in restoring an old version or just going back in time.

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 3, we create a clean hosting server setup using DigitalOcean. Next, we connect the Droplet IP to your domain name in DNS. After login via SSH, we are able to update the server and continue to the process of installing Apache 2 via a third-party repository. Depending on the Ubuntu setup, we need to change the version name, such as `precise`, `trusty`, `wily`, `vivid`, or `xenial`, to install the software.

In step 8, we submit a key that will validate the repository before we can start installing the Apache 2 software. In step 9, we use a simple command to update the repository and start installing.

There's more...

If you want to check whether Apache is running fine, use one of the following commands:

```
service apache2 status
netstat -anp | grep apache2
```

Installing NGINX

Throughout the following recipes, we will install the latest NGINX 1.9.x version on a SaaS platform hosted by DigitalOcean. The current NGINX version supports HTTP/2. This recipe will show you how to do it.

Getting ready

For this recipe, you will need to create an account at DigitalOcean <https://www.digitalocean.com/>. No other prerequisites are required.



For this cookbook, we will use a commercial SaaS hosting provider to get a production-ready environment. You can use any other solution out there to build on your preferred stack.

How to do it...

For the purpose of this recipe, let's assume that we need to create an NGINX hosting environment. The following steps will guide you through this:

1. Follow steps 1 until 6 in the previous recipe on how to install Apache.
2. Then, we will update our server to the latest updates available:

```
apt-get update && apt-get -y upgrade
```

3. Next, we will install NGINX using the latest mainline version 1.9.x. Currently, Ubuntu doesn't include the latest NGINX 1.9.x mainline version. The latest change log can be viewed at <http://nginx.org/en/CHANGES>. Run the following command:

```
echo "deb http://nginx.org/packages/mainline/ubuntu/ wily nginx" |
sudo tee -a /etc/apt/sources.list.d/nginx.list
echo "deb-src http://nginx.org/packages/mainline/ubuntu/ wily
nginx" | sudo tee -a /etc/apt/sources.list.d/nginx.list
```

4. Before we can install NGINX, we need to authorize the package by installing a signed key:

```
wget http://nginx.org/keys/nginx_signing.key | apt-key add nginx_signing.key
```

5. Now, we will update our Ubuntu system using the latest NGINX packages:

```
apt-get update && apt-get -y install nginx
```

6. Once every NGINX package is installed, we can check whether everything is in order by running the following command on the shell:

```
nginx -v
```

The output of this command is as follows:

```
root@mage2cookbook:~# nginx -v
nginx version: nginx/1.9.6
```

If you have followed steps 1 to 6, you will be able to see if the web server is running. Go to your favorite browser and search using `yourdomain.com`.



In the DigitalOcean control panel, you can create multiple snapshots at all times during the recipes. This will help you in restoring an old version or just going back in time, such as switching from Apache to NGINX or back.

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 5, we used the same Droplet to install NGINX. All steps are alike, but instead of installing Apache, we use NGINX instead. The only big difference is that it is an official NGINX repository.

There's more...

The current market share of NGINX is around 15% worldwide compared to 50% of Apache on active sites. Over the last couple of years, NGINX has grown and is commonly used as a stable web server for Magento hosting:

<http://news.netcraft.com/archives/2015/03/19/march-2015-web-server-survey.html>

If you want to check whether NGINX is running fine, use one of the following commands:

```
service nginx status
netstat -anp | grep nginx
```

Installing PHP-FPM

Throughout the following recipes, we will install the latest PHP 7.x version on Apache and NGINX. The current **PHP-FastCGI Process Manager (PHP-FPM)** will be installed through a third-party package. This recipe will show you how to do it.

 PHP 7 is the latest version of the PHP stack. It's also known as **PHP Next Generation (PHP NG)** and PHP's answer to HHVM. Currently, it's two to three times faster than PHP 5.x.

The whole code base is rewritten and uses less memory. Lots of PHP functions are backward-compatible with the PHP 5.x code base, so upgrading to PHP 7 should be easy.

Getting ready

For this recipe, you will need a preinstalled Apache or NGINX setup. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to create an PHP-FPM hosting environment. The following steps will guide you through this:

1. First, we will start installing PHP-FPM on Apache; later, we will do the same with NGINX.
2. Next, we will install PHP 7 on Apache 2 using a third-party package. Currently, Ubuntu doesn't include the latest PHP 7 yet. The third-party vendor that we will use is <https://launchpad.net/~ondrej/+archive/ubuntu/php>.

Run the following command:

```
echo "deb http://ppa.launchpad.net/ondrej/php/ubuntu wily main" |  
sudo tee -a /etc/apt/sources.list.d/php.list  
  
echo "deb-src http://ppa.launchpad.net/ondrej/php/ubuntu wily  
main" | sudo tee -a /etc/apt/sources.list.d/php.list
```

3. As we have already used the authorized Apache package from the same vendor, we are okay on the signed key. There is no need to install the key anymore.
4. Now, we will update our Ubuntu system using the latest PHP 7 packages:

```
apt-get update && apt-get -y install php7.0 php7.0-fpm php7.0-dev  
php7.0-mhash php7.0-mcrypt php7.0-curl php7.0-cli php7.0-mysql  
php7.0-gd php7.0-intl php7.0-xsl
```

5. Once every PHP 7 package is installed, we can check whether everything is in order by running the following command on the shell:

```
php -v
```

The output of this command is as follows:

```
root@mage2cookbook:/etc/php# php -v
PHP 7.0.0 (cli) ( NTS )
Copyright (c) 1997-2015 The PHP Group
Zend Engine v3.0.0-dev, Copyright (c) 1998-2015 Zend Technologies
```

6. Now, we will configure PHP-FPM with Apache. Out of the box, Apache 2 comes with a module named **mod_proxy**, which we will use to connect Apache to PHP-FPM. We will be using PHP-FPM in TCP mode instead of the Unix socket. The main reason is that it's easy to configure and we can use it later with HHVM. The default port 9000 will be our connector. To enable mod_proxy, run the following command on the shell:

```
a2enmod proxy_fcgi
```

7. Before we can connect Apache to PHP-FPM, we need to give PHP-FPM instructions to listen to the correct internal port. Run the following command from the shell:

```
sed -i "s/listen =.*/listen = 127.0.0.1:9000/" /etc/php/7.0/fpm/pool.d/www.conf
```

You can do this manually as well; edit the `www.conf` file with your favorite editor in your PHP-FPM pool directory. Search for `listen = /var/run/php/php7.0-fpm.sock` and change this to `listen = 127.0.0.1:9000`.

8. Restart your PHP-FPM process to use the altered changes. Run the following command from the shell:

```
service php7.0-fpm restart
```

9. Next, we need to configure Apache using the currently set-up internal **proxy_fcgi** module. The default Apache Virtual Host configuration file is located at `/etc/apache2/sites-enabled/000-default.conf`. Edit this file and add the following code just before the closing `</VirtualHost>` tag on one line without any breaks:

```
ProxyPassMatch ^/(.*\.\php(.*))$ fcgi://127.0.0.1:9000/var/www/html/$1
```

The following code is for the new 000-default.conf:

```
<VirtualHost *:80>
ServerAdmin webmaster@localhost
DocumentRoot /var/www/html
<Directory /var/www/html>
Options Indexes FollowSymLinks
AllowOverride All
Order allow,deny
allow from all
</Directory>

ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined

ProxyPassMatch ^/(.*\.php(/.*)?)$ fcgi://127.0.0.1:9000/var/www/
html/$1
</VirtualHost>
```

You can change DocumentRoot /var/www/html to any given preferred directory. However, for the rest of the recipes, we will stay with the default.

10. After saving the Apache configuration, we need to restart the web server. Run the following command from the shell:

```
service apache2 restart
```

11. Now, we need to check whether Apache and PHP-FPM are working fine. Go to the cd /var/www/html directory and create the following file including content:

```
echo "<?php phpinfo(); ?>" > phpinfo.php
```

12. If you have followed steps 1 to 11, you will be able to see if PHP-FPM works with your web server. Go to your favorite browser and search using yourdomain.com/phpinfo.php.

You should now see a phpinfo page like the following screenshot. Check on the second line for the **Server API**; this should be **FPM/FastCGI**.

The screenshot shows a web browser window with the title "phpinfo()". The address bar contains "mage2cookbook.com/phpinfo.php". The main content area is titled "PHP Version 7.0.0" and features a large "php" logo. Below the title, there is a table with numerous rows of PHP configuration details. One key entry in the table is "Server API" which is listed as "FPM/FastCGI". Other visible entries include "System" (Linux mage2cookbook.com 4.2.0-16-generic #19-Ubuntu SMP Thu Oct 8 15:35:06 UTC 2015 x86_64), "Virtual Directory Support" (disabled), "Configuration File (php.ini) Path" (/etc/php/7.0/fpm), and "PHP API" (20151012).

PHP Version 7.0.0	
System	Linux mage2cookbook.com 4.2.0-16-generic #19-Ubuntu SMP Thu Oct 8 15:35:06 UTC 2015 x86_64
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.0/fpm
Loaded Configuration File	/etc/php/7.0/fpm/php.ini
Scan this dir for additional .ini files	/etc/php/7.0/fpm/conf.d
Additional .ini files parsed	(none)
PHP API	20151012
PHP Extension	20151012
Zend Extension	320151012
Zend Extension Build	API320151012.NTS
PHP Extension Build	API20151012.NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	enabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, phar, zip
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	zlib.*, convert.iconv.* , mcrypt.* , mdecrypt.* , string.rot13 , string.toupper , string.tolower , string.strip_tags , convert.* , compress.zlib

If everything is working fine, we have completed the Apache PHP-FPM setup. Now let's do the same for the NGINX and PHP-FPM setup.

As we have already installed the PHP-FPM packages in steps 1 to 5, we now need to combine them with NGINX.

Before we continue, it is important to use either a clean DigitalOcean Droplet with NGINX running or you can disable the Apache server using the following command from the shell:

```
service apache2 stop && service nginx start
```

-
13. Next, we need to configure NGINX. The default NGINX configuration file is located at `/etc/nginx/conf.d/default.conf`. Remove the old configuration in this file and add the following code:

```
server {  
    listen      80;  
    server_name yourdomain.com;  
  
    root   /var/www/html;  
    index  index.php index.html;  
  
    access_log /var/log/nginx/access.log;  
    error_log /var/log/nginx/error.log;  
  
    location ~ \.php$ {  
  
        fastcgi_index index.php;  
        fastcgi_pass 127.0.0.1:9000;  
        fastcgi_param SCRIPT_FILENAME  
            $document_root$fastcgi_script_name;  
        include       fastcgi_params;  
    }  
  
    location ~ /\.ht {  
        deny  all;  
    }  
}
```

14. Save the new configuration using your favorite editor and restart your NGINX and PHP-FPM server using the following command on the shell:

```
service nginx restart && service php7.0-fpm restart
```

A nice trick to test if your configuration is correct is as follows:

```
nginx -t
```

If you have followed steps 13 to 14, you will be able to see if PHP-FPM works with your web server. Go to your favorite browser and search using your `yourdomain.com/phpinfo.php`.

You should now see a `phpinfo` page similar to the one on the previous page during the Apache PHP-FPM setup. Check on the second line for the **Server API**; this should be **FPM/FastCGI**.

Downloading the example code

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

You can download the code files by following these steps:



- ▶ Log in or register to our website using your e-mail address and password.
- ▶ Hover the mouse pointer on the **SUPPORT** tab at the top.
- ▶ Click on **Code Downloads & Errata**.
- ▶ Enter the name of the book in the **Search** box.
- ▶ Select the book for which you're looking to download the code files.
- ▶ Choose from the drop-down menu where you purchased this book from.
- ▶ Click on **Code Download**.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- ▶ WinRAR / 7-Zip for Windows
- ▶ Zipeg / iZip / UnRarX for Mac
- ▶ 7-Zip / PeaZip for Linux

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 4, we first install PHP-FPM from a third-party repository. The main reason that we use this repository is that it is easy to use and well-supported. Current Ubuntu versions don't have the latest PHP 7 available yet. If you are comfortable installing PHP from source, you can do so.

All PHP modules listed in step 4 are mandatory for Magento 2 to work properly. In step 6, we explain why we use PHP-FPM running on port 9000. Setting up a TCP port is much easier when using multiple PHP backends, switching from Apache to NGINX, or PHP-FPM versus HHVM.

In step 7, we add an Apache module that acts as a proxy, so we are able to connect PHP-FPM to Apache.

In step 8, we changed the PHP-FPM pool to switch from Unix sockets to TCP. If you prefer sockets, you can do so.

In step 10, we add the ProxyPassMatch rule to our Apache configuration file, which will serve all incoming PHP requests to the PHP-FPM server. After saving and restarting the Apache server, we are able to test if everything works.

In step 13, we configure NGINX to work with PHP-PFM. Creating `fastcgi_pass` does the trick to connect to port 9000.

There's more...

If you want to check whether PHP-FPM is running fine, use one of the following commands:

```
service php7.0-fpm status
netstat -anp | grep php-fpm
```

To check which server is running, check the headers by running the following command:

```
curl -I http://mage2cookbook.com/phpinfo.php
```

The output of this command will be as follows:

```
root@mage2cookbook:~# curl -I http://mage2cookbook.com/phpinfo.php
HTTP/1.1 200 OK
Server: nginx/1.9.6
```

Once we switch back to Apache we will see the following:

```
service nginx stop && service apache2 start
root@mage2cookbook:~# curl -I http://mage2cookbook.com/phpinfo.php
HTTP/1.1 200 OK
Server: Apache/2.4.17 (Ubuntu)
```



Use `phpinfo.php` wisely on a production environment. Sharing this information on a production website is not advised and could expose your security risks.

Installing HHVM

Throughout the following recipes, we will install the latest HHVM 3.9.x version on Apache and NGINX. HHVM has gained large popularity over the last couple of year and is well-known for its high performance on Magento. While Magento 1 was not supported by default, Magento 2 is. This recipe will show you how to do it.



HHVM (also known as the **HipHop Virtual Machine**) is a virtual machine for PHP developed by Facebook, with an associated **just-in-time (JIT)** compiler. Deploying HHVM on a Magento 2 website should lead to performance improvements across your web shop.

Getting ready

For this recipe, you will need a preinstalled Apache or NGINX setup. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to create an HHVM hosting environment. The following steps will guide you through this:

1. First, we will start installing HHVM on Apache; later, we will do the same with NGINX.
2. Next, we will install HHVM on Apache2 using the official prebuilt package by Facebook. Run the following command on the shell:

```
echo "deb http://dl.hhvm.com/ubuntu wily main" | sudo tee -a /etc/apt/sources.list.d/hhvm.list
```

3. Before we can install HHVM, we need to authorize the package by installing a signed key:

```
apt-key adv --keyserver keyserver.ubuntu.com --recv-keys  
0x5a16e7281be7a449
```

4. Now we will update our Ubuntu system using the latest HHVM packages:

```
apt-get update && apt-get -y install hhvm
```

5. Once every HHVM package is installed, we can check whether everything is in order by running the following command on the shell:

```
hhvm --version
```

The output of this command is as follows:

```
root@mage2cookbook:~# hhvm --version  
HipHop VM 3.10.1 (rel)
```

6. Since the setup of PHP-FPM, we have started using the internal port 9000 to handle all traffic on PHP. Now we will use the same port for HHVM. Stop PHP-FPM as it is still running and start HHVM using the following command from the shell:

```
service php7.0-fpm stop && service hhvm start
```

7. If you need to run dedicated HipHop code such as Hack (the default HipHop programming language) in the future, you may want to change your Apache configuration. Restart your Apache and HHVM server:

```
ProxyPassMatch ^/(.+\.hh|php)(/.*)? $ fcgi://127.0.0.1:9000/var/www/html/$1
```

8. Now, let's check whether HHVM and Apache are working. Create a PHP file called hhvm.php in the /var/www/html directory:

```
<?php  
if (defined('HHVM_VERSION')) {  
    echo "HHVM";  
} else {  
    echo "PHP";  
}
```

If you have followed steps 1 to 8, you will be able to see if HHVM works with your web server. Go to your favorite browser and search using your `yourdomain.com/hhvm.php`. You should now see a **HHVM** notice on your screen.

In newer versions of HHVM, the default `phpinfo()` ; tag works just fine on your screen. Go to your favorite browser and search using your `yourdomain.com/phpinfo.php`:

The screenshot shows a web browser window titled "HHVM phphinfo". The address bar contains "mage2cookbook.com/phpinfo.php". The main content area displays the title "HHVM Version 3.10.1" and a table with the following data:

Version	
Version	3.10.1
Version ID	31001
Debug	
Compiler ID	tags/HHVM-3.10.1-0-g689b4969a141620ee5a282ce0dbf72278c84d44b
Repo Schema	6c99ee1f98340f6f3ef397a332583f0e843a627d
PHP Version	5.6.99-hhvm
Zend Version	2.4.99
uname	Linux mage2cookbook.com 3.19.0-31-generic #36-Ubuntu SMP Wed Oct 7 15:04:02 UTC 2015 x86_64

Below this, there is a section titled "INI" containing the following table:

allow_url_fopen	1
...	...

9. If everything is working fine, we have completed the Apache HHVM setup. Now let's do the same for the NGINX and HHVM setup.

As we have already installed the HHVM packages in steps 1 to 5, we now need to combine them with NGINX.

10. Before we continue, it is important to use either a clean DigitalOcean Droplet with NGINX running or you can disable the Apache server using the following command from the shell:

```
service apache2 stop && service nginx start
```

The following is the content of NGINX configuration file:

```
server {  
    listen      80;  
    server_name yourdomain.com;  
  
    root   /var/www/html;  
    index  index.php index.html;  
  
    access_log /var/log/nginx/access.log;  
    error_log /var/log/nginx/error.log;  
  
    location ~ \.(hh|php)$ {  
  
        fastcgi_index index.php;  
        fastcgi_pass 127.0.0.1:9000;  
        fastcgi_param SCRIPT_FILENAME  
            $document_root$fastcgi_script_name;  
        include       fastcgi_params;  
    }  
  
    location ~ /\.ht {  
        deny  all;  
    }  
}
```

In the preceding example, we altered the location for PHP and Hack support:

```
location ~ \.(hh|php)$ {
```

11. You can now restart your NGINX and HHVM server to connect them using the follow command from the shell:

```
service nginx restart && service hhvm restart
```

-
12. Now let's check whether HHVM and NGINX are working. Create a PHP file called `hhvm.php` or use the one in the previous Apache HHVM setup in the `/var/www/html` directory:

```
<?php  
if (defined('HHVM_VERSION')) {  
    echo "HHVM";  
} else {  
    echo "PHP";  
}
```

If you have followed steps 9 to 11, you will be able to see if HHVM works with your web server. Go to your favorite browser and search using yourdomain.com/hhvm.php.

You should now see a **HHVM** notice on your screen, but you can also use yourdomain.com/phpinfo.php.

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 4, we use the official HHVM repository and install the software. The installation process is similar to the PHP-FPM setup. The important difference between HHVM and PHP-FPM is that HHVM does not have additional modules to install.

In step 7, we change `ProxyPassMatch` a little bit, and add the `.hh` extension parameter only. By default, the `.hh` extension is used only when creating HipHop (Hack)-dedicated code.

In step 10, we do the same procedure for NGINX. The only thing that we change is the `.hh` extension in the location. As HHVM runs by default on port 9000, we do not change anything here.

There's more...

If you want to check whether HHVM is running fine, use one of the following commands:

```
service hhvm status  
netstat -anp | grep hhvm
```

To check which server is running, check the headers by running the following command:

```
curl -I http://mage2cookbook.com/hhvm.php
```

The output of this command is as follows:

```
root@mage2cookbook:~# curl -I http://mage2cookbook.com/hhvm.php
HTTP/1.1 200 OK
Server: nginx/1.9.6
X-Powered-By: HHVM/3.10.1
```

Once we switch back to Apache, we will see the following:

```
service nginx stop && service apache2 start
root@mage2cookbook:~# curl -I http://mage2cookbook.com/hhvm.php
HTTP/1.1 200 OK
Server: Apache/2.4.17 (Ubuntu)
X-Powered-By: HHVM/3.10.1
```

Installing MySQL

Throughout the following recipes, we will install the latest MySQL 5.7.x version on Apache and NGINX. The current MySQL version is a milestone in the open source world. Its new performance and scalability features will be a great benefit for every Magento website. This recipe will show you how to do it.



MySQL 5.7 is an extremely exciting new version of the world's most popular open source database, which is two times faster than MySQL 5.6 while also improving usability, manageability, and security.

Getting ready

For this recipe, you will need a preinstalled Apache or NGINX and PHP-FPM or HHVM setup. No other prerequisites are required.

How to do it...

Before we can install the latest MySQL version, we need to download the software in our local system. The official MySQL APT repository provides you with a simple and convenient way to install and update MySQL products. Always use the latest software package using apt-get.

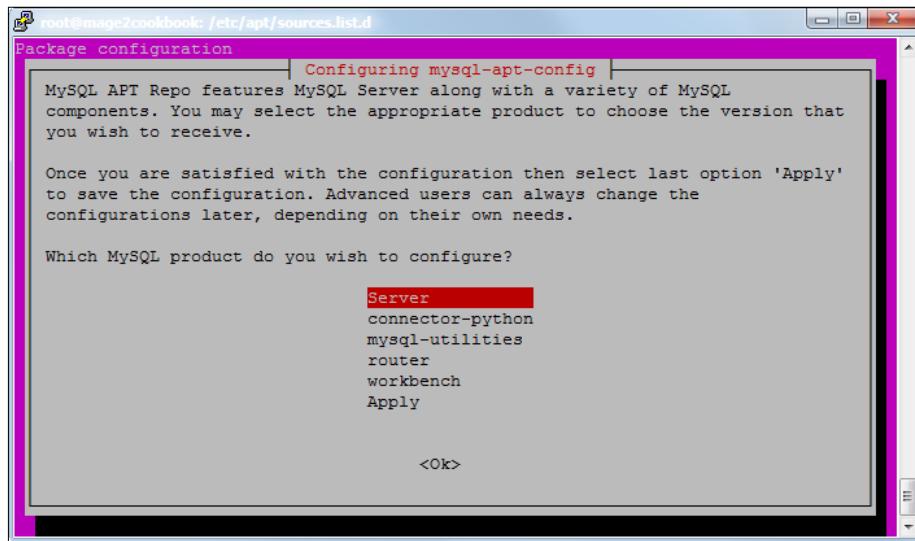
1. We will now install MySQL using the official vendor package. Run the following command from your root or home directory:

```
wget http://dev.mysql.com/get/mysql-apt-config_0.5.3-1_all.deb
```

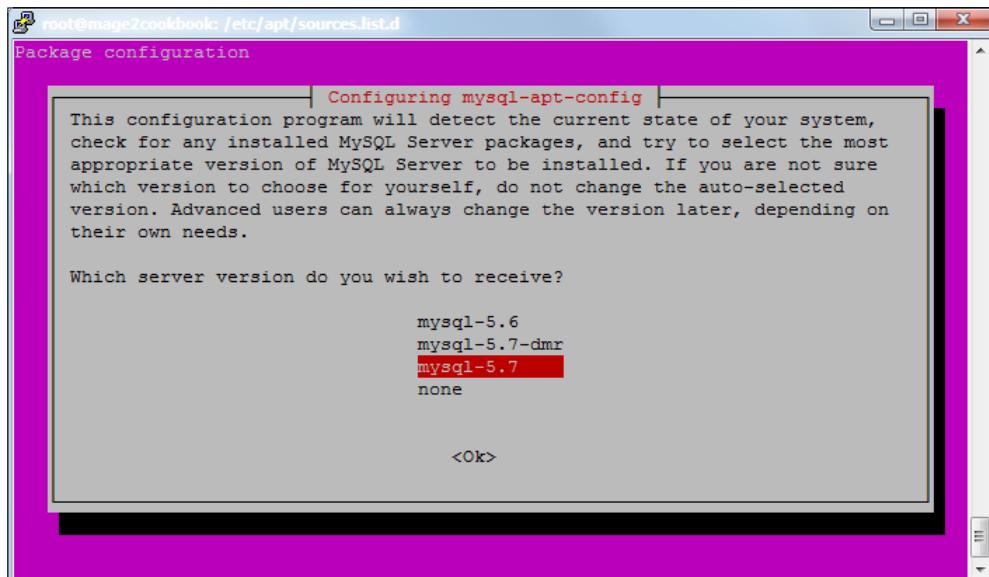
2. To download the rest of the latest MySQL package, we first install the `mysql-apt-config` package. Run the following command:

```
dpkg -i mysql-apt-config_0.5.3-1_all.deb
```

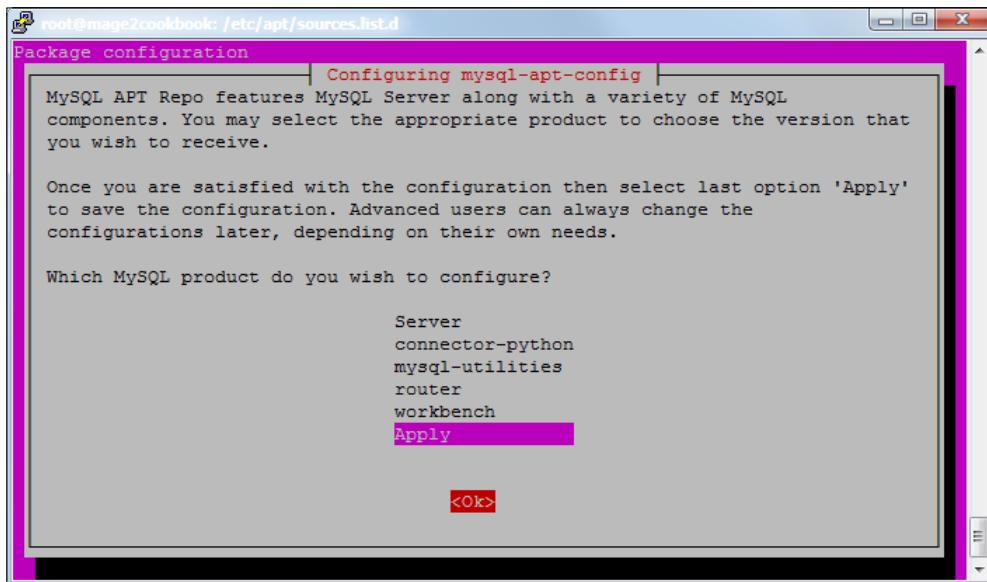
During installation, it will ask which MySQL product you wish to configure. Select **Server**:



Next, it will ask which Server version you wish to receive. Select **mysql-5.7**:



In the next screen, click on **Apply**:



- Now, we will update our Ubuntu system using the latest MySQL packages:

```
apt-get update && apt-get -y install mysql-server
```

- During installation, it will ask you several questions. The first one will be to choose a new password for the MySQL **root** user. Always make sure to create a new dedicated user only for a database; using **root** is just an example and not advised on production.
- Once every MySQL package is installed, we can check whether everything is in order by running the following command on the shell:

```
mysql --version
```

The output of this command is as follows:

```
root@mage2cookbook:~# mysql --version
mysql Ver 14.14 Distrib 5.7.9, for Linux (x86_64)
```

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 3, we download the official MySQL package and use a graphical interface to install the latest version. The whole process is pretty straightforward. Remember to use a dedicated user that has all the privileges instead of the root user.

There's more...

If you want to check whether MySQL is running fine, use one of the following commands:

```
service mysql status  
netstat -anp | grep mysql
```

After the MySQL server installation is finished, you can log in on the shell using the following command:

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

Installing Magento 2

Throughout the following recipes, we will install Magento 2 on a preconfigured SaaS platform hosted by DigitalOcean. Installing Magento 2 differs a lot from the current Magento 1 version. An important change is the use of **Composer**. We will use Composer to install all the third-party modules and manage the core of Magento 2. This recipe will show you how to do it.



Composer is a dependency management tool for PHP. It allows you to declare the software libraries for your project. It will help you install or update them.

Getting ready

For this recipe, you will need a preinstalled Apache or NGINX, PHP or HHVM and MySQL setup. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to create a Magento 2 hosting environment. The following steps will guide you through this:

1. First, we will start installing Magento 2 on Apache; later, we will do the same with NGINX. Throughout this recipe, you can use PHP-FPM or HHVM. For now, we will focus only on PHP-FPM.
2. Installing Composer is pretty straightforward. For the rest of the recipes, we will be installing Composer in the `/usr/local/bin` directory. This way, we can use it system-wide. Run the following command on the shell in the `/usr/local/bin` directory:

```
curl -sS https://getcomposer.org/installer | php
```

If you have not installed **curl** yet, you may need to run the following command on the shell:

```
apt-get -y install curl
```

To check whether Composer is working fine, run the following command:

```
mv composer.phar composer  
./composer
```

You will get the following output:

```
root@mage2cookbook:/var/www/html#
root@mage2cookbook:/var/www/html# composer

[composer logo]

Composer version 1.0-dev (2f069bffd2e9c9acfef993cc2a6d08ec867b52e) 2015-11-14 16:19:45

Usage:
  command [options] [arguments]

Options:
  -h, --help           Display this help message
  -q, --quiet          Do not output any message
  -V, --version         Display this application version
  --ansi               Force ANSI output
  --no-ansi             Disable ANSI output
  -n, --no-interaction Do not ask any interactive question
  --profile             Display timing and memory usage information
  -d, --working-dir=WORKING-DIR If specified, use the given directory as working directory.
  -v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug

Available commands:
  about            Short information about Composer
  archive          Create an archive of this composer package
  browse           Opens the package's repository URL or homepage in your browser.
  clear-cache      Clears composer's internal package cache.
  clearcache       Clears composer's internal package cache.
  config           Set config options
  create-project   Create new project from a package into given directory.
  depends          Shows which packages depend on the given package
  diagnose         Diagnoses the system to identify common errors.
  dump-autoload   Dumps the autoloader
  dumpautoload    Dumps the autoloader
  global            Allows running commands in the global composer dir ($COMPOSER_HOME).
  help              Displays help for a command
  home              Opens the package's repository URL or homepage in your browser.
  info              Show information about packages
  init              Creates a basic composer.json file in current directory.
  install           Installs the project dependencies from the composer.lock file if present, or falls back on the composer.json.
  licenses          Show information about licenses of dependencies
  list              Lists commands
  remove            Removes a package from the require or require-dev
  require           Adds required packages to your composer.json and installs them
  run-script        Run the scripts defined in composer.json.
  search            Search for packages
  self-update       Updates composer.phar to the latest version.
  selfupdate        Updates composer.phar to the latest version.
  show              Show information about packages
  status            Show a list of locally modified packages
  suggests          Show package suggestions
  update            Updates your dependencies to the latest version according to composer.json, and updates the composer.lock file.
  validate          Validates a composer.json and composer.lock
root@mage2cookbook:/var/www/html#
```

While we will be using a search-friendly URL in Magento, we need to enable the Apache **mod_rewrite** module before we can continue. Run the following code on the shell:

a2enmod rewrite

Restart your Apache server to complete the new configuration. Run the following code on the shell:

```
service apache2 restart
```

If you see the following output on your screen, everything is correct:

```
root@mage2cookbook:/var/www/html# a2enmod rewrite
Enabling module rewrite.
To activate the new configuration, you need to run:
service apache2 restart
```

Installing Magento 2 can be done in several ways. First, we will be showing you how to install it using the latest version from the GitHub release branch. Download the complete package and unzip or untar the files in your /var/www/html directory.

The URL for the latest releases is <https://github.com/magento/magento2/releases>.



Make sure that your /var/www/html has no files in it anymore.
(Use the rm * command to clean up.)

Execute either of the following commands:

```
wget https://github.com/magento/magento2/
archive/2.0.0.zip
wget https://github.com/magento/magento2/
archive/2.0.0.tar.gz
```

While extracting your files, make sure to move or copy them in to the /var/www/html directory.

- Now let's set the ownership and permissions:

```
chown -R www-data:www-data /var/www/html
find . -type d -exec chmod 770 {} \; && find . -type f -exec chmod
660 {} \; && chmod u+x bin/magento
```

- Now we use Composer to resolve all of our dependencies. Run the following command on the shell:

```
cd /var/www/html/ && composer install
```

If your installation is set up correctly, the installation will continue, using the correct PHP modules. However, if, in some way, your PHP installing is not correct, you may get the following error messages. Update your system and try again:

Your requirements could not be resolved to an installable set of packages.



Problem 1

- The requested PHP extension ext-gd * is missing from your system.

Problem 2

- The requested PHP extension ext-curl * is missing from your system.

Problem 3

- The requested PHP extension ext-intl * is missing from your system.

- During the installation process, you will get a notice to create a GitHub OAuth token. The download rate limit is pretty small. Copy the URL from your shell window in your browser, log in at GitHub, or create an account and token:

New personal access token

Token description

Composer on mage2cookbook.com 2015-11-09 1333

What's this token for?

Select scopes

Scopes *limit* access for personal tokens. [Read more about OAuth scopes.](#)

<input checked="" type="checkbox"/> repo ⓘ	<input type="checkbox"/> repo:status ⓘ	<input type="checkbox"/> repo_deployment ⓘ
<input type="checkbox"/> public_repo ⓘ	<input type="checkbox"/> delete_repo ⓘ	<input type="checkbox"/> user ⓘ
<input type="checkbox"/> user:email ⓘ	<input type="checkbox"/> user:follow ⓘ	<input type="checkbox"/> admin:org ⓘ
<input type="checkbox"/> write:org ⓘ	<input type="checkbox"/> read:org ⓘ	<input type="checkbox"/> admin:public_key ⓘ
<input type="checkbox"/> write:public_key ⓘ	<input type="checkbox"/> read:public_key ⓘ	<input type="checkbox"/> admin:repo_hook ⓘ
<input type="checkbox"/> write:repo_hook ⓘ	<input type="checkbox"/> read:repo_hook ⓘ	<input type="checkbox"/> admin:org_hook ⓘ
<input type="checkbox"/> gist ⓘ	<input type="checkbox"/> notifications ⓘ	

Generate token

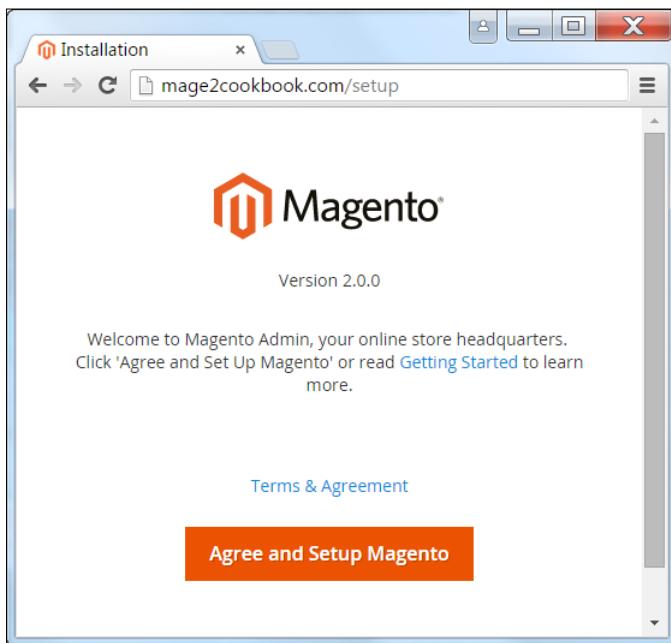
ⓘ Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API](#) over Basic Authentication.

Copy the generated token in the command prompt of your current shell window and the token will be saved for future reference in `/root/.composer/auth.json`. You can continue the rest of the installation.

6. Next, we will be using the setup wizard to continue the rest of the installation. In *Chapter 2, Magento 2 System Tools*, we will be using the shell installation method. Go to your favorite browser and run the following URL:

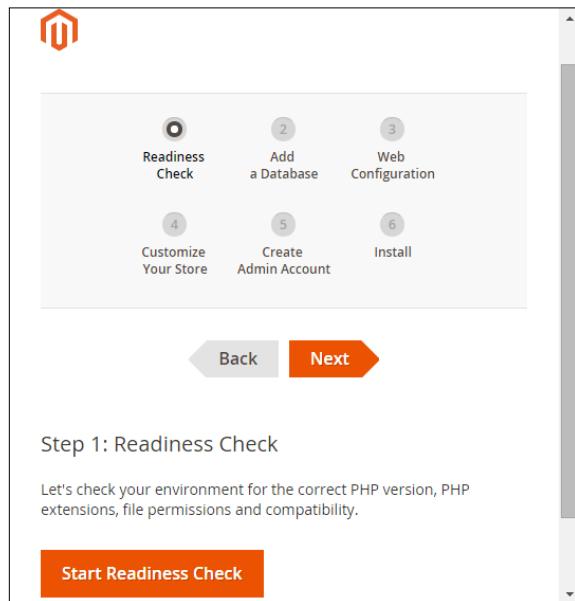
`http://mage2cookbook.com/setup`

As a result, we will get the following screen:

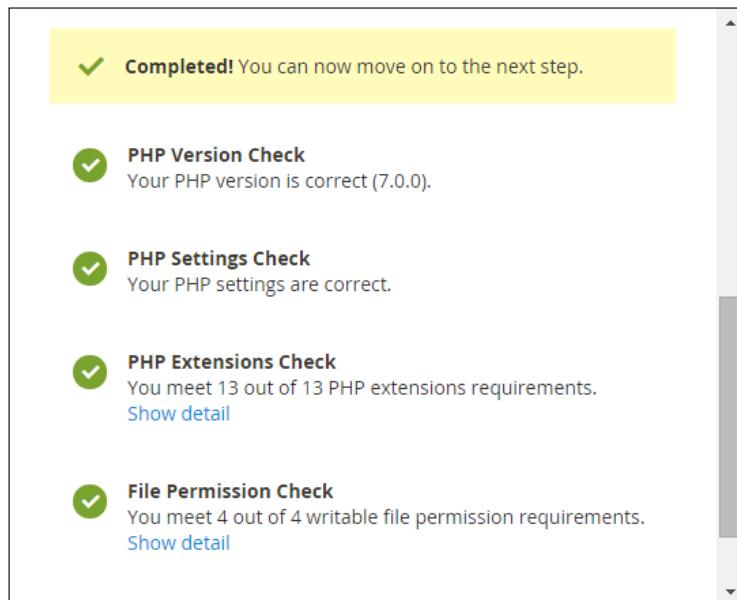


7. Now click on **Agree and Setup Magento**.

8. Let's start analyzing the readiness of your setup. Click on the **Next** button:



9. Magento 2 now will check your hosting setup for the correct PHP version, PHP modules, PHP settings, and file permissions, as shown in the following screenshot:



10. Continue the installation flow and commit your database credentials in the following screen:

The screenshot shows the 'Add a Database' step of the Magento setup process. At the top, there's a navigation bar with a back arrow, forward arrow, and a search icon. Below it, the URL is mage2cookbook.com/setup/#/add-database. The main interface has a header with the Magento logo and a progress bar showing six steps: 1. Readiness Check, 2. Add a Database (which is currently selected), 3. Web Configuration, 4. Customize Your Store, 5. Create Admin Account, and 6. Install. Below the progress bar are two buttons: 'Back' and 'Next'.

Step 2: Add a Database

Database Server Host *
localhost

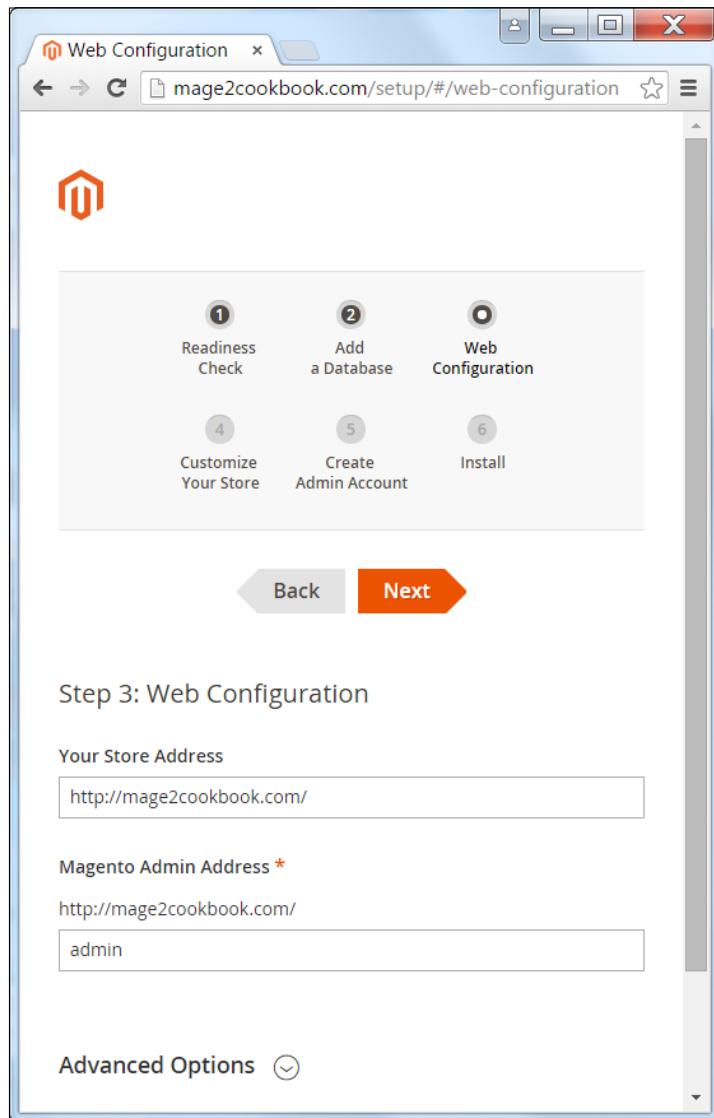
Database Server Username *
root

Database Server Password
(not always necessary)

Database Name *
magento

Table prefix
(optional)

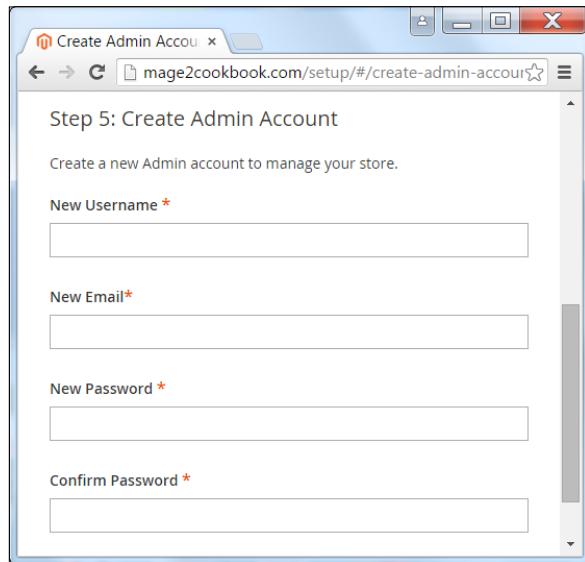
11. In **Step 3**, we will set up our store address and administrator address. In **Advanced Options**, we are able to pick HTTPS, Apache rewrites, encryption key, and where to store the session key:



Step 4 gives you the opportunity to choose your time zone, currency, and language. In Magento 2, there is a brand new option to pick which modules need to be installed. You can easily select them by just clicking the checkbox, as shown in the following screenshot:

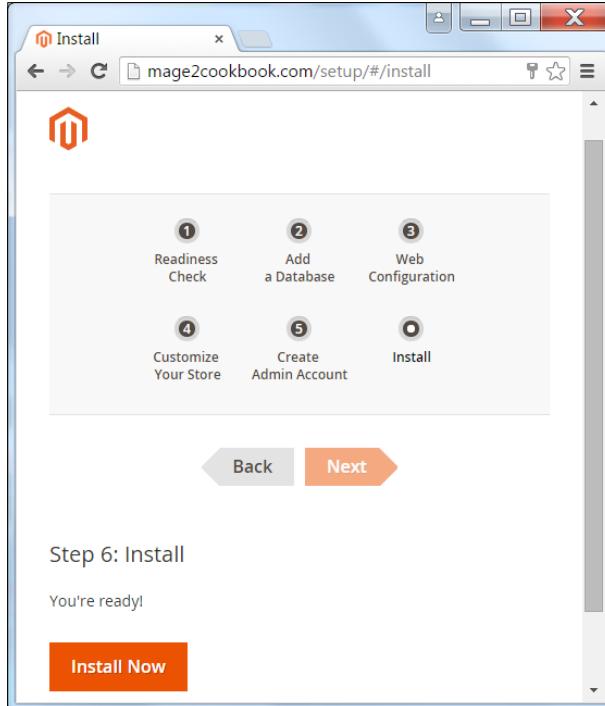
The screenshot shows the 'Customize Your Store' step of the Magento 2 setup process. At the top, there are six numbered steps: 1. Readiness Check, 2. Add a Database, 3. Web Configuration, 4. Customize Your Store (which is the current step), 5. Create Admin Account, and 6. Install. Below the steps are 'Back' and 'Next' buttons. The main content area is titled 'Step 4: Customize Your Store'. It contains three dropdown menus for 'Store Default Time Zone*', 'Store Default Currency*', and 'Store Default Language*'. The 'Time Zone' dropdown is set to 'GMT (UTC)'. The 'Currency' dropdown is set to 'US Dollar (USD)'. The 'Language' dropdown is set to 'English (United States)'. Below these fields is a section titled 'Advanced Modules Configurations' with a link. Under this section, there is a checked checkbox labeled 'Select All' and a list of three checked checkboxes: 'Magento_AdminNotification', 'Magento_AdvancedPricingImportExport', and 'Magento_Authorization'.

12. In **Step 5**, you can choose your username and password:



The screenshot shows a web browser window titled "Create Admin Accou" with the URL "mage2cookbook.com/setup/#/create-admin-accou". The page is titled "Step 5: Create Admin Account" and contains instructions: "Create a new Admin account to manage your store." It features four input fields with red asterisks indicating required fields: "New Username", "New Email", "New Password", and "Confirm Password".

13. Now we are almost there; the final **Step 6** will start installing our Magento 2 environment:

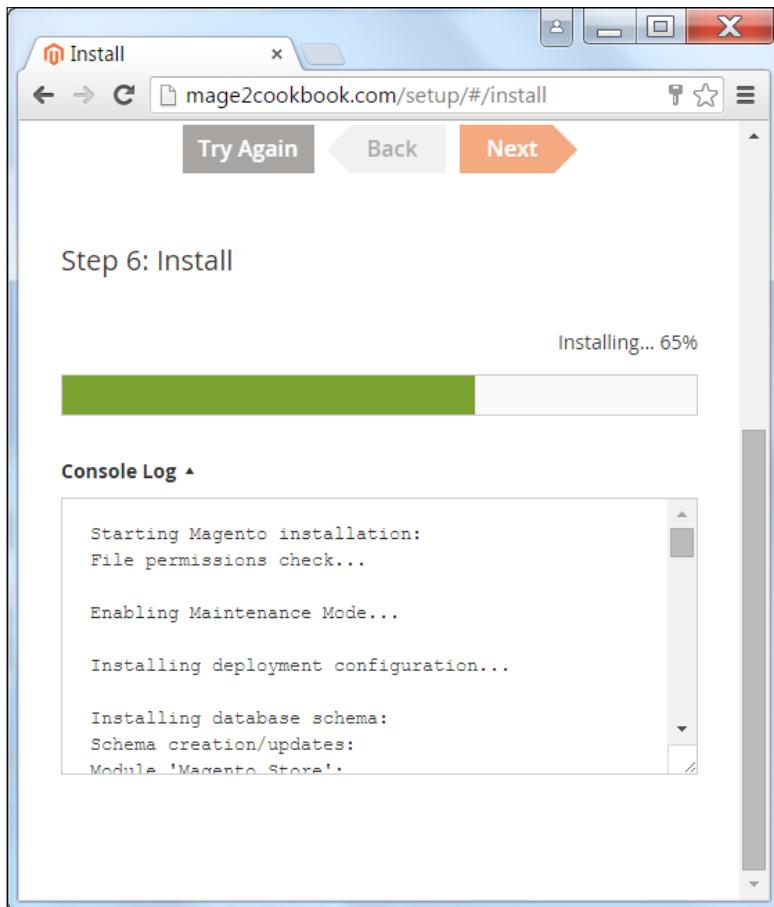


The screenshot shows a web browser window titled "Install" with the URL "mage2cookbook.com/setup/#/install". The page displays the final step of the installation process, titled "Step 6: Install". It shows five numbered steps: 1. Readiness Check, 2. Add a Database, 3. Web Configuration, 4. Customize Your Store, and 5. Create Admin Account. Step 5 is currently selected, indicated by a highlighted circle. Below the steps are "Back" and "Next" buttons. The "Next" button is orange. The text "You're ready!" is displayed above the "Install Now" button.

The installation step is really fast in Magento 2 and is done within a minute depending on your server. If you get an error in the console screen, such as a missing database, create one via the shell with the following command:

```
mysql -u root -p  
create database magento;
```

Installation of Magento 2 can be seen in the following screenshot:



14. Congratulations, you have successfully installed Magento 2. You will be given administrative information on your login and store address.

For security reasons, you may want to alter the write permissions of your /app/etc directory to only read permissions with the following command:

```
chmod -R 440 /var/www/html/app/etc
```

Now we will focus on the configuration of Magento 2 on NGINX. We don't need to reinstall Magento 2; we just need to alter the current NGINX setup with the appropriate setting.

15. Now, let's go to the NGINX configuration directory and update `default.conf` in `/etc/nginx/conf.d`. Open the `default.conf` file and change it with the following settings:

```
upstream fastcgi_backend {
    server 127.0.0.1:9000;
}

server {
    listen      80;
    server_name yourdomain.com;

    set $MAGE_ROOT /var/www/html;
    set $MAGE_MODE developer;

    include /var/www/html/nginx.conf.sample;

    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;

    location ~ /\.ht {
        deny all;
    }
}
```

As you can see, we are now using an `upstream` and `set $MAGE` setting and have removed the `fastcgi_pass`.

However, the most important element is the Magento 2 `nginx.conf.sample` file, which is in the root directory of your Magento 2 instance. For now, it is easy to include this directory but not advised on a production level. It's best that you copy this file to your NGINX configuration directory and store it there. Don't forget to change the following setting:

```
include /var/www/html/nginx.conf.sample;
```

16. Save your configuration and run the following command to check whether your settings are correct. If not, change your configuration:

```
nginx -t
```

17. If your syntax is okay and the test is successful, then restart your NGINX server. Don't forget to stop your Apache server. Run the following command on the shell:

```
service apache2 stop && service nginx start
```

To check whether you are running NGINX, use the following command:

```
curl -I http://mage2cookbook.com/  
root@mage2cookbook:~# curl -I http://mage2cookbook.com/HTTP/1.1  
200 OKServer: nginx/1.9.6
```

18. Congratulations, you just finished the installation of Magento 2 on NGINX.

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 and 2, we install Composer. Composer is our most important element before we can start installing Magento 2. As we are using Apache, we need to enable the mod_rewrite module, which is mandatory in Magento 2. Next, we download the latest Magento 2 version from GitHub and unzip the code in our directory.

In step 4, we check whether our setup is resolving all dependencies. Depending on the outcome, we need to install or enable the additional PHP modules.

In step 5, we create a GitHub token and store it in the auth.json file. We need this because the download rate limit is small when downloading additional software dependencies.

In step 6, we use the Magento 2 setup directory to install via the graphical user interface. All the remaining steps till 12 are self-explanatory.

In step 14, we switch from Apache to NGINX. The configuration file contains the following important elements: nginx.conf.sample, upstream fastcgi_backend, and \$MAGE to finish our setup. The upstream parameter makes sure that all PHP requests are connected to PHP-FPM, the \$MAGE variable connects the web directory, and include refers to the master configuration file.

There's more...

If you want to check which Apache modules are enabled, use the following command:

```
apachectl -M
```

The equivalent for NGINX to check your modules is nginx -v.

Installing Magento 2 on Hypernode

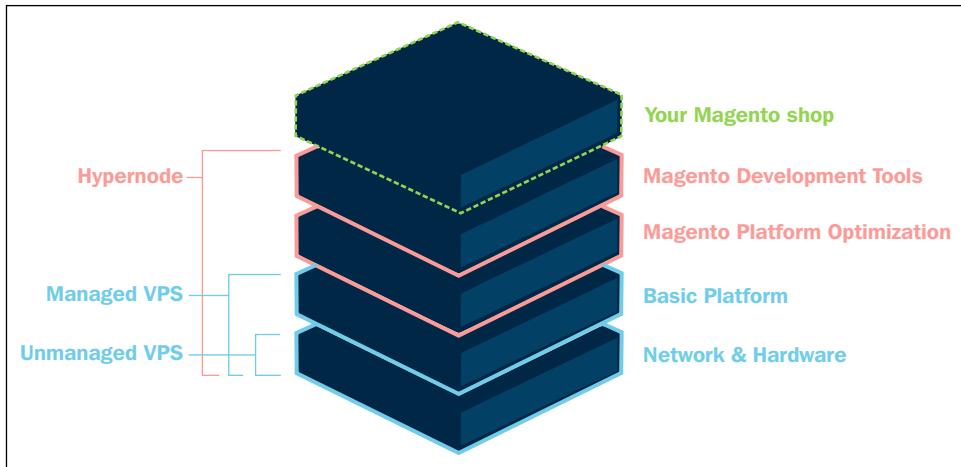
Throughout the following recipes, we will install Magento 2 on a fully managed platform hosted by Hypernode. Hypernode is an advanced platform for successful Magento shops.

Hypernode is a radically different approach to Magento hosting that actively improves your shop's performance. Its unique Magento platform is developed by Byte Internet, a Dutch hoster with eight years of experience in hosting Magento.

Hypernode has been developed in close consultation with Magento developers, with the objective of having Magento web shops perform to the best advantage and make its development several times easier. This recipe will show you how to do it.

[ With Hypernode, you will get your own fully managed isolated cloud server. This server has been fully configured for Magento 2. The best and latest software such as NGINX, HHVM, PHP-FPM, Redis, and MySQL are installed as a standard. In addition, they offer all the tools that you, as a Magento developer, would need in order to work comfortably: New Relic, Git, Vagrant, and Magento-specific tools such as ModMan, N98 Magerun, Sphinx Search, and much more.]

The following image illustrates working of Hypernode which can be found at <https://www.hypernode.com/>:



Getting ready

For this recipe, you will need to create an account at Hypernode <https://www.hypernode.com/>. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to create a Magento 2 hosting environment on the Hypernode platform. The following steps will guide you through this:

1. First, we start by creating a user account for Hypernode.

There are two options to create a Hypernode.

The first option is a default clean Magento 2 setup using the link, <https://auth.byte.nl/account/register/magento2>, and committing your name, e-mail, company name, and so on. For the following steps, make sure to submit the correct cell number. You will get a text message to confirm that this account is valid and not spam. Now perform steps 2 through 9 to set up Magento 2 the easy way.

If you want to start using Magento 1 before switching to Magento 2, then follow all of the steps until step 11. To do so, use the link, <https://auth.byte.nl/account/register/hypernode/>, and commit your name, e-mail, company name, and so on. For the following steps, make sure to submit the correct cell number. You will get a text message to confirm that this account is valid and not spam.

2. Now, open up your mail account, and confirm the e-mail that you received from Byte Internet. Next, you will be prompted with a text message on your cell phone; commit the code to continue. Make sure to pick a password for the Hypernode control panel, which we will use later.
3. Next, commit your domain name, or any other name that will relate to this setup. The name will be used to create a subdomain on the Hypernode. The name could be called `yourdomain.hypernode.io`.
4. After submitting your domain name, creating the Hypernode will take between 20-30 minutes, or less. All current Hypernodes are created on the same hosting platform that we used before at DigitalOcean. The difference between this setup for Magento 2 is that it is a managed setup. So we don't need to configure PHP, MySQL, NGINX, and so on.
5. Now go to the Hypernode control panel to check the current status on the creation of the Hypernode. Go to <https://service.byte.nl>.

- Click on your domain name and then click on Hypernode settings in the control panel. Here, you will find the current status and health of your setup:

Hypernode settings		
General information		
Application name	magento2cookbook	
Application type	Magento	
Application host	magento2cookbook.hypernode.io	
SSH information		
SSH Hostname	magento2cookbook.hypernode.io	
SSH Username	app	
SSH Authorized Keys		Manage keys >
SSL information		
SSL Certificates		None >
MySQL information		
MySQL Hostname	mysqlmaster.magento2cookbook.hypernode.io	
MySQL Port	3306	
MySQL Username	app	
MySQL Password		See documentation 

- Next, we need to set up an SSH key to log in via SSH. By default, this is the only way to log in. Go to <https://service.byte.nl/sshkeymanager/> and submit your personal key or create a new pair. It will take less than 10 minutes to auto-submit this to your Hypernode.
- Now you can log in to the Hypernode via the shell. The hostname is the same as your domain name, which we had set up in the beginning. The username is always app. The example should look as follows:

```
ssh app@yourdomain.hypernode.io
```
- If everything worked out fine, you should now have access to the Hypernode on <http://yourdomain.hypernode.io>.

10. By default, the current Hypernode setup has a Magento 1 preinstalled setup. We will now switch to Magento 2 using the following commands via the shell:

```
n98-magerun --root-dir=/data/web/public uninstall  
--installationFolder=/data/web/public -force
```

```
touch ~/nginx/magento2.flag
```

```
mkdir ~/magento2
```

```
cd ~/magento2
```

```
wget -qO- https://magento.mirror.hypernode.com/releases/magento2-  
latest.tar.gz | tar xfz -
```

```
echo "create database magento2" | mysql
```

```
chmod 755 bin/magento
```

```
cat ~/.my.cnf
```

```
bin/magento setup:install --db-host=[HOSTNAME]  
--dbname=[DATABASE] --db-user=app --db-password=[DATABASE_  
PASSWORD] --admin-firstname=[YOURFIRSTNAME] --admin-  
lastname=[YOURSURNAME] --admin-user=[ADMINNAME] --admin-  
password=[ADMINPASSWORD] --adminemail=[YOUR@EMAIL.COM] --base-  
url=[YOUR.HYPERNODE.IO] --language=[en_US] --timezone=[Europe/  
Berlin] --currency=[EUR] --use-rewrites=1
```

```
rm -rf ~/public
```

```
mkdir ~/public
```

```
cd ~/magento2
```

```
ln -fs ../magento2/pub/* ../public
```

The detailed instructions can also be found on the Hypernode knowledge base, <https://support.hypernode.com/knowledgebase/installing-magento-2-on-hypernode/>. This also includes a small tutorial regarding a sample data setup.

11. Now open your browser and go to your domain name to check whether everything is working correctly.
12. Congratulations, you just finished configuring Magento 2 on Hypernode.

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 4, we create a new Hypernode account at <https://www.byte.n1/>. These steps will create a new Byte customer that stores a reference to the Hypernode. The Hypernode is auto-created in the background on DigitalOcean. During the whole process, you will get multiple mails for validation and additional account information regarding the Hypernode logins.

In steps 5 through 11, we need to switch from the default Magento 1 preinstalled setup to Magento 2. All setups are self-explanatory and result in a clean Magento 2 setup located on a managed hosting platform and ready for production.

There's more...

Hypernode provides you with a large set of tools useful for your Magento setup. Check your service panel for the latest tools and links available. Here is a small section of some useful links:

<https://yourdomain.hypernode.io/phpmyadmin/>

https://support.hypernode.com/knowledgebase_category/getting-started/

<https://support.hypernode.com/knowledgebase/configure-redis/>

<https://support.hypernode.com/knowledgebase/varnish-on-hypernode/>

Managing Magento 2 on Docker

Docker is a new way of packaging your application and building containers for every single process. It is very lightweight and easy to set up. Creating building blocks for Apache, NGINX, MySQL, PHP, HHVM, and Magento individually and running them together can save lots of time during development, testing, and production.

In this recipe, we will not cover the Docker fundamentals but learn how to run a Magento 2 Docker setup on your DigitalOcean Droplet using existing containers.

Getting ready

Before we can start using Magento 2 on Docker, we need to create a clean Droplet. Back up your current Droplet by creating a snapshot. Here are some easy steps to create a snapshot and start a new Droplet with Docker preinstalled:

1. Power off your current Droplet in the DigitalOcean control panel.

- Select **Take Snapshot** and choose a name. (This may take some time depending on how big your current environment is.)



- Select **Rebuild Droplet**, which is in the **Destroy** menu, choose **Ubuntu docker (1.9.1 on 14.04)**, and press **Rebuild from Image**.

How to do it...

For the purpose of this recipe, let's assume that we need to create a Magento 2 Docker setup. The following steps will guide you through this:

- Log in to your Docker Droplet. You can check your current Docker version with the following command:
`docker -version`
- Before we start pulling the Magento 2 Docker container, we first need to pull a MySQL container. Run the following command in your shell:
`docker run -d --name mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=admin mysql:5.6`

The Docker run command will automatically download a MySQL 5.6 container, which will run in the background once it's done.

- Now we can check whether the MySQL container works. Run the `docker images` command, and then check which images are available. If you want to log in to the container, run the following command:

`docker exec -it mysql bash`

You can run any other command here, and then check whether MySQL is running. For example, run the `ps --aux` command and you will see the process of MySQL.

- Now we start pulling the Magento 2 Docker container to our local machine. We are using a prebuilt Docker container hosted at the Docker hub (<https://hub.docker.com/u/raybogman/>).

Run the following command to install a clean Magento 2 setup:

```
docker run --rm --name magento2 -it -p 80:80 --link mysql:mysql -e  
MYSQL_USER=root -e MYSQL_PASSWORD=admin -e PUBLIC_HOST=yourdomain.  
com raybogman/mage2cookbook-docker $*
```

Run the following command to install a Magento 2 setup including sample data:

```
docker run --rm --name magento2 -it -p 80:80 --link mysql:mysql -e  
MYSQL_USER=root -e MYSQL_PASSWORD=admin -e PUBLIC_HOST=yourdomain.  
com raybogman/mage2cookbook-sample-docker $*
```

Change the PUBLIC_HOST setting with your own IP or domain name. The SampleData version has all of its assets to create a preinstalled Magento 2 setup.

- Be patient now; this may take some time. The latest Magento 2 container is downloaded and executed on the fly. The final phase is the execution of Apache that will run in the foreground.
- Now open your browser and, depending on your public IP or domain name, execute this.
- Now we can check whether the Magento 2 container works. Run the `docker images` command, and then check which images are available. If you want to log in to the container, run the following command:

```
docker exec -it magento2 bash
```

You can run any other command here, and then check whether Apache 2 is running. For example, run the `ps --aux` command and you will see the process of Apache 2. All Magento 2 files are located at `/var/www/magento2`.

- Congratulations, you have Magento 2 running using a Docker container. The login credentials are the Magento username (admin), password (password123), and backend URL (<http://yourdomain.com/admin>).

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 8, we create a Docker setup for Magento 2.

In steps 2 and 3, we set up a MySQL container that will be downloaded automatically and run in the background.

In step 4, we download the prebuilt Magento 2 Docker container and connect it to the MySQL container. The installation process can take some time; Magento needs to deploy the whole setup.

In step 7, you learn how to get shell access to the Magento container and maintain it.

There's more...

To kill/shut down the current Magento 2 container, use the *Ctrl + C* command in the running shell.

This Magento 2 Docker container is designed only for demo purposes, and is there to run in the foreground and not as a daemon in the background.

Check out the source code of the Magento 2 Docker container on GitHub:

<https://github.com/mage2cookbook>.

Some basic commands to use Docker are as follows:

<code>docker ps</code>	Docker running containers
<code>docker images</code>	Docker local containers
<code>docker exec -it bash</code>	Access to running container shell
<code>docker rm -f \$(docker ps -a -q)</code>	Remove all running containers
<code>docker rmi -f \$(docker images -q)</code>	Remove all containers

2

Magento 2 System Tools

In this chapter, we will cover the basic tasks related to managing the system tools of Magento 2. You will learn the following recipes:

- ▶ Installing Magento 2 sample data via GUI
- ▶ Installing Magento 2 sample data via the command line
- ▶ Managing Magento 2 indexes via the command line
- ▶ Managing Magento 2 cache via the command line
- ▶ Managing Magento 2 backup via the command line
- ▶ Managing Magento 2 set mode (MAGE_MODE)
- ▶ Transferring your Magento 1 database to Magento 2

Introduction

This chapter explains how to install and manage Magento 2 on a production-like environment. We will be installing a new Magento 2 instance via the shell command with and without sample data. Besides the setup, managing Magento 2 is different from the current Magento version. We will be using a lot of tools from the command line so basic shell knowledge is advised. The command-line tool in the /bin directory is similar to the current **Swiss army knife** tool in the current Magento version known as **n98-magerun**.

Using `bin/magento` and **Composer** is one of the new key features in Magento 2 that will rock your world.

The recipes in this chapter will focus primarily on a more advanced setup of how to install Magento 2 and manage it. However, in some situations, we will dive in deeper related to the subject.

Here is an overview of all the command-line tools in Magento 2:

```
root@mage2cookbook:/var/www/html# bin/magento  
Magento CLI version 2.0.0
```

Usage:

```
command [options] [arguments]
```

Options:

```
--help (-h)          Display this help message  
--quiet (-q)         Do not output any message  
--verbose (-v|vv|vvv) Increase the verbosity of messages: 1 for normal  
output, 2 for more verbose output and 3 for debug  
--version (-V)       Display this application version  
--ansi               Force ANSI output  
--no-ansi             Disable ANSI output  
--no-interaction (-n) Do not ask any interactive question
```

The following commands are available in the command-line tools in Magento 2:

Commands	Description
help	This displays help for a command
list	This lists the commands
admin	
admin:user:create	This creates an administrator
admin:user:unlock	This unlocks the administrator account
cache	
cache:clean	This cleans the cache type(s)
cache:disable	This disables the cache type(s)
cache:enable	This enables the cache type(s)
cache:flush	This flushes the cache storage used by the cache type(s)
cache:status	This checks the cache status
catalog	
catalog:images:resize	This creates resized product images

Commands	Description
cron	
cron:run	This runs jobs by schedule
customer	
customer:hash:upgrade	This upgrades the customer's hash according to the latest algorithm
deploy	
deploy:mode:set	This sets the application mode
deploy:mode:show	This displays the current application mode
dev	
dev:source-theme:deploy	This collects and publishes source files for a theme
dev:tests:run	This runs tests
dev:urn-catalog:generate	This generates the catalog of URNs to *.xsd
dev:xml:convert	This converts XML files using XSL style sheets
i18n	
i18n:collect-phrases	This discovers phrases in the code base
i18n:pack	This saves language packages
i18n:uninstall	This uninstalls language packages
indexer	
indexer:info	This shows allowed indexers
indexer:reindex	This reindeces data
indexer:set-mode	This sets the index mode type
indexer:show-mode	This shows the index mode
indexer:status	This shows the status of an indexer
maintenance	
maintenance:allow-ips	This sets the maintenance mode exempt IPs
maintenance:disable	This disables the maintenance mode
maintenance:enable	This enables the maintenance mode
maintenance:status	This displays the maintenance mode status
module	
module:disable	This disables specified modules
module:enable	This enables specified modules
module:status	This displays the status of modules
module:uninstall	This uninstalls modules installed by Composer
sampleddata	
sampleddata:deploy	This deploys sample data modules

Commands	Description
sampledata:remove	This removes all sample data from composer.json
sampledata:reset	This resets sample data modules for reinstallation
theme	
theme:uninstall	This uninstalls the theme
info	
info:adminuri	This displays the Magento Admin URI
info:backups:list	This prints a list of available backup files
info:currency:list	This displays the list of available currencies
info:dependencies:show-framework	This shows the number of dependencies on the Magento framework
info:dependencies:show-modules	This shows the number of dependencies between modules
info:dependencies:show-modules-circular	This shows the number of circular dependencies between modules
info:language:list	This displays a list of available language locales
info:timezone:list	This displays a list of available time zones
setup	
setup:backup	This takes a backup of the Magento Application code base, media, and database
setup:config:set	This creates or modifies the deployment configuration
setup:cron:run	This runs a cron job scheduled for the setup application
setup:db-data:upgrade	This installs and upgrades data in the DB
setup:db-schema:upgrade	This installs and upgrades the DB schema
setup:db:status	This checks whether the DB schema or data require an upgrade
setup:di:compile	This generates the DI configuration and all non-existing interceptors and factories
setup:di:compile-multi-tenant	This generates all non-existing proxies and factories and precompiles class definitions, inheritance information, and plugin definitions
setup:install	This installs the Magento application
setup:performance:generate-fixtures	This generates fixtures
setup:rollback	This rolls back the Magento application code base, media, and database

Commands	Description
setup:static-content:deploy	This deploys static view files
setup:store-config:set	This installs the store configuration
setup:uninstall	This uninstalls the Magento application
setup:upgrade	This upgrades the Magento application, DB data, and schema

Throughout this chapter, you can pick your own preferred hosting setup as we set up in *Chapter 1, Installing Magento 2 on Apache and NGINX*. We will be using an NGINX-based setup. The Apache setup is pretty straightforward; when needed, we will address specified configuration settings when they occur.

Installing Magento 2 sample data via GUI

Installing Magento 2 via the **graphical user interface (GUI)** is not new. We have already done this in *Chapter 1, Installing Magento 2 on Apache and NGINX*. Now, we will be installing a new clean version including the sample data.

The sample data can be installed during and at the end of the procedure. We will be using a composer.json file for our setup prerequisites. First, we will be installing a clean version with sample data, and later, I will show you how to install it at the end in case you already have a preinstalled version.

Getting ready

For this recipe, we will use a Droplet created in *Chapter 1, Installing Magento 2 on Apache and NGINX*, at DigitalOcean, <https://www.digitalocean.com/>. We will be using NGINX, PHP-FPM, and a Composer-based setup. No other prerequisites are required.

How to do it...

For the first step, you can either create a new Droplet or rebuild a clean Droplet based on a Ubuntu or RedHat DigitalOcean image.

The option to rebuild is located in the DigitalOcean control panel in the **Destroy** menu and then rebuild Droplet:

The screenshot shows the DigitalOcean control panel interface for a droplet named "Ubuntu baseline-chapter-2". On the left, there's a sidebar with links: Access, Power, Resize, Snapshots, Settings, Backups, Graphs, History, and **Destroy**. The "Destroy" link is highlighted in blue. The main area has two sections: "Destroy Droplet" and "Rebuild Droplet".

Destroy Droplet: A warning message says "This is irreversible. We will destroy your Droplet and all associated backups." There's a checked checkbox labeled "Scrub Data" and a red-bordered "Destroy" button.

Rebuild Droplet: A message says "This will rebuild your Droplet using the original image you specified when you deployed. Please be advised that all data will be lost." Below it is a dropdown menu titled "Select an Image" which lists various OS images. The "Ubuntu 15.04 x64" option is selected and highlighted with a blue bar.

The steps to install Magento 2 sample data via GUI are as follows:

1. Preferably, pick the new snapshot or the already created one and press **Rebuild from Image**. The rebuild will take around 60 seconds.
2. Now log in to your new or current Droplet. We will be referring to a new build Droplet throughout this recipe.

3. Now let's download the latest Magento 2 version including sample data. Go to <https://www.magentocommerce.com/download>, pick **Full Release with Sample Data (ZIP with sample data)**, and unpack this in your web directory. We refer to /var/www/html in this recipe.

If you are using a ZIP package, make sure that you install the unzip package first, running the following command on the shell:

```
apt-get install unzip
```

4. Now let's set the ownership and permissions:

```
chown -R www-data:www-data /var/www/html
```

```
find . -type d -exec chmod 770 {} \; && find . -type f -exec chmod 660 {} \; && chmod u+x bin/magento
```

5. Now we use Composer to resolve all of our dependencies. Run the following command from the shell:

```
cd /var/www/html/ && composer install
```

During the installation process, you will get a notice to create a GitHub OAuth token. The download rate limit is pretty small. Copy the URL from your shell window in your browser, log in at GitHub or create an account, and create the token. You may also check *Chapter 1, Installing Magento 2 on Apache and NGINX*, for more details on this topic.

6. Next, we will be using the setup wizard to continue the rest of the installation. In this chapter, we will be using the shell installation method. Go to your favorite browser and enter the following URL:

```
http://yourdomain.com/setup
```

Continue your flow until **Step 4: Customize Your Store**. As we have chosen a Magento 2 package including sample data, all software modules are preinstalled and listed in the advanced modules configurations list.

Here is an overview of all the modules selected by Magento that can be managed during installation:

Advanced Modules Configurations

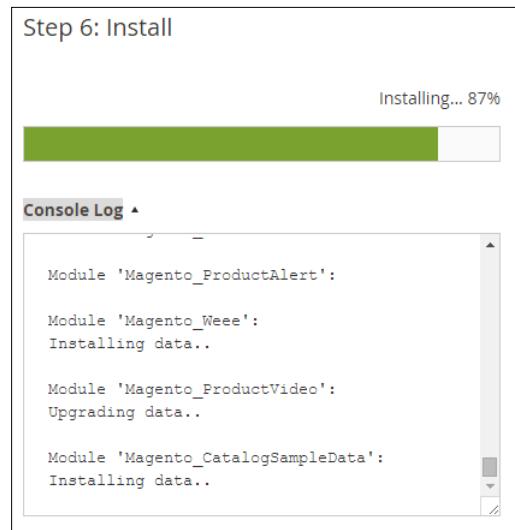
Select All

- Magento_AdminNotification
- Magento_AdvancedPricingImportExport
- Magento_Authorization
- Magento_Authorizenet
- Magento_Backend
- Magento_Backup
- Magento_Braintree
- Magento_Bundle
- Magento_BundleImportExport
- Magento_BundleSampleData
- Magento_CacheInvalidate
- Magento_Captcha
- Magento_Catalog
- Magento_CatalogImportExport
- Magento_CatalogInventory
- Magento_CatalogRule
- Magento_CatalogRuleConfigurable
- Magento_CatalogRuleSampleData
- Magento_CatalogSampleData
- Magento_CatalogSearch
- Magento_CatalogUrlRewrite

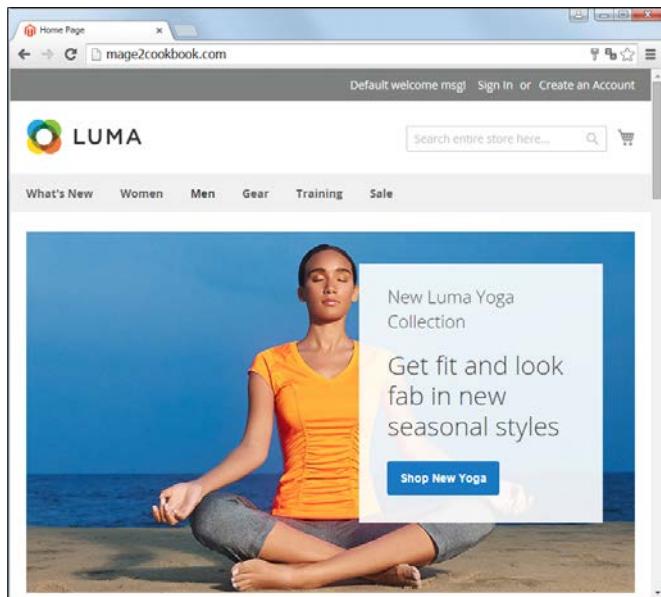
117 out of 117 selected

- Now, continue the rest of the steps and install Magento 2. Installing sample data can take some time, so don't close or refresh your browser.

The progress bar and console log will share all details regarding the current status:



Congratulations, you just finished the installation of Magento 2 including sample data. Now go to your browser using `yourdomain.com`; you will see the default Magento 2 layout theme called **Luma**, as follows:



How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 6, we created a Magento 2 version including sample data. The process is almost the same as we did in the previous chapter. The most important change is that we downloaded a full version including sample data. In this build, Magento submitted all of the media files and data sample packages that we need to complete the process. During the graphical setup, we were able to choose which sample data packages are needed. This process could take some time to complete.

There's more...

In the `/var` directory, you can find the following hidden file with the current state of the installed sample data:

```
less /var/www/html/var/.sample-data-state.flag
```

Want to start all over again? Magento 2 has a magic uninstall option that cleans everything on the fly, cache, and database. You can use the `php bin/magento setup:uninstall` command on the shell, as shown here:

```
root@mage2cookbook:/var/www/html# php bin/magento setup:uninstall
Are you sure you want to uninstall Magento? [y/N]y
Starting Magento uninstallation:
Cache cleared successfully
Cleaning up database `magento2`
File system cleanup:
/var/www/html/pub/static/_requirejs
/var/www/html/pub/static/adminhtml
/var/www/html/pub/static/frontend
/var/www/html/var/cache
/var/www/html/var/composer_home
/var/www/html/var/di
/var/www/html/var/generation
/var/www/html/var/log
/var/www/html/var/page_cache
/var/www/html/var/tmp
/var/www/html/var/view_preprocessed
/var/www/html/app/etc/config.php
/var/www/html/app/etc/env.php
```

Installing Magento 2 sample data via the command line

Installing Magento 2 via the shell is not new. In the current Magento version, it was already possible using the `install.php` file. The configuration looked like this:

```
/usr/local/bin/php -f install.php -- \
--license_agreement_accepted "yes" \
--locale "en_US" \
--timezone "America/Los_Angeles" \
--default_currency "USD" \
--db_host "mysql.example.com" \
--db_name "your_db_name" \
--db_user "your_db_username" \
--db_pass "your_db_password" \
--db_prefix "" \
--admin_frontname "admin" \
--url "http://www.examplesite.com/store" \
--use_rewrites "yes" \
--use_secure "no" \
--secure_base_url "" \
--use_secure_admin "no" \
--admin_firstname "your_first_name" \
--admin_lastname "your_last_name" \
--admin_email "your_email@example.com" \
--admin_username "your_admin_username" \
--admin_password "your_admin_password"
```

It was very easy to script and use multiple times on any given environment.

In Magento 2, the logic stayed the same but now, it's much easier to use. We will be using a `composer.json` file for our setup prerequisites.

Getting ready

For this recipe, we will use a Droplet created in *Chapter 1, Installing Magento 2 on Apache and NGINX*, at DigitalOcean, <https://www.digitalocean.com/>. We will be using NGINX, PHP-FPM, and a Composer-based setup. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to create a Magento 2 hosting environment including sample data. The following steps will guide you through this:

1. In this recipe, check your `/root/.composer/auth.json` file if you have a Magento of GitHub repository token, username, and password. If not, create them. Here is an example (the username and password are dummies):

```
{  
    "http-basic": {  
        "repo.magento.com": {  
            "username": "256e8f49b66689ecf18b07bc3cc2ca2d",  
            "password": "cb1c7ef2e14b666d8a4e99fe40c8393a"  
        }  
    },  
    "github-oauth": {  
        "github.com": "e960f7000803e2832ce5f7a637d58a666"  
    }  
}
```

You can create a Magento authentication key in the user section of the Magento connect portal. Go to <http://www.magentocommerce.com/magento-connect/>, navigate to the **Developers | Secure Keys** menu item, and create one. **Public Key** is your username and **Private Key** is your password.

The GitHub token can be created under the tokens section of your GitHub account. Go to <https://github.com/settings/tokens> and press **Generate new token**. Copy the token in your `auth.json` file of your home or root directory.

2. Now, let's create a Composer project using the shell. Go to your web server `/var/www/html` directory and use the following command:

```
composer create-project "magento/project-community-edition" /var/www/html --prefer-dist --repository-url https://repo.magento.com/
```

3. Now use the following command on the shell. This will add the sample data package to the Magento `composer.json` file.

```
php bin/magento sampledata:deploy
```

You may get the following error notice; ignore this once you set up the `auth.json` file:

```
[Composer\Downloader\TransportException]
```

```
The 'https://repo.magento.com/packages.json' URL required authentication.
```

```
You must be using the interactive console to authenticate
```

- Run the following command on the shell. This will download all sample data to your Magento 2 environment.

```
composer update
```

- Make sure that you have the correct file permission before you continue:

```
chown -R www-data:www-data /var/www/html
```

The easy way to continue is to visit our setup page from the browser as we did in the *Installing Magento 2 sample data via GUI* recipe of this chapter using the `http://yourdomain.com/setup` URL.

- We can also use the shell to finish the setup using the following script:

```
bin/magento setup:install \
--db-host=localhost \
--db-name=<your-db-name> \
--db-user=<db-user>" \
--db-password=<db-password>" \
--backend-frontname=<admin-path> \
--base-url=http://yourdomain.com/ \
--admin-lastname=<your-lastname> \
--admin-firstname=<your-firstname> \
--admin-email=<your-email> \
--admin-user=<your-admin-user> \
--admin-password=<your-password> \
```



Always make sure that `bin/magento` has the correct permissions to execute:

```
chmod 755 bin/magento
```

Congratulations, you just finished the installation of Magento 2 including sample data. Now go to your browser using `yourdomain.com`, and you will see the default Magento 2 layout theme called Luma.

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 6, we installed Magento 2 via the command shell. Before we could continue, we needed to create an `auth.json` file that stores our Magento and GitHub tokens. Without them, we may not be able to install the software easily due to download restrictions or dependencies.

In step 2, we used one line of code to trigger the whole download process of Magento 2. Depending on whether this is your first call, the process can take some time. Once you install Magento for the second time, the process installs much faster because of the locally stored cache files.

In step 3, we used `bin/magento` to update the `composer.json` file including the sample data packages. Then we needed to update Composer before we could install Magento 2 from the shell; otherwise, the sample data would not be included.

In step 6, we used the `bin/magento setup:install` parameter to commit all of the database, URL admin path, domain name, and user credentials to the setup of Magento 2.

There's more...

Always make sure that your system PATH is exported to your system using the following command from the shell (where `/var/www/html` is your Magento root folder):

```
export PATH=$PATH:/var/www/html/bin
```

More information on environmental variables can be found here:

<https://www.digitalocean.com/community/tutorials/how-to-read-and-set-environmental-and-shell-variables-on-a-linux-vps>

Managing Magento 2 indexes via the command line

In the current version of Magento, using indexes is one of the most important key features. Without the correct indexes, we will not be able to use Magento properly.

What do the indexes do, and why are they so important? One of the key elements is to make things run faster. Without indexing, Magento 2 would have to calculate data on the fly. In Magento 2, we will be using the following nine indexes:

- ▶ **Customer Grid:** This indexer rebuilds the customer's grid. This is a new indexer in Magento 2 for optimized rendering of the `adminhtml` backend pages.
- ▶ **Category Products:** This indexer creates the association between categories and products based on the associations that you set in the backend on the categories and relates to the Product Categories indexer. The flat catalog indexer creates a flat optimized table in the database.
- ▶ **Product Categories:** This indexer creates the association between products and categories based on the associations that you set in the backend on the products and relates to the Category Products indexer. The flat product indexer creates a flat optimized table in the database.

- ▶ **Product Price:** This indexer relates to the products and their advanced pricing options based on, for example, customer group, website, and catalog discount rules. The indexer aggregates the data in tables (`catalog_product_index_price_*`) and makes the selects (sorting and filtering) much easier.
- ▶ **Product EAV:** This indexer reorganizes the **Entity Attribute Value (EAV)** product structure to the flat structure. The product EAV indexer is related to the Category Products and Product Categories indexer and creates a flat optimized table in the database.
- ▶ **Stock:** This indexer rebuilds and calculates the current stock data.
- ▶ **Catalog Search:** This indexer rebuilds the catalog product fulltext search.
- ▶ **Catalog Rule Product:** This indexer creates and updates the created catalog price rule set.
- ▶ **Catalog Product Rule:** This indexer creates and updates the created shopping cart price rule set.

The database table looks as follows:

state_id	indexer_id	status
1	customer_grid	valid
2	catalog_category_product	invalid
3	catalog_product_category	invalid
4	catalog_product_price	valid
5	catalog_product_attribute	valid
6	cataloginventory_stock	valid
7	catalogsearch_fulltext	valid
8	catalogrule_rule	invalid
9	catalogrule_product	valid

The control panel of the backend looks like this:

	Indexer	Description	Mode	Status	Updated
<input type="checkbox"/>	Customer Grid	Rebuild Customer grid index	UPDATE ON SAVE	READY	Nov 23, 2015, 9:18:31 PM
<input type="checkbox"/>	Category Products	Indexed category/products association	UPDATE ON SAVE	REINDEX REQUIRED	Nov 23, 2015, 9:19:05 PM
<input type="checkbox"/>	Product Categories	Indexed product/categories association	UPDATE ON SAVE	REINDEX REQUIRED	Nov 23, 2015, 9:19:05 PM
<input type="checkbox"/>	Product Price	Index product prices	UPDATE ON SAVE	READY	Nov 23, 2015, 9:18:33 PM
<input type="checkbox"/>	Product EAV	Index product EAV	UPDATE ON SAVE	READY	Nov 23, 2015, 9:18:34 PM
<input type="checkbox"/>	Stock	Index stock	UPDATE ON SAVE	READY	Nov 23, 2015, 9:18:34 PM
<input type="checkbox"/>	Catalog Search	Rebuild Catalog product fulltext search index	UPDATE ON SAVE	READY	Nov 23, 2015, 9:18:37 PM
<input type="checkbox"/>	Catalog Rule Product	Indexed rule/product association	UPDATE ON SAVE	REINDEX REQUIRED	Nov 23, 2015, 9:19:14 PM
<input type="checkbox"/>	Catalog Product Rule	Indexed product/rule association	UPDATE ON SAVE	READY	Nov 23, 2015, 9:18:37 PM

Getting ready

For this recipe, we will use a Droplet created in *Chapter 1, Installing Magento 2 on Apache and NGINX*, at DigitalOcean, <https://www.digitalocean.com/>. We will be using NGINX, PHP-FPM, and a Composer-based setup. A working Magento 2 setup is required.

How to do it...

The following are the steps to implement the recipe:

1. In Magento 2, we only have the option to change the **Update on Save** or **Update by Schedule** state:

Update on Save: Index tables are updated immediately after the dictionary data is changed

Update by Schedule: Index tables are updated by cron job according to the configured schedule

2. We will be using the shell during this. This will be the only way to update your indexes. Go to your shell and run the following command from your web server directory:

```
php bin/magento indexer:info
```

We now get an overview of all the indexers.

3. Now run `php bin/magento indexer:status`; this will give us an up-to-date status of the current indexes.

4. Now run `php bin/magento indexer:show-mode`; this information is related to the Update on Save or Update by Schedule modes.

5. We can switch the modes using the following command on the shell:

```
php bin/magento indexer:info realtime customer_grid
```

Using the mode option **realtime** (Update on Save) or **schedule** (Update by Schedule) may set the indexer.

6. One of the most commonly used commands is as follows:

```
php bin/magento indexer:reindex
```

This will reindex all the indexers. However, you can also reindex them individually using the following command:

```
php bin/magento indexer:reindex customer_grid
```

We can also use the following command:

```
php bin/magento indexer:reindex customer_grid catalog_category_
product etc...
```

7. Now you can rerun `indexer:status` to check whether all the indexers are up to date.

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 7, you learned how to use the `bin/magento` indexer.

In step 5, you learned how to check what the current status is of all the indexes. In step 4, we saw how to switch from the realtime Update on Save to the schedule Update on Schedule modes.

In step 6, you learned how to reindex them individually.

There's more...

You can also check the current status of the indexer using MySQL. Run the following command in the shell:

```
mysql -u <username> --database <dbname> -p -e "select * from indexer_
state"
```

Managing Magento 2 cache via the command line

Cache management is one of the new optimized key features in Magento 2. We will be using the following cache types:

Cache types	Cache type code name	Description
Configuration	config	Magento collects configuration from all modules, merges it, and saves the merged result to the cache. This cache also contains store-specific settings stored in the filesystem and database. Clean or flush this cache type after modifying configuration files.

Cache types	Cache type code name	Description
Layout	layout	<p>This is the compiled page layout (that is, the layout components from all components).</p> <p>Clean or flush this cache type after modifying layout files.</p>
Block HTML output	block_html	<p>This is the HTML page fragments per block.</p> <p>Clean or flush this cache type after modifying the view layer.</p>
Collections data	collections	<p>This is the result of database queries.</p> <p>If necessary, Magento cleans up this cache automatically, but third-party developers can put any data in any segment of the cache.</p> <p>Clean or flush this cache type if your custom module uses logic that results in cache entries that Magento cannot clean.</p>
DDL	db_ddl	<p>This is the database schema.</p> <p>If necessary, Magento cleans up this cache automatically, but third-party developers can put any data in any segment of the cache.</p> <p>Clean or flush this cache type after you make custom changes to the database schema (in other words, updates that Magento does not make itself).</p> <p>One way to update the database schema automatically is using the <code>magento setup:db-schema:upgrade</code> command.</p>
EAV	eav	<p>This is metadata related to EAV attributes (for example, store labels, links to related PHP code, attribute rendering, search settings, and so on).</p> <p>You should not typically need to clean or flush this cache type.</p>

Cache types	Cache type code name	Description
Page cache	full_page	This is the generated HTML pages. If necessary, Magento cleans up this cache automatically, but third-party developers can put any data in any segment of the cache. Clean or flush this cache type after modifying code level that affects HTML output. It's recommended to keep this cache enabled because caching HTML improves performance significantly.
Reflection	reflection	This removes a dependency between the web API module and the Customer module.
Translations	translate	This is the merged translations from all modules.
Integration configuration	config_integration	This is the compiled integrations. Clean or flush this cache after changing or adding integrations.
Integration API configuration	config_integration_api	This is the compiled integration APIs.
Web services configuration	config_webservice	This is the web API structure.

By default, we will be using Full Page Cache now in the community version, which is a great improvement next to the web services (API) caches.

Depending on the current **development**, **default**, and **production** state, caches will be different.

In the next recipes of this chapter, we will dive deeper into the use of different states.

Getting ready

When cleaning or flushing your cache, Magento will flush its content from either the `var/cache` or `var/full_page` directory. In this recipe, we will refer to the `bin/magento cache:enable`, `bin/magento cache:disable`, `bin/magento cache:clean`, or `bin/magento cache:flush` options.

How to do it...

For the purpose of this recipe, let's assume that we need to manage the Magento 2 cache setup. The following steps will guide you through this:

1. Let's first check the current status using the following command:

```
php bin/magento cache:status
```

The output looks like this:

A screenshot of a terminal window titled "root@mage2cookbook: /var/www/html". The window shows the command "php bin/magento cache:status" being run and its output. The output displays the current status of various cache types, each with a value of 1. The cache types listed are: config, layout, block_html, collections, reflection, db_ddl, eav, config_integration, config_integration_api, full_page, translate, and config_webservice.

```
root@mage2cookbook:/var/www/html# php bin/magento cache:status
Current status:
    config: 1
    layout: 1
    block_html: 1
    collections: 1
    reflection: 1
        db_ddl: 1
            eav: 1
    config_integration: 1
    config_integration_api: 1
        full_page: 1
        translate: 1
    config_webservice: 1
root@mage2cookbook:/var/www/html#
```

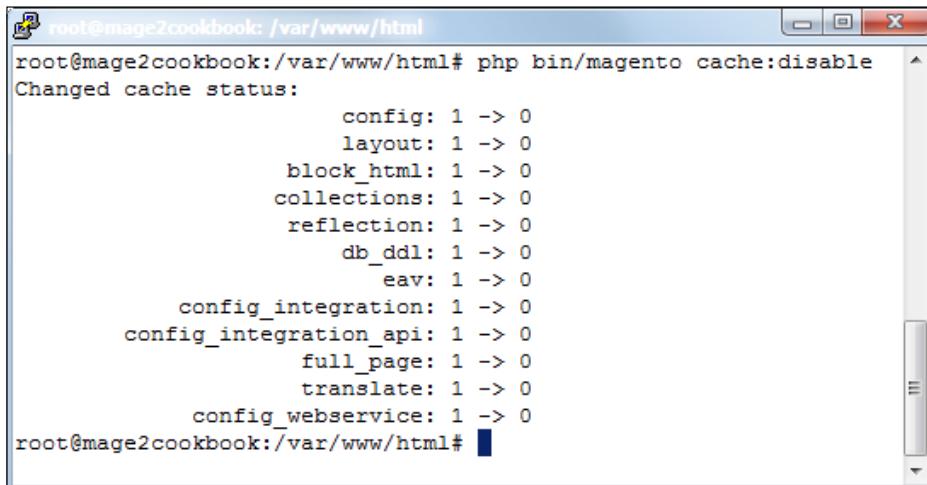
2. Now we will check how to enable and disable caches individually or all at once. Use the following command to disable the caches individually:

```
php bin/magento cache:disable config
```

This will disable the cache for only config. You may pick any cache type code name. To enable the config cache back, use the following command:

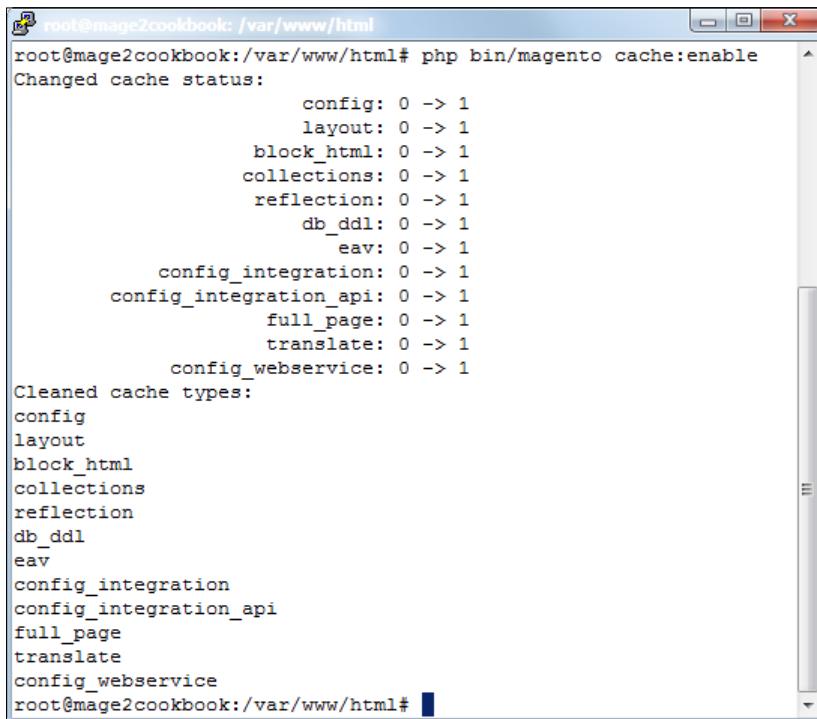
```
php bin/magento cache:enable config
```

When skipping the cache type code name behind the command, we will be able to enable or disable the caches all at once. It will look like this when we disable the cache:



```
root@mage2cookbook:/var/www/html# php bin/magento cache:disable
Changed cache status:
    config: 1 -> 0
    layout: 1 -> 0
    block_html: 1 -> 0
    collections: 1 -> 0
    reflection: 1 -> 0
    db_ddl: 1 -> 0
    eav: 1 -> 0
    config_integration: 1 -> 0
    config_integration_api: 1 -> 0
    full_page: 1 -> 0
    translate: 1 -> 0
    config_webservice: 1 -> 0
root@mage2cookbook:/var/www/html#
```

3. When we want to re-enable all caches, we use `php bin/magento cache:enable`. As you can see now, after enabling the caches, they are cleaned as well:



```
root@mage2cookbook:/var/www/html# php bin/magento cache:enable
Changed cache status:
    config: 0 -> 1
    layout: 0 -> 1
    block_html: 0 -> 1
    collections: 0 -> 1
    reflection: 0 -> 1
    db_ddl: 0 -> 1
    eav: 0 -> 1
    config_integration: 0 -> 1
    config_integration_api: 0 -> 1
    full_page: 0 -> 1
    translate: 0 -> 1
    config_webservice: 0 -> 1
Cleaned cache types:
config
layout
block_html
collections
reflection
db_ddl
eav
config_integration
config_integration_api
full_page
translate
config_webservice
root@mage2cookbook:/var/www/html#
```

4. Now let's clean the caches individually using `php bin/magento cache:clean config`. By removing the cache type code name, we will be able to clean all of them at once.
5. Now let's flush the caches individually using `php bin/magento cache:flush config`. By removing the cache type code name, we will be able to clean all of them at once.



Cleaning a cache type deletes all items from enabled Magento cache types only. In other words, this option does not affect other processes or applications because it cleans only the cache that Magento uses.

Disabled cache types are not cleaned.

Flushing a cache type purges the cache storage (such as Redis, Memcache, and so on), which might affect other process' applications that are using the same storage.

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 5, you learned how to manage the cache in Magento 2.

In step 1, you learned how to use the `status` option to check what the current cache status is. In step 2, we were able to enable or disable the caches individually.

In steps 4 and 5, you learned how to flush and clean the caches individually.

There's more...

You can also flush all cached items running the following command from the shell:

```
php bin/magento cache:flush -all
```

Keep in mind that cleaning or flushing your Full Page Cache can resolve in a cold (no-cache) page, so warming up all pages is advised.

Managing Magento 2 backup via the command line

Every production environment needs a proper backup plan. By default, Magento 2 has a complete set of options to create and roll back backups.

When creating a backup, we can use one of the following options:

Option	Meaning	Backup file name
--code	This creates a backup from the filesystem (excluding /var and /pub/static)	var/backups/<timestamp>_filesystem.tgz
--media	This creates a backup from the /media directory	var/backups/<timestamp>_filesystem_media.tgz
-db	This creates a backup from the current database	var/backups/<timestamp>_db.gz



Besides the Magento backup options, it is always advisable to use an alternative backup solution connected to a backup storage.



Getting ready

When creating a backup, Magento will store it in the var/backups directory. In this recipe, we will refer to the bin/magento setup:backup and bin/magento setup:rollback options.

How to do it...

For the purpose of this recipe, let's assume that we need to manage the Magento 2 backup setup. The following steps will guide you through this:

1. Let's start creating a code-only backup using the following command:

```
php bin/magento setup:backup --code
```

Once Magento sets the maintenance code flag, the web shop is offline for everybody. Creating a code backup takes some time. The backup will be stored in var/backups. A clean Magento 2 code backup is around 123 MB.

2. Now we will create a database-only backup using the following command:

```
php bin/magento setup:backup --db
```

A clean Magento 2 database backup is around 14 MB.

3. For a media backup, we use the following command:

```
php bin/magento setup:backup --media
```

A clean Magento 2 media backup with sample data is around 153 MB.

- Now let's check what backups have been created, using the following command:

```
php bin/magento info:backup:list
```

This will give us an overview of what's available:

```
Showing backup files in /var/www/html/var/backups.
```

Backup Filename	Backup Type
1448480907_filesystem_code.tgz	code
1448481232_db.gz	db
1448481295_filesystem_media.tgz	media

- Rolling back a backup is very easy. You can use the following command on the shell:

```
php bin/magento setup:rollback [-c|--code-file=<name>] [-m|--media-file=<name>] [-d|--db-file=<name>]
```

For example, to restore a database backup, we can use the following command:

```
php bin/magento setup:rollback -d 1448481232_db.gz
```

You will get the following notification to confirm your action:

```
You are about to remove current code and/or database tables. Are you sure? [y/N]
```



While confirming the action, you could get a **Segmentation fault**. You can fix this using the following command. This could be related to PHP 7. Always use the latest version:

```
ulimit -s 65536
```

You can also store this in the `.bashrc` file on your system.

- Congratulations, you just rolled back a backup. Pick any type to do the same. Always flush and clean your cache once you are done.

- Setting up a scheduled backup schema is a whole different ball game. We first need to set up a cron job using the following command:

```
crontab -e  
*/1 * * * * php /var/www/html/bin/magento cron:run
```

This command will create a cron schedule in the Magento 2 database.

8. Creating a daily, weekly, or monthly backup can be done in the administrator backend. Log in and navigate to **Stores | Configuration | Advanced | System | Scheduled Backup Settings**:

ADVANCED		Scheduled Backup Settings		
Admin		Enable Scheduled Backup	Yes	[GLOBAL]
System		Backup Type	Database	[GLOBAL]
Advanced		Start Time	00 : 00 : 00	[GLOBAL]
Developer		Frequency	Daily	[GLOBAL]
		Maintenance Mode	Yes	[GLOBAL]
Please put your store into maintenance mode during backup.				

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 8, you learned how to create rollbacks and manage backups via the command line.

In step 1, we created a backup for our code base only. In step 2, we created a backup for the database and in step 3, for all the media files.

In step 4, we listed all the created backups that are located in `var/backups`.

The process in step 5 is related to the rollback scenario. Depending on the backup size, rolling back could take some time.

In steps 7 and 8, we configured a cron job to schedule our daily backup process automatically.

There's more...

You can also check the current status of the cron schedule using MySQL. Run the following command from the shell:

```
mysql -u <username> --database <dbname> -p -e "select * from cron_schedule"
```

Managing Magento 2 set mode (MAGE_MODE)

Magento 2 comes with a new feature set called MAGE_MODE. This option gives you the configuration to run Magento in either **developer**, **default**, or **production** mode.

This feature is very important during development and product phases. It gives a developer the tool to debug or created optimized caches for high performance needs.

By default, the **default** mode is set.

The following table describes the modes in which we can run Magento:

Mode name	Description
default	<p>When no given mode is given, this is explicitly set and has the following benefits:</p> <ul style="list-style-type: none"> • Static view file caching is enabled • Enables automatic code compilation (code optimization) • Exceptions are written to the log files. • Hides the custom X-Magento-* HTTP response header
developer	<p>It has the following benefits:</p> <ul style="list-style-type: none"> • Disables static view file caching • Enables automatic code compilation (code optimization) • Shows the custom X-Magento-* HTTP response header • Verbose logging • Slowest performance state
production	<p>It has the following benefits:</p> <ul style="list-style-type: none"> • Optimized caches available • Exceptions are written to the log files. • Static files are not cached

[ When using a **Development Test Acceptance Production (DTAP)** environment, it is important to run your DTA in development mode and production on P.]

Getting ready

Always check what web server you are using; the settings for Apache and NGINX are not the same. In this recipe, we will be showing you how to set this on both of them.

How to do it...

For the purpose of this recipe, let's assume that we need to manage the Magento 2 production or development setup. The following steps will guide you through this:

1. In this recipe, the setup of the MAGE_SET option is different for NGINX and Apache. In Apache, we can use either the .htaccess file or configure this in the vhost file. We will first look into the Apache setup. While all recipes of this chapter are based on NGINX, it's best to skip this part and continue to the next listed topic or retrieve an DigitalOcean Droplet that we had set up in *Chapter 1, Installing Magento 2 on Apache and NGINX*.

Go to the .htaccess file in your web root directory and remove the # (hash or pound sign) at the fifth line from the top:

```
SetEnv MAGE_MODE developer
```

Change it to the following:

```
SetEnv MAGE_MODE production
```

If you are using a server-based configuration instead of the .htaccess file, use the following then:

```
SetEnv MAGE_MODE "developer"
```

The following code is for the current 000-default.conf:

```
<VirtualHost *:80>
ServerAdmin webmaster@localhost
DocumentRoot /var/www/html
SetEnv MAGE_MODE "developer"
<Directory /var/www/html>
Options Indexes FollowSymLinks
AllowOverride All
Order allow,deny
allow from all
</Directory>

ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined

ProxyPassMatch ^/(.*\.\php(.*))\$ fcgi://127.0.0.1:9000/var/www/
html/\$1
</VirtualHost>
```

Now continue to step 3.

- Now let's do the setup for NGINX. Go to your vhost file. If you are using the NGINX setup from *Chapter 1, Installing Magento 2 on Apache and NGINX*, you will find it in `/etc/nginx/conf.d/default.conf`.

Open the `default.conf` file and go to the following rule:

```
set $MAGE_MODE developer;
```

Change this to the following:

```
set $MAGE_MODE production;
```

Restart your NGINX server now using the following command:

```
service nginx restart
```

- Check the current status in Magento using the following command:

```
php bin/magento deploy:mode:show
```

By default, it shows the default status. We now switch the status using the following:

```
php bin/magento deploy:mode:set production
```

We can also use the following command:

```
php bin/magento deploy:mode:set developer
```

This will trigger the maintenance mode and start creating all necessary optimized static files needed.

- We can also run this manually. However, first we will need to create static files in the `pub/static` directory. Run the following command on the shell:

```
php bin/magento setup:static-content:deploy
```

- If you want to skip the code compilation, use the `--skip-compilation` option, as shown in the following command:

```
php bin/magento deploy:mode:set developer --skip-compilation
```

- Remember to check your permissions and ownership of the newly created files.



Code compilation consists of caches, optimized code, optimized dependency injection, proxies, and so on. Magento 2 needs these files to serve an optimized code base to the client's browser.

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 6, you learned configuring the development and production modes in Magento 2.

In step 1, we configured the `SetEnv` parameter in the `.htaccess` file of Apache to set the correct mode. There is also an option to configure this in the Apache configuration file instead.

In step 2, we configured the `set $MAGE_MODE` parameter in NGINX to use the correct mode.

In step 3, we used the `bin/magento deploy` option to tell Magento to start using the selected mode in NGINX or Apache, and create additional static files when running in production mode or show the correct debug headers in the developer mode.

In step 4, you learned how to deploy static content in the `pub/static` directory when running in production mode. This option will trigger the whole process of merging and compiling the correct code in the public folder.

There's more...

Use `curl` to check your HTTP response header to see what current state you are running, as shown in the following:

```
curl -I http://mage2cookbook.com/
```

```
HTTP/1.1 200 OK
Server: nginx/1.9.6
X-Magento-Cache-Debug: HIT
```

Always check the current status in your HTTP header and Magento shell. Only setting the web server configuration will not automatically trigger the Magento configuration and can mislead you.

Transferring your Magento 1 database to Magento 2

Moving your Magento 1 to Magento 2 may be one of the most challenging things out there. Luckily, Magento supported us with a database migration option.

The Magento 2 **Data Migration Tool** is here to help you convert your products, customers, order/sales data, store configuration, promotions/sales rules, and more to move to a clean Magento 2 setup.

Custom code, Extensions, and Themes are out of the current scope of the Data Migration Tool.

The currently supported migrations are the **Community Edition (CE)** versions 1.6.x, 1.7.x, 1.8.x, and 1.9.x and **Enterprise Edition (EE)** versions 1.11.x, 1.12.x, 1.14.x, and 1.14.x.



Check with your third-party extension developer for a Data Migration Tool to move the database code base to Magento 2.



Getting ready

Before we can start migrating our system, we need to check the following:

- ▶ Have a clean Magento 2 system running as we set up in *Chapter 1, Installing Magento 2 on Apache and NGINX*.
- ▶ Disable your cron jobs.
- ▶ Always back up your databases and old and new Magento versions.
- ▶ Check whether there is a network connection from the current Magento 1 to Magento 2 server. Check the firewall for database access if needed (port 3306).
- ▶ Only use the exact version number, so that the data-migration-tool 2.0.0 corresponds with Magento 2.0.0.



You may copy your current production database to the new Magento 2 server and run it there.



A migration of Magento 1 to Magento 2 has the following five phases that are important to follow in the correct order:

1. **Settings:** Migration of the settings is step 1. This will transfer all information from the stores, website, and system configuration.
The command is `php bin/magento migrate:settings`.
2. **Data:** Migration of the data is step 2. This will transfer all categories, products, customers, orders, wishlists, ratings, and so on
The command is `php bin/magento migrate:data`.
3. **Delta:** Migration of the delta is step 3. This is an important step and is used to transfer Magento 1 data to Magento 2 where new updates occur. It will update the most recent data of customers, orders, or other customer-related data. It is common to use this command before going live.
The command is `php bin/magento migrate:delta`.
4. **Media:** Migration of the media files is easy; just copy all files from `/media` to `/pub/media`.

5. **Custom modules/themes:** Migration of your modules or themes is out of the scope of the migration tool. Contact your solutions provider to check whether they have a new version available. This also applies to any custom-made themes or theme packages bought online.

How to do it...

For the purpose of this recipe, let's assume that we need to manage a Magento 1 to Magento 2 migration setup. The following steps will guide you through this:

1. First, we need to run the following command to add data-migration-tool to your current Composer setup:

```
composer config repositories.data-migration-tool git https://github.com/magento/data-migration-tool-ce
composer require magento/data-migration-tool:dev-master
```

Wait while all dependencies are updated.

2. Now check whether the migration tools are available in the bin/magento shell tool:

Commands	Description
migrate	
migrate:data	Main migration of data
migrate:delta	Migration of the data that is added to Magento after the main migration
migrate:settings	Migration of the system configuration

3. For this recipe, we will be using a Magento 1 database installation on our DigitalOcean Droplet. You may pick any of your production or Magento 1 sample data SQL dumps. We will be using a Magento 1.9.2.2 sample data SQL dump. Our database is called **magento1**.

4. Now, we need to map the database configuration files from the Magento 1 database to the Magento 2 database. Always make sure that you are using a clean database; otherwise, you can run the following command:

```
php bin/magento setup:uninstall
```

Go to /var/www/html/vendor/magento/data-migration-tool/etc/ce-to-ce and pick the correct database version mentioned in the directory. If correct, you will see two files called config.xml.dist and map.xml.dist.

5. Copy config.xml.dist to config.xml using the cp command:

```
cp config.xml.dist config.xml
```

6. Open your config.xml and look for the <source> tag (line 94). Change it accurately with the database username and password:

```
<source>
    <database host="localhost" name="magento1" user="root"
              password="mypassword"/>
</source>
<destination>
    <database host="localhost" name="magento2" user="root"
              password="mypassword"/>
</destination>
<options>
    <source_prefix>myprefix-from-magento1</source_prefix>
    <crypt_key>mycrypt-key-from-magento1</crypt_key>
</options>
```

If you are using a custom prefix in your database or you wish to use your encryption key on your Magento 2 setup, you can add this to the <options> section, as shown in the previous code.

7. Now we can start step 1 of the settings migration using the following command:

```
php bin/magento migrate:settings /var/www/magento2/vendor/magento/
data-migration-tool/etc/ce-to-ce/1.9.2.2/config.xml
```

As you can see, here we are using the 1.9.2.2 version. Depending on your version, you may change this before running the command.

The output result looks like this:

```
[2015-12-02 20:28:56] [INFO] [mode: settings] [stage: integrity
check] [step: Settings Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-02 20:28:56] [INFO] [mode: settings] [stage: integrity
check] [step: Stores Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-02 20:28:56] [INFO] [mode: settings] [stage: data migration]
[step: Settings Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-02 20:28:59] [INFO] [mode: settings] [stage: data migration]
[step: Stores Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-02 20:28:59] [INFO] [mode: settings] [stage: volume check]
[step: Stores Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-02 20:28:59] [INFO] [mode: settings] [stage: volume check]
[step: Stores Step]: Migration completed
```

8. You can check your Magento 2 system configuration backend if all updated settings are available. If so, you can continue.
9. If step 1 is correct, we continue to migrate our data to Magento 2 using the following command:

```
php bin/magento migrate:data /var/www/magento2/vendor/magento/
data-migration-tool/etc/ce-to-ce/1.9.2.2/config.xml
```

The output result looks like this:

```
[2015-12-03 19:06:28] [INFO] [mode: data][stage: integrity check][step: EAV Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:29] [INFO] [mode: data][stage: integrity check][step: Customer Attributes Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:29] [INFO] [mode: data][stage: integrity check][step: Map Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:33] [INFO] [mode: data][stage: integrity check][step: Url Rewrite Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:33] [INFO] [mode: data][stage: integrity check][step: Log Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:33] [INFO] [mode: data][stage: integrity check][step: Ratings Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:33] [INFO] [mode: data][stage: integrity check][step: ConfigurablePrices step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:33] [INFO] [mode: data][stage: integrity check][step: OrderGrids Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:33] [INFO] [mode: data][stage: integrity check][step: Tier Price Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:33] [INFO] [mode: data][stage: integrity check][step: SalesIncrement Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:33] [INFO] [mode: data][stage: setup triggers][step: Stage]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:36] [INFO] [mode: data][stage: data migration][step: EAV Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:37] [INFO] [mode: data][stage: volume check][step: EAV Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:37] [INFO] [mode: data][stage: data migration][step: Customer Attributes Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:37] [INFO] [mode: data][stage: volume check][step: Customer Attributes Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:37] [INFO] [mode: data][stage: data migration][step: Map Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:46] [INFO] [mode: data][stage: volume check][step: Map Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:46] [INFO] [mode: data][stage: data migration][step: Url Rewrite Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:46] [INFO] [mode: data][stage: volume check][step: Url Rewrite Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:46] [INFO] [mode: data][stage: data migration][step: Log Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:46] [INFO] [mode: data][stage: volume check][step: Log Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:46] [INFO] [mode: data][stage: data migration][step: Ratings Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:46] [INFO] [mode: data][stage: volume check][step: Ratings Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:46] [INFO] [mode: data][stage: data migration][step: ConfigurablePrices step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:46] [INFO] [mode: data][stage: volume check][step: ConfigurablePrices step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:46] [INFO] [mode: data][stage: data migration][step: OrderGrids Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:47] [INFO] [mode: data][stage: volume check][step: OrderGrids Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:47] [INFO] [mode: data][stage: data migration][step: Tier Price Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:47] [INFO] [mode: data][stage: volume check][step: Tier Price Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:47] [INFO] [mode: data][stage: data migration][step: SalesIncrement Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:47] [INFO] [mode: data][stage: volume check][step: SalesIncrement Step]: started
100% [=====] Remaining Time: 1 sec
[2015-12-03 19:06:47] [INFO] [mode: data][stage: volume check][step: SalesIncrement Step]: Migration completed
```

10. You can check your Magento 2 catalogs, products, orders, and customers if they are all updated. If so, you can continue.

11. Before you continue, make sure to reindex and flush your caches at once:

```
php bin/magento indexer:reindex  
php bin/magento cache:clean  
php bin/magento cache:flush
```

12. Now check the frontend and backend whether your data is available in Magento 2. If not, check the `migration.log` file located in `/var`.

13. In this recipe, we used a default Magento 1.9.2.2 setup. After the settings and data migration, we created a sales order in Magento 1. Now, using the `delta` option, we push the data to Magento 2 using the following command:

```
php bin/magento migrate:delta /var/www/magento2/vendor/magento/  
data-migration-tool/etc/ce-to-ce/1.9.2.2/config.xml
```

The output result looks like this:

```
[2015-12-03 19:54:15] [INFO] [mode: delta][stage: delta delivering][step: Customer Attributes Step]: started  
[2015-12-03 19:54:16] [INFO] [mode: delta][stage: volume check][step: Customer Attributes Step]: started  
[2015-12-03 19:54:16] [INFO] [mode: delta][stage: delta delivering][step: Map Step]: started  
[2015-12-03 19:54:16] [INFO] [mode: delta][stage: volume check][step: Map Step]: started  
[2015-12-03 19:54:16] [INFO] [mode: delta][stage: delta delivering][step: Log Step]: started  
[2015-12-03 19:54:16] [INFO] [mode: delta][stage: volume check][step: Log Step]: started  
[2015-12-03 19:54:16] [INFO] [mode: delta][stage: delta delivering][step: orderGrids Step]: started  
[2015-12-03 19:54:16] [INFO] [mode: delta][stage: volume check][step: orderGrids Step]: started  
[2015-12-03 19:54:16] [INFO] [mode: delta][stage: delta delivering][step: SalesIncrement Step]: started  
[2015-12-03 19:54:16] [INFO] [mode: delta][stage: volume check][step: SalesIncrement Step]: started  
[2015-12-03 19:54:17] [INFO] [mode: delta][stage: volume check][step: SalesIncrement Step]: Migration completed successfully  
[2015-12-03 19:54:17] [INFO] [mode: delta][stage: volume check][step: SalesIncrement Step]: Automatic restart in 5 sec. Use CTRL-C to abort  
[2015-12-03 19:54:22] [INFO] [mode: delta][stage: delta delivering][step: Customer Attributes Step]: started  
[2015-12-03 19:54:22] [INFO] [mode: delta][stage: volume check][step: Customer Attributes Step]: started  
[2015-12-03 19:54:22] [INFO] [mode: delta][stage: delta delivering][step: Map Step]: started  
[2015-12-03 19:54:22] [INFO] [mode: delta][stage: volume check][step: Map Step]: started  
[2015-12-03 19:54:22] [INFO] [mode: delta][stage: delta delivering][step: Log Step]: started  
[2015-12-03 19:54:22] [INFO] [mode: delta][stage: volume check][step: Log Step]: started  
[2015-12-03 19:54:22] [INFO] [mode: delta][stage: delta delivering][step: OrderGrids Step]: started  
[2015-12-03 19:54:22] [INFO] [mode: delta][stage: volume check][step: OrderGrids Step]: started  
[2015-12-03 19:54:22] [INFO] [mode: delta][stage: delta delivering][step: SalesIncrement Step]: started  
[2015-12-03 19:54:22] [INFO] [mode: delta][stage: volume check][step: SalesIncrement Step]: started  
[2015-12-03 19:54:22] [INFO] [mode: delta][stage: volume check][step: SalesIncrement Step]: Migration completed successfully  
[2015-12-03 19:54:22] [INFO] [mode: delta][stage: volume check][step: SalesIncrement Step]: Automatic restart in 5 sec. Use CTRL-C to abort
```

14. Now check your sales and customer data. Congratulations, you successfully migrated your database from Magento 1 to Magento 2.

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 14, you learned how to use the Magento 2 migration tool.

In step 1, we used Composer to add an additional repository for data migration. After installing all of the packages, they are available in the `bin/magento` tool. In this setup example, we used a clean Magento 1.9.x database.

In step 4, we made sure to run on a clean Magento 2 setup. Depending on your setup, go to `vendor/magento/data-migration-tool/etc` and select the correct version. Magento 2 supports the migration option for CE and EE. Once we configured the `config.xml` file with the Magento 1 database information in step 6, we were ready to go.

In step 7, we used the `bin/magento` migration setting to start the whole process. We started with the `setting` parameter and continued with the `data` and `delta` parameters in steps 7 through 13. We must not forget the reindexing and updating of our caches before using them. The `delta` parameter option can be run multiple times as it only updates the latest information, which is helpful before going live and switching to production.

There's more...

As every Magento setup is unique, migrating from Magento 1 to Magento 2 can be hard sometimes. In some situations, you may need to change your tables in the mapping configuration located in `vendor/magento/data-migration-tool/etc/ce-to-ce/<version>/map.xml.dist`.

Resetting your **setting**, **data**, and **delta** migration is easy using the `[-r | --reset]` parameter in your command. This allows you to rerun all migration scripts from the beginning.

Always check for the currently supported versions on the Magento GitHub Data Migration Tool page at the following link:

<https://github.com/magento/data-migration-tool-ce>

[ There is also an alternative Data Migration Tool available by **UberTheme** at https://github.com/ubertheme/magento2_data_migration.]

3

Enabling Performance in Magento 2

In this chapter, we will cover the basic tasks related to optimizing your performance in Magento 2. You will learn the following recipes:

- ▶ Configuring Redis for backend cache
- ▶ Configuring Memcached for session caching
- ▶ Configuring Varnish as a Full Page Cache
- ▶ Configuring Magento 2 with CloudFlare
- ▶ Configuring optimized images in Magento 2
- ▶ Configuring Magento 2 with HTTP/2
- ▶ Configuring Magento 2 performance testing

Introduction

This chapter explains one of the most important elements of Magento. From the early Magento days, performance has been a hard topic to cover. Many setups out there in the e-commerce world are performance-great, but most of the time, the majority are having issues. Magento 1 may not be the best performance platform out there.

However, now we have Magento 2, a brand new platform designed for performance. From the very first day, the main Magento developers focused on a better framework and the outcome is great. According to the latest information, Magento focused on a **Google PageSpeed** ranking of 90% or more.

In this chapter, we will dive in deeper on how to configure Redis caching and Memcached sessions. By default, Magento 2 supports Varnish, and we will manage all of the steps on how to set it up.

Serving the correct catalog or product images is very important, and will save lots of bandwidth on a desktop but most of all on a mobile device, which will have a better user experience.

As we mentioned in *Chapter 1, Installing Magento 2 on Apache and NGINX*, the new HTTP/2 protocol is a very new important element in the web server configuration. We will set up a full-force HTTP/2 configuration, including SSL.

For a high-demanding Magento 2 website serving customers all over the globe, we introduce **CloudFlare**, which is a CDN provider optimized for Magento.

Without performance testing, Magento 2 will not perform well. You will learn how to create a company-like profile, including websites, stores, catalogs, products, orders, and much more.



Throughout this recipe, you can pick your own preferred hosting setup as we did in *Chapter 1, Installing Magento 2 on Apache and NGINX*. We will be using an NGINX-based setup. The Apache setup is pretty straightforward; when needed, we will address specified configuration settings when they occur.

Configuring Redis for backend cache

Redis may be one of the best improvements since we used Memcache(d) or **Alternative PHP Cache (APC)**. For the last couple of years, Redis is available in Magento 1 and has a big performance benefit.

What is Redis and why is it important for Magento? Well, Redis is not new; its initial release dates to the beginning of 2009—almost as young as Magento 1. Redis is a key-value storage database that stores the data in-memory of your web server. Besides this, the in-memory caches are fast and also have a persistence feature that is really important when a server reboots. All caches are not flushed during a reboot and are available in-memory when the web server is up again.

In the beginning of the Magento 1 area, we used Memcache(d) or APC, which worked very well but not as well as Redis. In Magento 1, Redis was used for a backend cache and session storage most of the time. Some websites also used it as a **Full Page Cache (FPC)** storage.

One other great advantage of Redis is that it has multiple database containers, one for the default cache and the other for the FPC. Although the Redis performance is better in a lot of cases, it is not the *Holy Grail*. There are drawbacks to its architecture.

Getting ready

For this recipe, we will use a Droplet created in *Chapter 2, Magento 2 System Tools*, at DigitalOcean, <https://www.digitalocean.com/>. We will be using NGINX, PHP-FPM, and a Composer-based setup, including sample data connected to a Redis server. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to create a Magento 2 Redis setup. The following steps will guide you through this:

1. First, we need to install the Redis server and Redis PHP client before we can connect it to Magento. Follow the next step on the shell:

```
cd /opt
wget http://download.redis.io/releases/redis-3.0.5.tar.gz
tar xzf redis-3.0.5.tar.gz
cd redis-3.0.5
make && make install
```

Change the version number of the current one if needed.

2. Go to the /opt/redis-3.0.5/utils directory and run the following script:
`./install-server.sh`
3. Commit to the following questions: (Press *Enter* to all of them; the default is just fine.)

```
Welcome to the redis service installer
This script will help you easily set up a running redis server
```

```
Please select the redis port for this instance: [6379]
Selecting default: 6379

Please select the redis config file name [/etc/redis/6379.conf]
Selected default - /etc/redis/6379.conf

Please select the redis log file name [/var/log/redis_6379.log]
Selected default - /var/log/redis_6379.log

Please select the data directory for this instance [/var/lib/
redis/6379]
Selected default - /var/lib/redis/6379

Please select the redis executable path [/usr/local/bin/redis-
server]
```

```
Selected config:  
Port : 6379  
Config file : /etc/redis/6379.conf  
Log file : /var/log/redis_6379.log  
Data dir : /var/lib/redis/6379  
Executable : /usr/local/bin/redis-server  
Cli Executable : /usr/local/bin/redis-cli  
Is this ok? Then press ENTER to go on or Ctrl-C to abort.  
Copied /tmp/6379.conf => /etc/init.d/redis_6379  
Installing service...  
Success!  
Starting Redis server...  
Installation successful!
```

4. Now let's test our Redis server using the following command:

```
redis-cli -version  
service redis_6379 status  
netstat -anp | grep redis
```

As you can see, the Redis server is running under port 6379.

5. The next important element is installing a PHP module that can communicate with the Redis server. We will use PHP Redis here (<https://github.com/phpredis/phpredis>).

Use the following command to install PHP Redis:

```
cd /opt  
git clone https://github.com/phpredis/phpredis.git  
cd phpredis  
phpize  
.configure  
make && make install
```

- Now, we need to let PHP know there is a Redis extension available that we can use. Run the following command:

```
echo "extension=redis.so" | sudo tee /etc/php5/mods-available/  
redis.ini
```

Depending on whether you are using PHP 5 or PHP 7, you may want to change the PHP path.

- Now we need to link the Redis PHP extension to PHP-FPM and PHP CLI. Run the following commands:

```
cd /  
ln -s /etc/php5/mods-available/redis.ini /etc/php5/fpm/conf.d/20-  
redis.ini  
ln -s /etc/php5/mods-available/redis.ini /etc/php5/cli/conf.d/20-  
redis.ini
```

- If everything is correct, we can restart the PHP-FPM server to activate the Redis PHP extension. Run the following command:

```
service php5-fpm restart
```

- To make sure that the Redis PHP and Redis server are running together, we can use the following command:

```
php -r "if (new Redis() == true){ echo \"\r\n OK \r\n\"; }"
```

By default, creating a `phpinfo.php` page in the root directory in Magento 2 will not work. First, you need to create the `phpinfo.php` file in the `/pub` directory. Then, you need to change the NGINX configuration (`nginx.conf.sample`) from location `~ (index|get|static|report|404|503)\.php$ {` to location `~ (index|get|static|report|404|503|phpinfo)\.php$ {`, which is located at the bottom of the file. In Apache, we don't have an issue like this; it works by default.

 Use `phpinfo.php` wisely on a production environment. Sharing this information on a production website is not advised and could expose your security risks.

- Congratulations, you just finished the Redis server and PHP Redis setup. Now let's continue with the Magento 2 part.

11. Open the `env.php` file in Magento 2 located at `/app/etc` and add the following code at the top:



```
<?php
return array (
    'cache' =>
        array (
            'frontend' =>
                array (
                    'default' =>
                        array (
                            'backend' => 'Cm_Cache_Backend_Redis',
                            'backend_options' =>
                                array (
                                    'server' => '127.0.0.1',
                                    'port' => '6379',
                                    'persistent' => '',
                                    'database' => '0',
                                    'force_standalone' => '0',
                                    'connect_retries' => '1',
                                    'read_timeout' => '10',
                                    'automatic_cleaning_factor' => '0',
                                    'compress_data' => '1',
                                    'compress_tags' => '1',
                                    'compress_threshold' => '20480',
                                    'compression_lib' => 'gzip',
                                ),
                ),
            'page_cache' =>
                array (
                    'backend' => 'Cm_Cache_Backend_Redis',
                    'backend_options' =>
                        array (
                            'server' => '127.0.0.1',
                            'port' => '6379',
                            'persistent' => '',
                            'database' => '1',
                            'force_standalone' => '0',
                            'connect_retries' => '1',
                            'read_timeout' => '10',
                            'automatic_cleaning_factor' => '0',
                            'compress_data' => '0',
                            'compress_tags' => '1',
                            'compress_threshold' => '20480',
                            'compression_lib' => 'gzip',
                        ),
                ),
        ),
    'backend' =>
        array (
            'frontName' => 'admin',
        ),
    'install' =>
```

12. As you can see, we are using two databases—one for the default cache and one for **page_cache** (Full Page Cache). Now, save your file and remove any cache in /var/page_cache and /var/cache. Let's open up your browser and refresh your website.

If everything is configured correctly in the env.php file, you should not get any errors and the ar/page_cache and var/cache directories should be empty.

13. To check how many keys Redis received, we can run the following command from the shell:

```
redis-cli
```

On the prompt, continue with INFO; this will give you a list of the following details:

```
root@mage2cookbook:/var/www/magento2/var/cache# redis-cli INFO
# Server
redis_version:3.0.5
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:76f6ee2908f61611
redis_mode:standalone
os:Linux 3.13.0-57-generic x86_64
arch_bits:64
multiplexing_api:epoll
gcc_version:4.9.2
process_id:9713
run_id:8eb836577d6f3ec4a2d1a179ee99e538543589e6
tcp_port:6379
uptime_in_seconds:86922
uptime_in_days:1
hz:10
lru_clock:6845253
config_file:/etc/redis/6379.conf

# Clients
connected_clients:1
client_longest_output_list:0
client_biggest_input_buf:0
blocked_clients:0

# Memory
used_memory:5661432
used_memory_human:5.40M
used_memory_rss:11952128
used_memory_peak:7778328
used_memory_peak_human:7.42M
used_memory_lua:39936
mem_fragmentation_ratio:2.11
mem_allocator:jemalloc-3.6.0
```

To close the Redis terminal, use exit.

14. Congratulations, you just finished configuring the Redis server and PHP Redis with Magento 2.

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 13, we installed a Redis server and configured Magento 2 to store the backend cache.

In step 1, we installed Redis from source and compiled the code. This version is more stable than the default one available in Ubuntu. After compiling the code, we are able to use an install script to create a working setup running on port 6379.

After installing and testing the code in steps 3 and 4, we start installing the PHP Redis module. This code is pulled from GitHub and compiled from source.

In step 6, we created a `redis.ini` file, which is linked in step 7 with the correct PHP module directory. Before we can test it, we need to restart the PHP-FPM server and use a simple PHP command to test if everything is working fine.

In step 11, we added an additional piece of code to the `env.php` file, which will tell Magento 2 to store all of the cache in Redis as of now.

There's more...

If you are interested in monitoring your Redis server, the next step is interesting. Clone **PHPRedMin** (<https://github.com/sasanrose/phpredmin>) in your Magento 2 root directory, `/var/www/html`. Make sure to change the ownership to `www-data` for the owner and group.

Go to your `/var/www/html/pub` directory and create a symbolic link using the following command:

```
ln -s ../phpredmin/public phpredmin
```

Chown the ownership of the symbolic link with the following command:

```
chown -h www-data:www-data phpredmin
```

Go to to your NGINX configuration directory, /etc/nginx/conf.d, open the default.conf file, and including the following content below error_log:

```
location ~ ^/phpredmin/.+\.php {
    fastcgi_split_path_info ^(.+\.php)(/.+)$;

    set $fsn /index.php;
    if (-f $document_root$fastcgi_script_name) {
        set $fsn $fastcgi_script_name;
    }

    # php5-fpm
    fastcgi_pass fastcgi_backend;
    fastcgi_index index.php;

    fastcgi_param SCRIPT_FILENAME $document_root$fsn;
    fastcgi_param PATH_INFO $fastcgi_path_info;
    fastcgi_param PATH_TRANSLATED $document_root$fsn;

    include fastcgi_params;
}
```

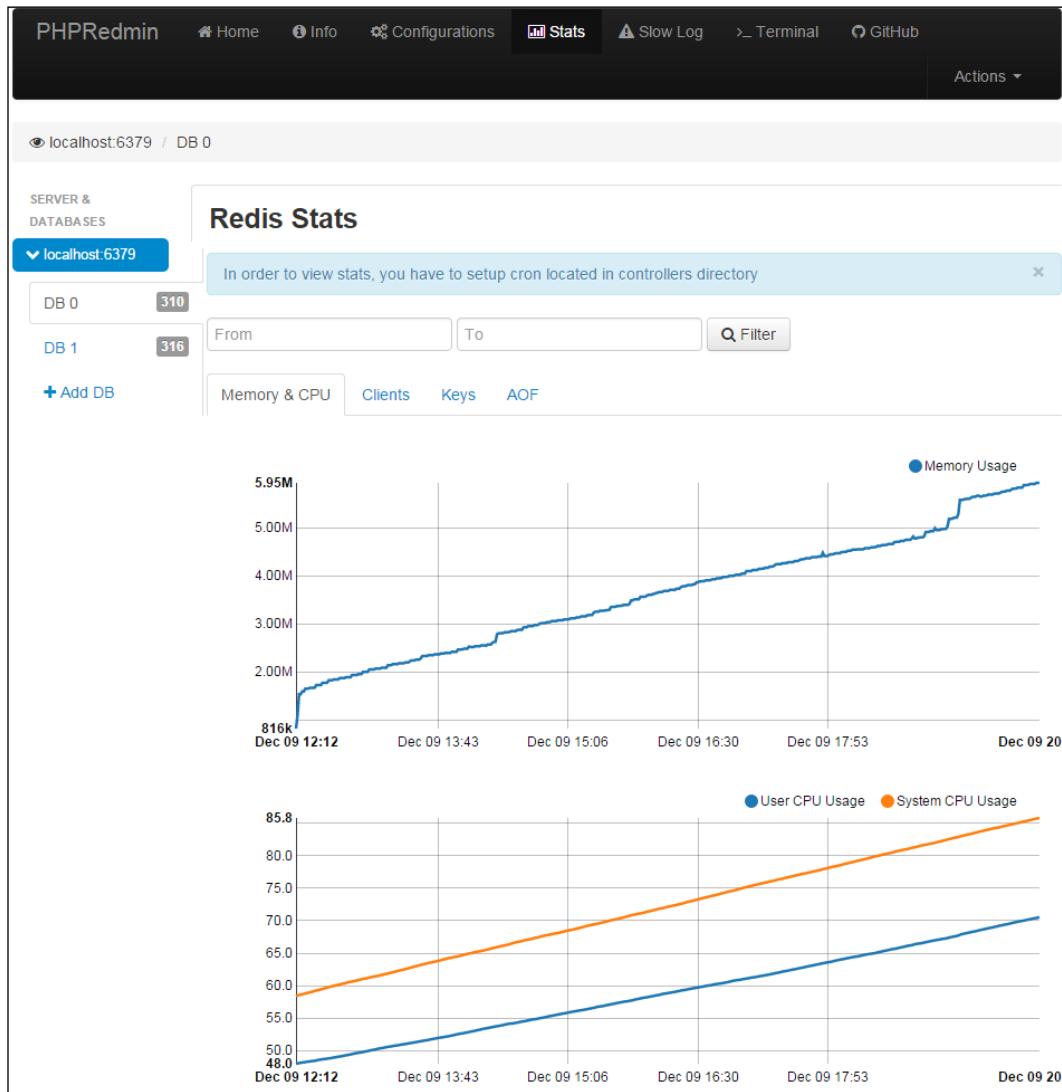
Now save and restart your NGINX server with `service nginx restart`.

Before we can continue, we need to add a cronjob rule to gather our Redis data and show it in PHPRedMin. Add the following rule to your crontab.

Open crontab using `crontab -e`:

```
* * * * * cd /var/www/html/pub/phpredmin && php index.php cron/index
```

Open your browser and surf to <http://yourdomain.com/phpredmin>, and press **Stats** in the top menu. Now you should see the following information:



[ On a production site, you will want to add an IP block so that only you can gain access.]

Configuring Memcached for session caching

As Magento 2 does not support Redis session caching from the beginning, we need to use Memcached instead. Memcached has been around for a long time and was used in Magento 1, since the beginning, as backend and session caching.

Memcached is a distributed memory caching system. It is a flexible in-memory storage container to cache data. As the session handler in the PHP Redis does not support session locking, we use Memcached instead. Keep in mind that Memcached is not persisted, so after restarting the server or daemon, all the data is gone. This could have an impact on a production environment—lost sessions or baskets.

Getting ready

For this recipe, we will use a Droplet created in *Chapter 2, Magento 2 System Tools*, at DigitalOcean, <https://www.digitalocean.com/>. We will be using NGINX, PHP-FPM, and a Composer-based setup including sample data connected to a Memcached server. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to create a Magento 2 Memcached setup. The following steps will guide you through this:

1. First, we need to install the Memcached server and Memcached PHP client before we can connect it to Magento. Follow the next step on the shell:

```
apt-get install -y libevent-dev  
apt-get install -y memcached
```

2. Now let's test our Memcached server using the following command:

```
memcached -V  
service memcached status  
netstat -anp | grep memcached
```

As you can see, the Memcached server is running under port 11211.

3. The next important element is installing a PHP module that can communicate with the Memcached server. We will be using the PHP Memcached extension and not the PHP Memcache extension (without the *d* at the end). The PHP Memcached (*d*) will be supporting PHP 7.

Use the following command to install PHP Memcached:

```
apt-get install php5-memcached
```

We can also use the following command to install it:

```
apt-get install php-memcached (php7)
```

- Now let's check whether PHP has the correct Memcached extension installed. Run the following command:

```
cat /etc/php5/mods-available/memcached.ini
```

Now you should see the Memcached extension called `extension=memcached.so`.

Depending on whether you are using PHP 5 or PHP 7, you may want to change the PHP path.

- If everything is correct, we can restart the PHP-FPM server to activate the Memcached PHP extension. Run the following command:

```
service php5-fpm restart
```

- To make sure that the Memcached PHP and Memcached servers are running together, we can check using the following command:

```
php -r "if (new Memcached() == true){ echo \"\r\n OK \r\n\"; }"
```

It can also be checked using the following command:

```
echo "stats settings" | nc localhost 11211
```

By default, creating a `phpinfo.php` page in the root directory in Magento 2 will not work. First, you need to create the `phpinfo.php` file in the `/pub` directory. Then, you need to change the NGINX configuration from `location ~ (index|get|static|report|404|503)\.php$ {` to `location ~ (index|get|static|report|404|503|phpinfo)\.php$ {`, which is located at the bottom of the file. In Apache, we don't have an issue like this; it works by default.

 Use `phpinfo.php` wisely on a production environment. Sharing this information on a production website is not advised and could expose your security risks.

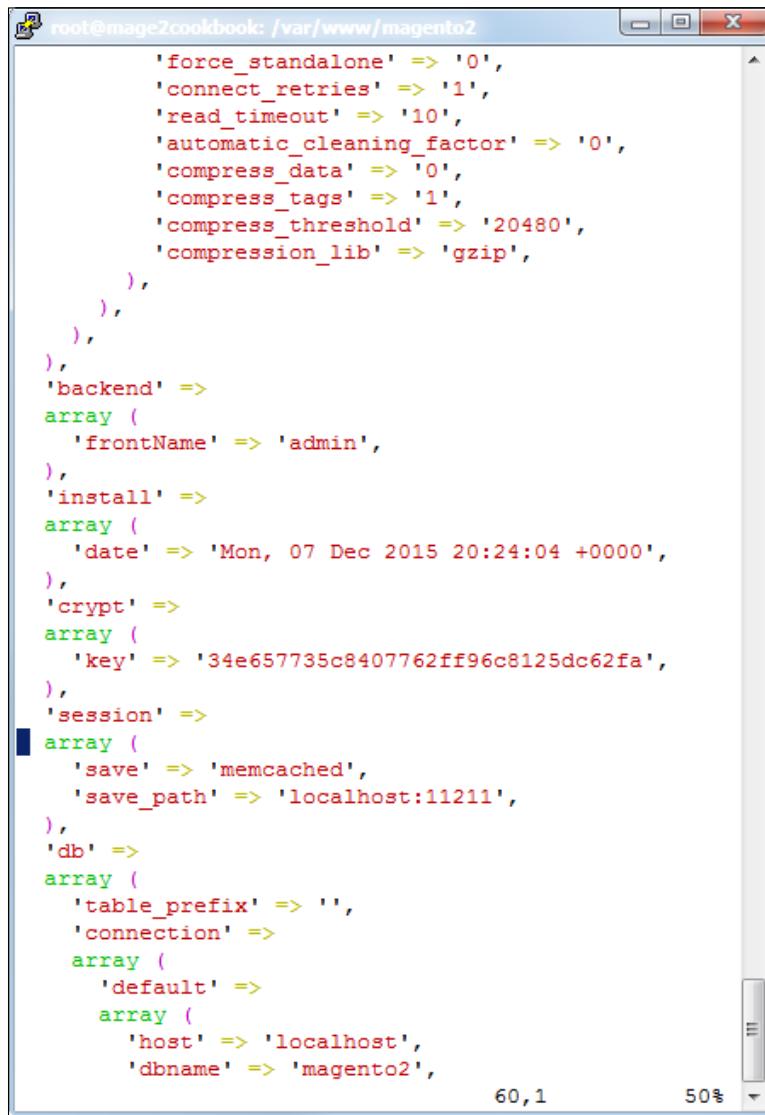
- Congratulations, you just finished the Memcached server and PHP Memcached setup. Now let's continue with the Magento 2 part.
- Open the `env.php` file in Magento 2 located at `/app/etc` and change the following code in the session section:

```
'session' =>  
    array (  
        'save' => 'files',  
    ),
```

Change the preceding code to the following:

```
'session' =>
    array (
        'save' => 'memcached',
        'save_path' => 'localhost:11211'
    ),
```

This is shown in the following screenshot:



The screenshot shows a terminal window titled "root@mage2cookbook: /var/www/magento2". The window displays a block of PHP-style configuration code. The code defines several arrays for different system components. The 'session' component is explicitly set to use 'memcached' as the save method and 'localhost:11211' as the save path. Other components like 'crypt' and 'db' are also defined with their respective settings.

```
'force_standalone' => '0',
'connect_retries' => '1',
'read_timeout' => '10',
'automatic_cleaning_factor' => '0',
'compress_data' => '0',
'compress_tags' => '1',
'compress_threshold' => '20480',
'compression_lib' => 'gzip',
),
),
),
),
),
'backend' =>
array (
    'frontName' => 'admin',
),
'install' =>
array (
    'date' => 'Mon, 07 Dec 2015 20:24:04 +0000',
),
'crypt' =>
array (
    'key' => '34e657735c8407762ff96c8125dc62fa',
),
'session' =>
array (
    'save' => 'memcached',
    'save_path' => 'localhost:11211',
),
'db' =>
array (
    'table_prefix' => '',
    'connection' =>
        array (
            'default' =>
                array (
                    'host' => 'localhost',
                    'dbname' => 'magento2',

```

9. Now save your file and remove any caches and sessions in `var/page_cache`, `var/cache` and `var/session`. Restart your website using the following command:
`service nginx restart && service php-fpm restart`

Let's open up your browser and refresh your website.

If everything is configured correctly in the `env.php` file, you should not get any errors and the `/var/session` directory should be empty.

10. To check how many keys Memcached received, we can run the following command from the shell:

```
echo "stats items" | nc localhost 11211
```

11. Congratulations, you just finished configuring the Memcached server and PHP Memcached with Magento 2.

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 11, we installed a Memcached server and configured Magento 2 to store the sessions.

In steps 1 through 3, we installed the default Ubuntu Memcached server and PHP Memcached client modules. Depending on whether we are using PHP 5 or 7, we pick a different one. Before this, we make sure to restart the PHP-FPM server and test if everything is working correctly.

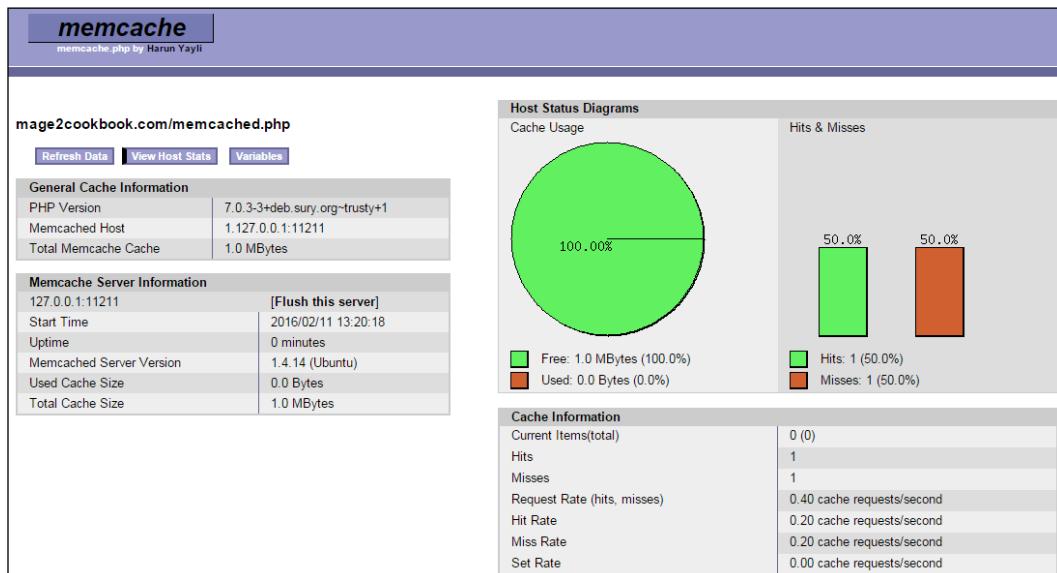
In step 8, we added an additional piece of code to the `env.php` file, which will tell Magento 2 to store all of the sessions in Memcached as of now.

There's more...

If you are interested in monitoring your Memcached server, the next step is interesting. Clone the `memcached.php` file from the GitHub Gist (<https://gist.github.com/raybogman/b8b7b4d21bf34ed9dd76>) in your Magento 2 root directory, `/var/www/html/pub`. Make sure to change the ownership to `www-data` for the owner and group.

By default, creating a `memcached.php` page in the root directory in Magento 2 will not work. First, you need to store the `memcached.php` file in the `/pub` directory. Then, you need to change the NGINX configuration from `location ~ (index|get|static|repo rt|404|503)\.php$ {` to `location ~ (index|get|static|report|404|503|mem cached)\.php$ {`, which is located at the bottom of the file. In Apache, we don't have an issue like this; it works by default.

Once installed correctly, the page will look as follows:



Note that this Memcached viewer is an outdated version created in 2008 by Harun Yayli and is not maintained anymore. Use it wisely.

As an alternative, you can also use <https://github.com/clickalicious/phpMemAdmin>.

Configuring Varnish as the Full Page Cache

Varnish may be one of the most interesting elements described in this book, besides Magento 2, of course. What is Varnish and why is it that important? Well, Varnish is like a Ferrari, very fast on the track but hard to maintain or tune. In technical terms, Varnish is an HTTP accelerator designed for heavy websites. Magento users love fast websites.

By default, Varnish support is now included in Magento 2. In Magento 1, we commonly used **Turpentine** by **Nexcess** (<https://github.com/nexcess/magento-turpentine>). The configuration of Varnish is not for the faint-hearted. Varnish includes a **Varnish Configuration Language (VCL)** file, which holds all the elements to be cached or not.

Setting up a Varnish server may be simple; configuring the VCL is not. Magento 2 provides a default VCL file that works out of the box, but be aware of any custom extensions or layout updates. Any customization has to be added manually in the VCL file before Varnish can cache them.



By default, Varnish does not support HTTPS; you may need an SSL proxy such as NGINX or Apache to do this.



Getting ready

For this recipe, we will use a Droplet created in *Chapter 2, Magento 2 System Tools*, at DigitalOcean, <https://www.digitalocean.com/>. We will be using NGINX, PHP-FPM, and a Composer-based setup including sample data connected to a Varnish server. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to create a Magento 2 Varnish setup. The following steps will guide you through this:

1. First, we need to install the Varnish server before we can connect it to Magento. Follow the next step on the shell:

```
apt-get install -y apt-transport-https
```

By default, all current Ubuntu versions support apt-transport-https.

2. Let's create a new Varnish repository using the following code:

```
echo "deb https://repo.varnish-cache.org/ubuntu/ trusty  
varnish-4.1" | sudo tee -a /etc/apt/sources.list.d/varnish-cache.  
list
```

3. Add the Varnish key to our system using the following command:

```
curl https://repo.varnish-cache.org/GPG-key.txt | apt-key add -
```

4. Now we can update our server so that the Varnish software is made available for use. Run the following command:

```
apt-get update && apt-get install -y varnish  
service varnish start
```

5. Now let's test our Varnish server using the following command:

```
varnishd -v  
service varnish status  
netstat -anp | grep varnish
```

As you can see, the Varnish server is running under ports 6081 and 6082.

6. Before we can use Varnish as a frontend server, we need to change the NGINX or Apache port. In NGINX, we use the following command:

```
sed -i 's/80/8080/' /etc/nginx/conf.d/default.conf
```

We are using port 8080 as an internal port.

Your configuration file could look as follows:

```
server {  
    listen 8080;  
  
    server_name yourdomain.com;
```

7. Now let's update the Varnish server. We need to change the default port 6081 to 80. Use the following command to change the /etc/default/varnish file:

```
sed -i 's/6081/80/' /etc/default/varnish
```

8. By default, there is a small bug in the Ubuntu system that is using the new systemd setup. The systemd servers will not update their configuration script after a restart or reboot. Let's update this manually using the following command:

```
sed -i 's/6081/80/' /lib/systemd/system/varnish.service
```

Update the systemd process with the following code:

```
systemctl daemon-reload
```

9. Next, we restart the Varnish and NGINX (or Apache) servers. Run the following command:

```
service varnish restart && service nginx restart
```

10. Now you can check whether Varnish and NGINX are running on the correct port. Use the following command:

```
netstat -upnlt | egrep 'varnish|nginx'
```

11. Congratulations, Varnish is running on port 80 and NGINX is running on port 8080.

12. Now update Magento. Log in to the backend of your Magento site, navigate to **Stores** | **Configuration** | **Advanced** | **System** | **Full Page Cache**, and select **Varnish Caching** in the drop-down menu. If you are running Varnish on the same server as your website, you are okay with the localhost and backend port 8080. It's better to install Varnish on a single dedicated server. You may need to change these settings correctly. Export the correct VCL for the Varnish file. As we are using Varnish 4, we will download it.

Always make sure that Magento is running in the developer mode when setting up Varnish. When ready to launch, we can switch to the production mode:

The screenshot shows the 'ADVANCED' section of the Magento 2 Admin Panel. Under 'Full Page Cache', the 'Caching Application' is set to 'Varnish Caching' with a TTL of 86400 seconds. In the 'Varnish Configuration' section, the 'Access list' is set to 'localhost'. The 'Backend host' is also set to 'localhost'. The 'Backend port' is set to 8080. There are two buttons at the bottom: 'Export Configuration' (disabled) and 'Export VCL for Varnish 3'.

Scheduled Backup Settings

Full Page Cache

Caching Application: Varnish Caching [GLOBAL]

TTL for public content: 86400 [GLOBAL]

Public content cache lifetime in seconds. If field is empty default value 86400 will be saved.

Varnish Configuration

Access list: localhost [GLOBAL]

IPs access list separated with ',' that can purge Varnish configuration for config file generation. If field is empty default value localhost will be saved.

Backend host: localhost [GLOBAL]

Specify backend host for config file generation. If field is empty default value localhost will be saved.

Backend port: 8080 [GLOBAL]

Specify backend port for config file generation. If field is empty default value 8080 will be saved.

Export Configuration (disabled)

Export VCL for Varnish 3 [GLOBAL]

Export VCL for Varnish 4 [GLOBAL]

13. Copy the file to the server and replace the current /etc/varnish/default.vcl file. Now open the file and change backend default to the following:

```
backend default {  
    .host = "127.0.0.1";  
    .port = "8080";  
}
```

14. Now let's restart the Varnish server to use the current VCL setup and flush our Magento cache:

```
service varnish restart  
php bin/magento cache:clean  
php bin/magento cache:flush
```

15. Using the Magento 2 developer mode is necessary; it will show us an **X-Magento-Cache-Debug** notice. Use the following command to see if we have received a cache **HIT**:

```
curl -I http://yourdomain.com
```

The output of this command should be as follows:

```
root@mage2cookbook:~# curl -I http://mage2cookbook.com  
HTTP/1.1 200 OK  
Date: Mon, 14 Dec 2015 19:15:40 GMT  
Content-Type: text/html; charset=UTF-8  
X-Frame-Options: SAMEORIGIN  
X-Content-Type-Options: nosniff  
X-XSS-Protection: 1; mode=block  
X-Magento-Cache-Control: max-age=86400, public, s-maxage=86400  
Pragma: no-cache  
Expires: -1  
Cache-Control: no-store, no-cache, must-revalidate, max-age=0  
Vary: Accept-Encoding  
Age: 1127  
X-Magento-Cache-Debug: HIT  
Accept-Ranges: bytes  
Connection: keep-alive
```

16. Congratulations, you just finished configuring a Varnish server with Magento 2.

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 16, we installed and configured Varnish to speed up the full page caching.

In steps 1 through 4, we added the official repository to our system and installed Varnish.

In steps 6 and 7, we changed the NGINX port to 8080 instead of 80, and the Varnish port to 80. Now, Varnish will be our gatekeeper after restarting the NGINX and Varnish servers.

In step 12, we told Magento to start communicating with the Varnish server so that all frontend cacheable data is stored here.

There's more...

The current lifetime of the cache is 86,400 seconds, which is one day. So, installing a cache warmer will speed up your pages after an automatic cache flush by Magento. Always keep in mind that by default, all the pages are cold (without a cache hit) and the first GET (page view) can take longer. Varnish needs to build up the cache before customers can benefit from it.

Check out the following Varnish tools to monitor all the incoming data live:

varnishstat

The output of this command will be as follows:

```
root@mage2cookbook: ~
Uptime mgt: 1+00:50:08                                Hitrate n:      10    34    34
Uptime child: 1+00:50:08                                avg(n): 0.0716  0.0532  0.0532

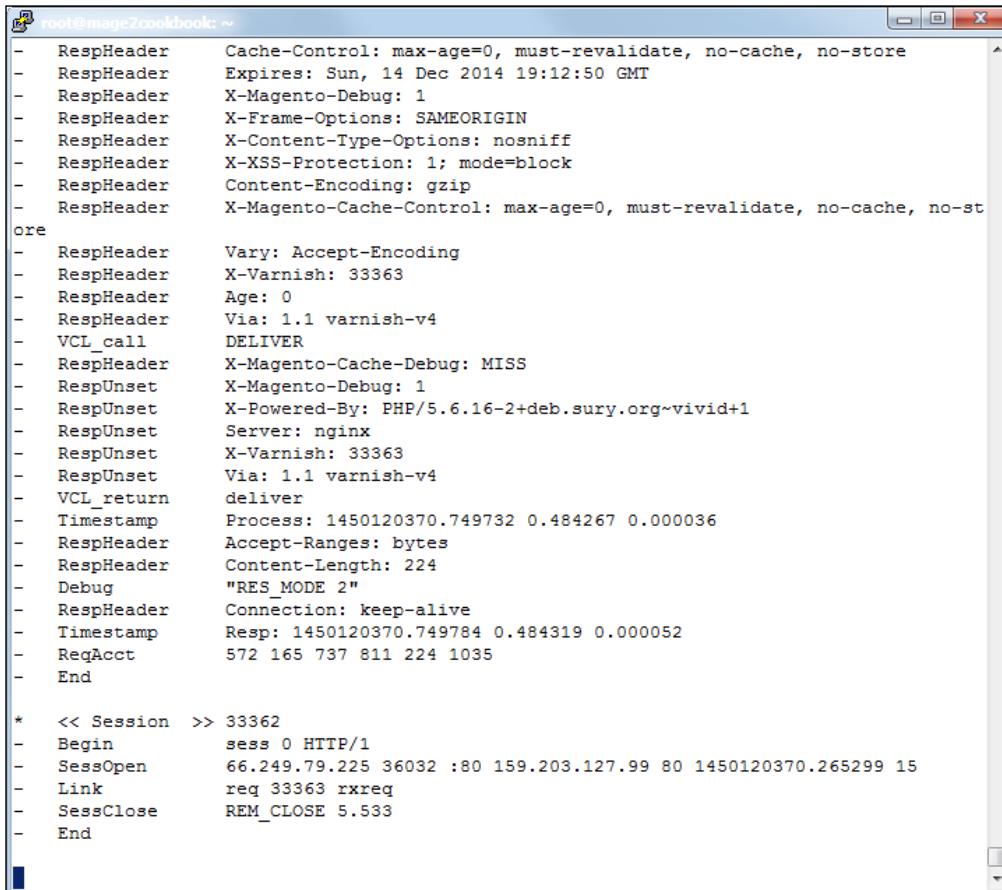
NAME          CURRENT   CHANGE   AVERAGE   AVG_10
MAIN.uptime   1+00:50:08
MAIN.sess_conn        964    0.00     .         0.49
MAIN.client_req_400       1    0.00     .         0.00
MAIN.client_req        4170    0.00     .        14.63
MAIN.cache_hit        2988    0.00     .        11.94
MAIN.cache_miss       1004    0.00     .         2.69
MAIN.backend_reuse      1192    0.00     .         0.94
MAIN.backend_recycle     1704    0.00     .         1.01
MAIN.fetch_length       146    0.00     .         0.16
MAIN.fetch_chunked      1535    0.00     .         0.85
MAIN.fetch_304          23    0.00     .         0.00
MAIN.pools             2    0.00     .         2.00
MAIN.threads           200   0.00     .        200.00
MAIN.threads_created     200   0.00     .         0.00
MAIN.busy_sleep        1575    0.00     .         0.00
MAIN.n_object          691    0.00     .       683.27
MAIN.n_objectcore       700   0.00     .       676.41
MAIN.n_objecthead       706   0.00     .       682.97
MAIN.n_backend          1    0.00     .         1.00
MAIN.n_expired          313   0.00     .       311.79
MAIN.s_sess             964   0.00     .         0.49
MAIN.s_req              4170   0.00     .        14.63
MAIN.s_pipe              1    0.00     .         0.00
MAIN.s_pass              678   0.00     .         0.02
MAIN.s_fetch             1682   0.00     .         2.70
MAIN.s_req_hdrbytes     3.06M  0.00   35.00    17.07K
MAIN.s_req_bodybytes    74.59K 0.00     .         0.00
MAIN.s_resp_hdrbytes    1.79M  0.00   20.00    5.17K
MAIN.s_resp_bodybytes   18.13M 0.00  212.00   19.72K
MAIN.s_pipe_hdrbytes     98    0.00     .         0.00
vvv MAIN.uptime
Child process uptime:
How long the child process has been running.

INFO 1-30/57
```

Now, let's execute the following command:

```
varnishlog
```

The output of this command will be as follows:



```
root@mage2cookbook: ~
- RespHeader Cache-Control: max-age=0, must-revalidate, no-cache, no-store
- RespHeader Expires: Sun, 14 Dec 2014 19:12:50 GMT
- RespHeader X-Magento-Debug: 1
- RespHeader X-Frame-Options: SAMEORIGIN
- RespHeader X-Content-Type-Options: nosniff
- RespHeader X-XSS-Protection: 1; mode=block
- RespHeader Content-Encoding: gzip
- RespHeader X-Magento-Cache-Control: max-age=0, must-revalidate, no-cache, no-store
ore
- RespHeader Vary: Accept-Encoding
- RespHeader X-Varnish: 33363
- RespHeader Age: 0
- RespHeader Via: 1.1 varnish-v4
- VCL_call DELIVER
- RespHeader X-Magento-Cache-Debug: MISS
- RespUnset X-Magento-Debug: 1
- RespUnset X-Powered-By: PHP/5.6.16-2+deb.sury.org~vivid+1
- RespUnset Server: nginx
- RespUnset X-Varnish: 33363
- RespUnset Via: 1.1 varnish-v4
- VCL_return deliver
- Timestamp Process: 1450120370.749732 0.484267 0.000036
- RespHeader Accept-Ranges: bytes
- RespHeader Content-Length: 224
- Debug "RES_MODE 2"
- RespHeader Connection: keep-alive
- Timestamp Resp: 1450120370.749784 0.484319 0.000052
- ReqAcct 572 165 737 811 224 1035
- End

* << Session >> 33363
- Begin sess 0 HTTP/1
- SessOpen 66.249.79.225 36032 :80 159.203.127.99 80 1450120370.265299 15
- Link req 33363 rxreq
- SessClose REM_CLOSE 5.533
- End
```

Configuring Magento 2 with CloudFlare

Are you managing an international-based brand-serving customer all over the globe? Then, using a **Content Delivery Network (CDN)** is the best idea. CDNs are a well-known technique to manage high-traffic websites. It is commonly used to distribute static assets such as images, CSS, and JavaScript as quickly as possible to the nearest location of the customers, which decreases the download times of the website.

The modern CDNs have much more to offer than just serving the assets to the customer. Currently, they improve the user experience with optimized HTML output, merging and deferring JavaScript, TCP optimization, and much more. Basic or advanced security is also top-of-mind, such as (D)DoS protection, SSL, **Web Application Firewall (WAF)**, and much more.

 Before using a CDN on production, test which CDN provider fits best for your purpose. Make sure that the POP locations that they serve match your customer locations.

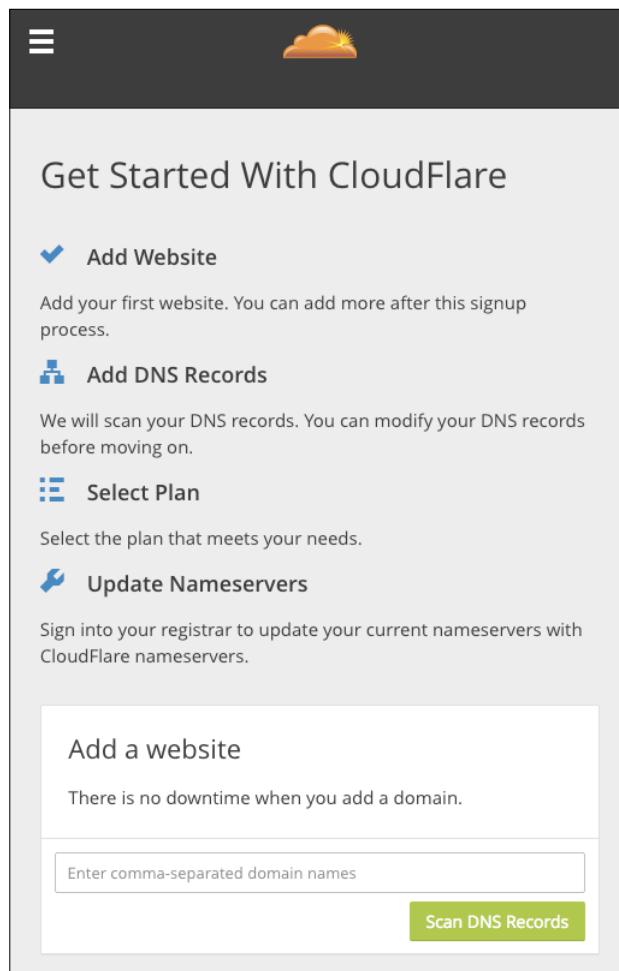
Getting ready

For this recipe, we will use a Droplet created in *Chapter 2, Magento 2 System Tools*, at DigitalOcean, <https://www.digitalocean.com/>. We will be using NGINX, PHP-FPM, and a Composer-based setup including sample data connected to the CloudFlare CDN. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to create a Magento 2 Varnish setup. The following steps will guide you through this:

1. First, we need to create an account at CloudFlare. Go to <https://www.cloudflare.com/a/sign-up> and complete the supplied form.
2. Now add a website URL. Choose the default URL of your Magento website, (We can add more URLs under the same CloudFlare account later.) and press **Scan DNS Records**:



The screenshot shows the 'Get Started With CloudFlare' page. At the top right is a yellow cloud icon with a sunburst. Below it, the title 'Get Started With CloudFlare' is displayed. There are four main steps listed:

- Add Website**: Adds your first website. You can add more after this signup process.
- Add DNS Records**: We will scan your DNS records. You can modify your DNS records before moving on.
- Select Plan**: Select the plan that meets your needs.
- Update Nameservers**: Sign into your registrar to update your current nameservers with CloudFlare nameservers.

Below these steps is a large callout box titled 'Add a website'. It contains the text: 'There is no downtime when you add a domain.' followed by a text input field labeled 'Enter comma-separated domain names' and a green button labeled 'Scan DNS Records'.

- Once completed, we need to verify that all of our DNS records are listed. This step is really important so make sure to check your current DNS settings and compare or add them to your CloudFlare DNS setup. By default, CloudFlare cannot match all the DNS records automatically.

Changing your records in this screen will not change anything in production yet. We still need to adjust the primary and secondary **Nameservers** before everything works. We will do this as shown in the following screenshot:

The screenshot shows the Cloudflare DNS Records interface for the domain `yourdomain.com`. The main heading is "Verify That All Of Your DNS Records Are Listed Below". A sub-section titled "DNS Records For `yourdomain.com`" contains instructions about Cloudflare's traffic routing and two warning icons:

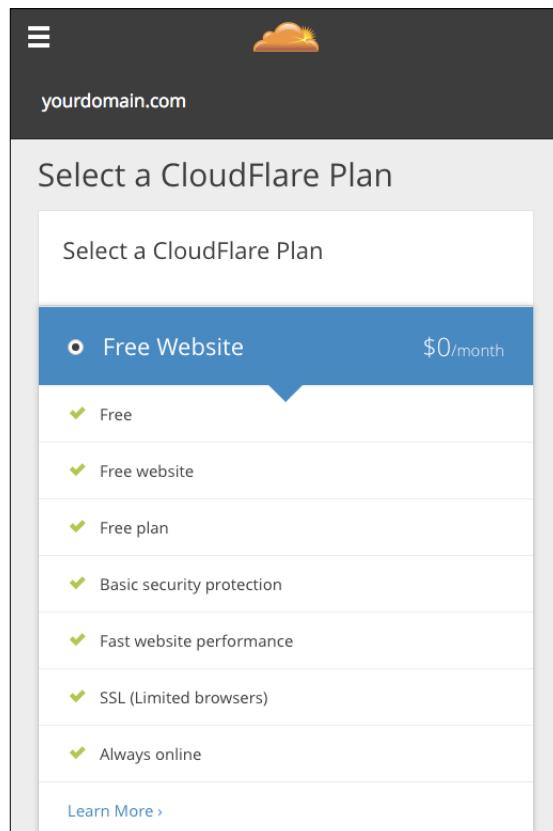
- An orange exclamation mark icon: "An A, AAAA, CNAME, or MX record is pointed to your origin server exposing your origin IP address."
- An orange exclamation mark icon: "An A, AAAA or CNAME record was not found pointing to the root domain. The `bogman.info` domain will not resolve."

Below these, there are two sections: "On CloudFlare" (with a cloud icon) and "Off CloudFlare" (with a cloud icon). Each section has a descriptive text and a status indicator.

- On CloudFlare**: "Traffic will be accelerated and protected by CloudFlare"
- Off CloudFlare**: "Traffic will bypass CloudFlare's network"

At the bottom, there is a search bar labeled "Search DNS records" and a green "Add Record" button.

- Choose your CloudFlare plan. Let's start with the **Free Website** plan. In a production environment, upgrading to a **Pro** or **Business** account is simple; just complete the billing form and you are all set. All new features will be available on the fly and ready to use:



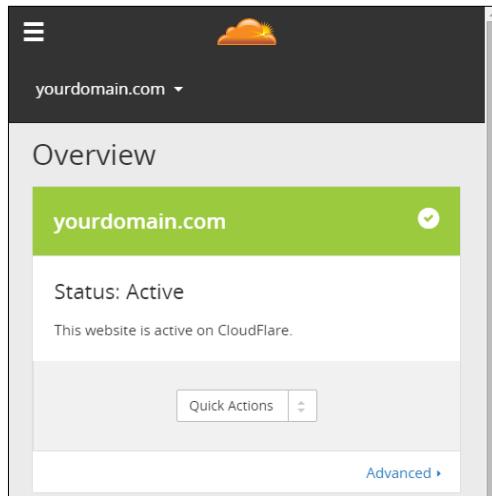
- Now we need to update our Nameservers. CloudFlare will list the Nameservers that we need to complete the last step.

Depending on your current DNS provider, this could be a simple or hard step. Changing the Nameservers is not always allowed by your provider.

 When ordering a new domain, check whether your provider allows you to change the Nameservers. Choosing the correct domain provider is not always a simple job.

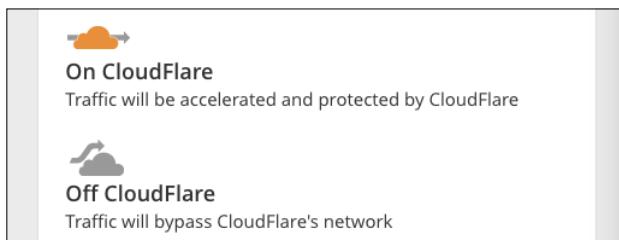
- After changing the Nameservers, we need to wait a maximum of 24 hours. The time depends on how quickly your current DNS provider updates them.

You can check your e-mail or refresh the CloudFlare dashboard to check whether your domain is **Active**:



- Let's go to the DNS dashboard and check whether our domain name is served using the CloudFlare accelerated and protection technique.

Once the cloud is orange, including an arrow passing through, then you are connected. Click on the cloud icon to change it:



- Let's check whether the DNS server is serving the correct records and CloudFlare is working. Run the following command on the shell of your current server:

```
dig yourdomain.com NS +short
```

The output looks as follows:

```
root@mage2cookbook:~# dig mage2cookbook.com NS +short
rocky.ns.cloudflare.com.
kate.ns.cloudflare.com.
```

You can also use the following command to check the IPs:

```
dig yourdomain.com +short
```

The output looks as follows:

```
root@mage2cookbook:~# dig mage2cookbook.com +short
104.18.56.216
104.18.57.216
```

9. Congratulations, you just finished configuring a CloudFlare CDN server with Magento 2.

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 9, we installed CloudFlare as a CDN to optimize our worldwide performance.

In steps 1 through 8, we created an account and moved our domain to the CloudFlare DNS. In step 7, we activated the orange cloud in DNS to start using the CDN optimization.

There's more...

If you are interested in how to test the performance of the CloudFlare setup, stay put. Here are some basic commands that you can use:

```
time curl --I http://yourdomain.com
```

The output looks as follows:

```
time curl -I http://mage2cookbook.com
HTTP/1.1 200 OK
X-Magento-Cache-Debug: HIT
Server: cloudflare-nginx
CF-RAY: 257330a8444d2bd6-AMS

real    0m0.198s
user    0m0.006s
sys     0m0.005s
```

Without CloudFlare, it looks as follows:

```
time curl -I http://mage2cookbook.com
HTTP/1.1 200 OK
X-Magento-Cache-Debug: HIT
```

```
real    0m0.253s
user    0m0.011s
sys     0m0.010s
```

Keep in mind that this current website is using Varnish. Our Magento 2 server is located in New York while our test server is located in Amsterdam. As you can see, in this test, we save 0.055s. This test is done from server to server. Doing a test from server to real browser clients on a desktop, or mobile device, will result in larger numbers. Larger numbers result in slower connections, which will result in lesser user experience.

Another great load testing tool is Siege. Using Siege helps you to understand how many concurrent clients can visit your website during high loads. We will just cover the basics of Siege here. Install Siege on another Droplet somewhere else in the world. Use the following command to install Siege:

```
apt-get install siege
```

Now let's run the following command. We will simulate 50 concurrent users for a period of three minutes. The -d option is the internal delay, in seconds, for which the users sleeps:

```
siege -c50 -d10 -t3M http://yourdomain.com
```

Without CloudFlare, the output looks as follows:

```
siege -c50 -d10 -t3M http://mage2cookbook.com
```

Transactions:	1732 hits
Availability:	100.00 %
Elapsed time:	179.79 secs
Data transferred:	15.47 MB
Response time:	0.18 secs
Transaction rate:	9.63 trans/sec
Throughput:	0.09 MB/sec
Concurrency:	1.71
Successful transactions:	1732

```
Failed transactions: 0
Longest transaction: 0.34
Shortest transaction: 0.15
```

With CloudFlare, the output looks as follows:

```
siege -c50 -d10 -t3M http://mage2cookbook.com
```

```
Transactions: 1716 hits
Availability: 100.00 %
Elapsed time: 179.74 secs
Data transferred: 14.05 MB
Response time: 0.10 secs
Transaction rate: 9.55 trans/sec
Throughput: 0.08 MB/sec
Concurrency: 0.96
Successful transactions: 1716
Failed transactions: 0
Longest transaction: 0.62
Shortest transaction: 0.08
```

In the last test, we can see that the **Response time** is 0.10 seconds compared to 0.18 seconds.

The test Droplet that we used was located in Amsterdam using two CPUs and 4 GB memory. For a real browser test, it is best to use tools such as Chrome developer tools. Those timings are more accurate and give you a better idea of the real user experience. Testing on a mobile device is a totally different ball game and is out of the scope of this book.

Configuring optimized images in Magento 2

Running a Magento store can be difficult—configuring the server, creating store views, and adding categories and products. Everyone knows that every product needs at least one product image. In some setups, we even have more than one. From this single master image, multiple thumbs are created, such as base image, small image, swatch image, and thumbnail.

By default, images are not optimized for the web when saving them in Photoshop. Images shown on a website are not exactly the same as images for print. The **Exchangeable Image File (EXIF)** data, for example, is not needed, and by removing this metadata, you can save lots of bytes. The smaller the image, the faster it's shown in the browser of the customer.

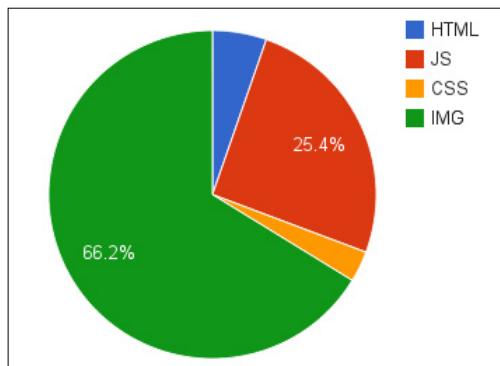
Here is an example of EXIF data (not optimized). The current file size is 620,888 bytes:

EXIF Data

File:

```
ExifByteOrder: Big-endian (Motorola, MM)
CurrentIPTCDigest: 50bb6030364fbdfb1842e98de0e81efe
ImageWidth: 1024
ImageHeight: 768
EncodingProcess: Baseline DCT, Huffman coding
BitsPerSample: 8
ColorComponents: 3
YCbCrSubSampling: YCbCr4:4:4 (1 1)
```

Storing all these images on your Magento server will result in slower pages and slower rendering of them. Almost 70% of all the content from a single page is filled with images:



So, optimizing images is not only important for desktop users, but also for mobile users. The less data they need to download, the better the user experience. Besides this, it's great for your search ranking optimization, battery consumption, and bandwidth/data plan.



By default, Magento 1 did not optimize the created catalog and CMS images. This could be optimized using software binaries such as **jpegtran**, **jpegoptim**, and **OptiPNG**.

If you don't have the option to install these, you could use **Rapido** image optimizer (<https://www.rapido.nu/>), which is a SaaS-based image optimizer for Magento. It is also the only optimizer that checks for the best available optimization per image and tunes all the images in the image cache directory on a daily basis.

Getting ready

For this recipe, we will use a Droplet created in *Chapter 2, Magento 2 System Tools*, at DigitalOcean, <https://www.digitalocean.com/>. We will be using NGINX, PHP-FPM, and a Composer-based setup including sample data for image optimization. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to optimize all Magento 2 images. The following steps will guide you through this:

1. By default, Magento 2 now uses an optimized GD2 PHP library, which is installed during the installation. The following command should be used during installation:

```
apt-get install php5-gd
```

Instead, we can also use the following command:

```
apt-get install php7.0-gd
```

To make sure that GD is installed correctly, run the following command:

```
php -i | grep gd
```

Run the following command to test which version of GD you are running:

```
echo '<?php var_dump(gd_info()); ?>' > gd.php
php gd.php
```

The output looks as follows:

```
root@mage2cookbook:/var/www/magento2/pub# php gd.php
array(13) {
    ["GD Version"]=>
        string(9) "2.1.1-dev"
```

2. Now let's log in to the backend of your Magento 2 control panel and navigate to the **Stores | Configuration | Advanced | Developer** section.

Here, you will find the following options:

- Template Settings**
- JavaScript Settings**
- CSS Settings**
- Image Processing Settings**
- Static Files Settings**

We first make sure that our `Image Adapter` is set to `PHP GD2`. We don't use the **ImageMagick** setting here. In the *There's more...* section of this recipe, you can find more information on this.

3. Next, we change the **CSS Settings**, **JavaScript Settings**, **Static Files Settings**, and **Template Settings**. In the **Template Settings**, adjust **Minify HTML** to **YES**.

Next, enable **JavaScript Bundling**, **Merge JavaScript**, and **Minify JavaScript** to **YES**.

Next, enable **Merge CSS** and **Minify CSS** to **YES**.

Last but not least, enable **Static Files** to **YES**. Save all the settings.

4. Before we have all the optimized code available, we need to recompile all static assets. Let's assume that we are preparing for production. Run the following code on the shell:

```
php bin/magento deploy:mode:set production
```

Before running the code, make sure to change your Apache or NGINX configuration to set `$MAGE_MODE production;` (Nginx) or `SetEnv MAGE_MODE production` (Apache). In *Managing Magento 2, set mode (MAGE_MODE)* recipe of Chapter 2, *Magento 2 System Tools*, we covered everything in detail.

After running this code, make sure to change your user and group permissions. Run the following command:

```
chown -R www-data:www-data *
```

5. Congratulations, you just finished configuring optimized images, JavaScript, and CSS with Magento 2. The following image is a screenshot from the Google PageSpeed insight page (<https://developers.google.com/speed/pagespeed/insights/>), where you can test your own pages:

The screenshot shows a list of 8 passed rules from the Google PageSpeed Insights report. Each rule has a green checkmark icon and a link to learn more about the optimization.

- 8 Passed Rules
- Avoid landing page redirects
- Enable compression
- Leverage browser caching
- Minify CSS
- Minify JavaScript
- Optimize images
- Prioritize visible content
- Reduce server response time

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 5, we configured the image optimizing technique, which is now by default available in Magento 2.

In step 1, we installed the PHP GD library and tested it. In step 2, we configured the Magento backend to start using the optimization by selecting the GD option and additional merging for JS and CSS.

In step 4, we ran the bin/magento production mode to start optimizing all of the code.

There's more...

Besides the PHP GD2 library, Magento 2 offers the option to switch to the ImageMagick library (<http://www.imagemagick.org/>). In basis, this library works great for image optimization, but during some tests, we found out that the GD2 had a smaller output. Besides the difference in size, ImageMagick generated files in the baseline (renders top-down) format instead of the progressive (renders from blurry to sharp) format that GD2 does.

Using progressive is the best commonly used format for web pages. It starts as a blurry image and turns sharp when done. It improves the user experience by loading the images incrementally.

If you still want to use ImageMagick, here are some basic commands. Run the following code on your shell. Then, switch to ImageMagick in your Magento configuration backend:

```
apt-get install -y imagemagick --fix-missing  
apt-get install -y php5-imagick  
service php-fpm restart  
php -i | grep imagick
```

Configuring Magento 2 with HTTP/2

December 17, 2015, is the day Google mentioned that HTTPS pages have top priority by default. Many Magento websites still use the default SSL pages or, even worse, don't use SSL at all.

Well, this will change now if your website depends on Google's search ranking. Using HTTP/2 in your setup is a must for high-performing and secure websites. The new protocol will be the new standard for fast and secure browsing.

HTTP/2 has many new benefits such as multiple TCP connections, cache pushing (server push), data compression, and much more. By default, HTTP/2 does not need SSL, but many browsers out there will support it only when configured using SSL. NGINX, for example, supports HTTP/2 only when configured including SSL; Apache, on the other hand, supports both, with or without SSL.

So, it is mandatory that we start using HTTP/2 including SSL for a safer and faster web.

Getting ready

For this recipe, we will use a Droplet created in *Chapter 2, Magento 2 System Tools*, at DigitalOcean, <https://www.digitalocean.com/>. We will be using NGINX, PHP-FPM, and a Composer-based setup including sample data for HTTP/2. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to create a Magento 2 using HTTP/2 including SSL. The following steps will guide you through this:

1. First, we need to configure and create an SSL certificate. Open `openssl.conf` located in `/etc/ssl` with your favorite editor:

```
vi /etc/ssl/openssl.conf
```

Go to line 127 [`req_distinguished_name`] and change or add the settings regarding your company and domain. Change the following lines; here is an example:

<code>countryName_default</code>	= Some-CountryName
<code>stateOrProvinceName_default</code>	= Some-State
<code>localityName_default</code>	= Some-CityName
<code>0.organizationName_default</code>	= Some-CompanyName
<code>organizationalUnitName_default</code>	= Some-DepartmentName
<code>commonName_default</code>	= Some-DomainName
<code>emailAddress_default</code>	= Some-Email

The following screenshot depicts an example of the same:

```

root@mage2cookbook: /etc/ssl
[ req_distinguished_name ]
countryName = Country Name (2 letter code)
countryName_default = AU
countryName_min = 2
countryName_max = 2

stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = New York

localityName = Locality Name (eg, city)
localityName_default = New York

0.organizationName = Organization Name (eg, company)
0.organizationName_default = Magento 2 CookBook

# 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 = Kitchen

commonName = Common Name (e.g. server FQDN or YOUR name)
commonName_default = *.mage2cookbook.com
commonName_max = 64

emailAddress = Email Address
emailAddress_default = info@mage2cookbook.com
emailAddress_max = 64

# SET-ex3 = SET extension number 3

[ req_attributes ]
challengePassword = A challenge password
challengePassword_min = 4
challengePassword_max = 20

unstructuredName = An optional company name

```

128,1 40% ▾

- After saving your `openssl.conf` file, we can create the `*.csr` and `*.key` files. We need the `*.csr` file and send it to our SSL provider. You may pick any SSL provider. Run the following command to generate them:

```
openssl req -new -newkey rsa:2048 -nodes -keyout yourname.key -out yourname.csr
```

Change yourname with any given name. When running the command, questions will be asked; hit enter to prompt when the default is okay. Here is a screenshot of the process:

```
root@mage2cookbook:/etc/ssl# openssl req -new -newkey rsa:2048 -nodes -keyout mage2cookbook.key -out mage2cookbook.csr
Generating a 2048 bit RSA private key
-----
writing new private key to 'mage2cookbook.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [New York]:
Locality Name (eg, city) [New York]:
Organization Name (eg, company) [Magento 2 CookBook]:
Organizational Unit Name (eg, section) [Kitchen]:
Common Name (e.g. server FQDN or YOUR name) [*.www.buy-certificate.com]:
Email Address [info@mage2cookbook.com]:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
root@mage2cookbook:/etc/ssl#
```

Check your certificate before you submit it. Run the following code to confirm your settings:

```
openssl req -in yourname.csr -text -noout
```

In this example, we used a wildcard SSL certificate. The wildcard starts with *.yourdomain.com. We use a wildcard to create unlimited subdomain names, which we will use later to create localized domain names such as de.yourdomain.com or fr.yourdomain.com.

If you don't need a wildcard domain and would rather use www.yourdomain.com or a naked domain such as yourdomain.com, commit this in your openssl.conf file.

3. Submit the *.csr file to your SSL provider and continue all the steps necessary. Depending on your provider, it can take minutes or hours. For the purpose of demonstration, we used <https://www.buy-certificate.com/>. On this website, there is an option to create a 30-day free SSL certificate. The whole process takes two to three minutes.

- Now let's download the ZIP file from your mail account to your Droplet and open it in your root directory. Unzip the `yourdomain-com.zip` file by running the following command:

```
unzip mage2cookbook-com.zip
```

- Your ZIP contains the following files (or similar ones):

```
Archive: mage2cookbook-com.zip
inflating: mage2cookbook-com.cer
inflating: readme.txt
inflating: RapidSSLSHA256CA-G3.cer
inflating: GeoTrustGlobalCA.cer
inflating: siteseal_nw4all.html
```

- Now we will merge the certificate and CA authority key. Use the following command on the shell:

```
cat mage2cookbook-com.cer RapidSSLSHA256CA-G3.cer > mage2cookbook-com-2015.cert
```

- Now let's copy the `mage2cookbook-com-2015.cert` file to `/etc/ssl/cert` using the following command:

```
cp mage2cookbook-com-2015.cert /etc/ssl/cert
```

- Move the generated `mage2cookbook.key` to `/etc/ssl/private` using the following command: (Let's assume that you are running the `openssl reg` command in the `/etc/ssl` directory.)

```
mv /etc/ssl/mage2cookbook.key /etc/ssl/private
```

- Now let's create a symbolic link of the keys. Run the following command:

```
ln -s /etc/ssl/private/mage2cookbook-com.key /etc/ssl/
mage2cookbook-com.key
```

```
ln -s /etc/ssl/certs/mage2cookbook-com-2015.cer /etc/ssl/
mage2cookbook-com.cert
```

Try to list all the files in the `/etc/ssl` directory using the following command. You should see the names of the files that we linked:

```
ll /etc/ssl
```

- Now let's go to the NGINX configuration directory and update `default.conf` in `/etc/nginx/conf.d`. Open the `default.conf` file and change it with the following settings:

```
upstream fastcgi_backend {
    server 127.0.0.1:9000;
}
```

```
server {  
    listen      80;  
    listen      443 ssl http2;  
  
    server_name yourdomain.com;  
  
    set $MAGE_ROOT /var/www/html;  
    set $MAGE_MODE developer;  
  
    ssl_certificate /etc/ssl/yourdomain.com.cert;  
    ssl_certificate_key /etc/ssl/yourdomain.com.key;  
  
    include /var/www/html/nginx.conf.sample;  
  
    access_log /var/log/nginx/access.log;  
    error_log /var/log/nginx/error.log;  
  
    location ~ /\.ht {  
        deny all;  
    }  
}
```

As you can see, we created a new `listen 443 ssl http2` section. Besides the `listen` section, we also created `ssl_certificate` and `ssl_certificate_key`. The `http2` flag in the `listen` section covers the entire HTTP/2 configuration.

11. Now, all you have to do is restart NGINX to use your new settings. Run the following command:

```
service nginx restart
```

12. Before we can test Magento in our browser, we need to flush and clean the cache. We also need to update Magento's configuration with the new secure URL. Run the following commands:

```
php bin/magento setup:store-config:set --base-url-secure="https://  
yourdomain.com/"  
  
php bin/magento setup:store-config:set --use-secure-admin="1"  
php bin/magento setup:store-config:set --use-secure="1"  
  
php bin/magento setup:static-content:deploy  
php bin/magento cache:clean  
php bin/magento cache:flush
```

Next, we go to <https://www.sslshopper.com/ssl-checker.html> and check our setup. Commit your domain name in the box and submit. If everything is configured correctly, the output should look as follows:

Server Hostname: yourdomain.com (e.g. www.google.com) **Check SSL**

- yourdomain.com resolves to 123.456.678.0**
- The certificate should be trusted by all major web browsers (all the correct intermediate certificates are installed).**
- The certificate was issued by [GeoTrust](#). [Write review of GeoTrust](#)**
- The certificate will expire in 26 days. [Remind me](#)**
- The hostname (yourdomain.com) is correctly listed in the certificate.**

Server

Common name: yourdomain.com
 SANs: yourdomain.com
 Valid from December 20, 2015 to January 20, 2016
 Serial Number: 597922 (0x91fa2)
 Signature Algorithm: sha256WithRSAEncryption
 Issuer: RapidSSL SHA256 CA - G3

Chain

Common name: RapidSSL SHA256 CA - G3
 Organization: GeoTrust Inc.
 Location: US
 Valid from August 29, 2014 to May 20, 2022
 Serial Number: 146039 (0x23a77)
 Signature Algorithm: sha256WithRSAEncryption
 Issuer: GeoTrust Global CA

13. Congratulations, you just finished configuring HTTP/2 with Magento 2. To test your HTTP/2 protocol, go to <https://tools.keycdn.com/http2-test> and submit yourdomain.com.

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 13, we created an SSL certificate, which we need to configure HTTP/2 in NGINX.

In step 1, we configured the `openssl.conf` file with our domain and business data. In step 2, we created a certificate request that we will be sending to the SSL provider.

In step 4, we downloaded the provided certificate file and unzipped the content. In step 6, we merged the domain certificate and certificate authority file into a single one. This file was then copied to the SSL directory.

In step 6, we copied the private key to the SSL private directory before we started creating a symlink of the private key and merge certificate in the /etc/ssl directory. The main reason why we stored the files in the private and cert directory is maintenance. When replacing or updating keys or certificates in the future, we only need to create a new symlink while our NGINX or Apache configuration can stay the same.

In step 10, we updated the NGINX configuration and added the `ssl_certificate` parameter including the correct SSL directory. In the `listen` parameter, we added the `http2` flag behind the `443 ssl` flag and restarted the NGINX server.

In step 12, we configured the HTTPS domains using the `bin/magento setup:store-config:set` option.

There's more...

Setting up Magento 2 including SSL and HTTP/2 is pretty straightforward. However, by default, the only URLs that serve HTTPS are `customer/account/login/`, `customer/account/create/`, `checkout/`, `checkout/cart/`, `contact/`, and `sales/guest/form/`. Currently, it's mandatory to have a full HTTPS website (Google: HTTPS as a ranking signal).

It is easy to update the Magento configuration to serve every URL on HTTPS using the following command:

```
php bin/magento setup:store-config:set --base-url="https://yourdomain.com/"  
php bin/magento setup:static-content:deploy  
php bin/magento cache:clean  
php bin/magento cache:flush
```



When using Varnish in your setup, make sure to offload your SSL. Varnish does not support SSL. The best common setup is NGINX as an SSL Proxy on the frontend, rather than Varnish, and in the backend, NGINX or Apache.

Configuring Magento 2 performance testing

Performance, performance, performance! This may be one of the most used words in the Magento 1 period. Every Magento website benefits from a great performing platform, and every customer loves it.

However, before we can create a great performing website, all sorts of elements have to be conquered. One of the missing elements in Magento 1 was creating sample data based on a company profile. As every Magento website is unique, so is performance testing based on their profile. Some companies have only one website, store catalog, and store view. Others have 800 websites and are converting one million orders per day.

Magento 2 now provides us with the option to run a profile that creates sample data based on your company profile. By default, there are four sample data profiles. Depending on the profile, creating the sample data may take a long time, so keep this in mind.

After creating the sample data based on your profile, you can start doing performance-based testing. Based on this profile, you may need to scale up or tune one of the components before going into production.

Getting ready

For this recipe, we will use a Droplet created in *Chapter 2, Magento 2 System Tools*, at DigitalOcean, <https://www.digitalocean.com/>. We will be using NGINX, PHP-FPM, and a Composer-based setup including Magento 2 (without sample data). No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to run a Magento 2 performance test. The following steps will guide you through this:

1. Before we can start generating a profile, we need a clean setup. Run the following command to start with a clean Magento 2 instance:

```
rm -rf * /var/www/html  
composer create-project --repository-url=https://repo.magento.com/  
magento/project-community-edition /var/www/html --prefer-dist  
chown -R www-data:www-data /var/www/html
```

2. Now let's install Magento 2 without sample data. Run the following command. We will be using the same procedure as we used in *Installing Magento 2 sample data via the command line* recipe of *Chapter 2, Magento 2 System Tools*:

```
bin/magento setup:install \  
--db-host=localhost \  
--db-name=<your-db-name> \  
--db-user=<db-user> \  
--db-password=<db-password> \  
--backend-frontname=<admin-path> \  
--base-url=http://yourdomain.com/ \  
--admin-lastname=<your-lastname> \  
--admin-firstname=<your-firstname> \  
--admin-email=<your-email> \  
--admin-user=<your-admin-user> \  
--admin-password=<your-password> \  
--use-rewrites=1 \  
--cleanup-database \  
--
```

3. After completing the install via the shell, we need to compile our code before we can start using it. Run the following command on the shell:

```
php bin/magento setup:di:compile-multi-tenant  
chown -R www-data:www-data /var/www/html
```

4. Check in your browser whether everything is working correctly before starting the small data profile. We use the small data profile because it does not take too long to run. Run the following command:

```
php bin/magento setup:perf:generate-fixtures /var/www/html/setup/  
performance-toolkit/profiles/ce/small.xml
```

5. All data profiles are located in the `setup/performance-toolkit/profiles/` directory. Depending on whether you are running CE or EE, you need to choose one of the subdirectories.

6. Depending on the profile that you ran, the output looks as follows:

The screenshot shows a terminal window titled "root@web1: /var/www/html". The command entered is "php bin/magento setup:perf:generate-fixtures". The output displays the generation of a performance profile with the following parameters:

```
Generating profile with following params:  
| - Websites: 1  
| - Store Groups: 1  
| - Store Views: 1  
| - Categories: 30  
| - Simple products: 800  
| - Configurable products: 50  
| - Customers: 20  
| - Cart Price Rules: 10  
| - Catalog Price Rules: 10  
| - Orders: 80  
Generating websites, stores and store views... done in 00:00:00  
Generating categories... done in 00:00:03  
Generating simple products... done in 00:00:08  
Generating configurable EAV variations... done in 00:00:00  
Generating configurable products... done in 00:00:04  
Generating customers... done in 00:00:00  
Generating Cart Price Rules... done in 00:00:00  
Generating catalog price rules... done in 00:00:04  
Generating tax rates... done in 00:03:44  
Generating orders... done in 00:00:05  
Config Changes... done in 00:00:00  
Indexers Mode Changes... done in 00:00:01  
Customer Grid index has been rebuilt successfully in 00:00:00  
Category Products index has been rebuilt successfully in 00:00:00  
Product Categories index has been rebuilt successfully in 00:00:00  
Product Price index has been rebuilt successfully in 00:00:00  
Product EAV index has been rebuilt successfully in 00:00:00  
Stock index has been rebuilt successfully in 00:00:00  
Catalog Rule Product index has been rebuilt successfully in 00:00:14  
Catalog Product Rule index has been rebuilt successfully in 00:00:14  
Catalog Search index has been rebuilt successfully in 00:00:00  
Total execution time: 00:04:49
```

7. Now you can open a browser and surf to `yourdomain.com`, and check whether everything is correct. You can also log in to the backend of your Magento website and check all the created settings.
8. Congratulations, you just finished creating profiles for performance testing with Magento 2. Now you can choose your favorite performance test tool and test your server:

The screenshot shows a Magento 2 storefront with a LUMA theme. At the top, there is a navigation bar with links for "Default welcome msg!", "Sign In or Create an Account", and "USD - US Dollar". The main header features the LUMA logo and a search bar with placeholder text "Search entire store here...". A shopping cart icon is also present.

The main content area displays a grid of categories. The categories are arranged in rows and columns, with some categories highlighted in blue. The categories visible include:

Category 1	Category 7	Category 13	Category 19	Category 25	Category 31
Category 37	Category 43	Category 49	Category 55	Category 61	Category 67
Category 73	Category 79	Category 85	Category 91	Category 97	Category 103
Category 109	Category 115	Category 121	Category 127	Category 133	
Category 139	Category 145	Category 151	Category 157	Category 163	
Category 169	Category 175	Category 181	Category 187	Category 193	
Category 199	Category 177		Category 217	Category 223	
Category 229	Category 235	Category 241	Category 247	Category 253	
Category 259	Category 265	Category 271	Category 277	Category 283	
Category 289	Category 295				

Below the grid, the breadcrumb navigation shows "Home > Category 295". The page title is "Category 295".

On the left side, there is a sidebar with a "Filter" section and a "Category" section. The "Category" section shows a link to "Category 297 56".

The main content area displays three large, identical placeholder images for products. Below these images, a message states "You have no items to compare.".

At the bottom of the page, there is a footer section with links for "My Wish List", "Configurable Product 147" (\$10.00), "Configurable Product 447" (\$10.00), and "Configurable Product 747" (\$10.00).

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 8, we created sample data to test the performance of the Magento 2 website.

In steps 1 and 2, we started with a clean Magento 2 setup using Composer and `bin/magento setup:install`.

In step 3, we needed to compile our Magento code base before we could input the sample data.

In step 4, we ran a generated fixture profile using the `bin/magento setup:perf` option. Depending on the profile, Magento will start creating all of the required data. Running a large profile set can take up to several hours. Adjusting the profile is self-explanatory.

There's more...

If the default profile does not fit your needs, you can create a custom profile. For example, copy the `small.xml` file to `mycustom.xml` in the same directory and open the file in your favorite editor. Run the following command on the shell:

```
cd /var/www/html/setup/performance-toolkit/profiles/ce/small.xml  
cp small.xml mycustom.xml  
vi mycustom.xml
```

Now you can change data such as `websites`, `store_groups`, `store_views`, `simple_products`, `configurable_products`, `categories`, `categories_nesting_level`, `catalog_price_rules`, `catalog_target_rules`, `cart_price_rules`, `cart_price_rules_floor`, `customers`, `tax_rates_file`, and `orders`:

```

<?xml version="1.0"?>
<!--
/**/
 * Copyright © 2015 Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<config>
    <profile>
        <websites>1</websites> <!-- Number of websites to generate -->
        <store_groups>1</store_groups> <!--Number of stores-->
        <store_views>1</store_views> <!-- Number of store views -->
        <simple_products>800</simple_products> <!-- Simple products count -->
        <configurable_products>50</configurable_products> <!--Configurable products count (each configurable has 3 simple products as options, that are not displayed individually in catalog) -->
        <categories>30</categories> <!-- Number of categories to generate -->
        <categories_nesting_level>3</categories_nesting_level> <!-- Nesting level for categories -->
        <catalog_price_rules>10</catalog_price_rules> <!-- Number os catalog price rules -->
        <cart_price_rules>10</cart_price_rules> <!-- Number of cart price rules -->
        <cart_price_rules_floor>2</cart_price_rules_floor> <!-- The price rule condition: minimum products amount in shopping cart for price rule to be applied -->
        <customers>20</customers> <!-- Number of customers to generate -->
        <tax_rates_file>tax_rates.csv</tax_rates_file> <!-- Tax rates file in fixtures directory-->
        <orders>80</orders> <!-- Orders count -->
    </profile>
</config>

```

Save your file, and use the following command:

Esc + :wq

Before we start, you need to check whether your setup is clean. Otherwise, start with step 1 of the recipe.

Now we can run the `mycustom.xml` file with the following command:

```
php bin/magento setup:perf:generate-fixtures /var/www/html/setup/performance-toolkit/profiles/ce/mycustom.xml
```

[ Running the profile can take some time. Be aware to adjust your server setup accordingly. An extra large profile can take up to a couple of hours to create.]

4

Creating Catalogs and Categories

In this chapter, we will cover the basic tasks related to creating a catalog and products in Magento 2. You will learn how to:

- ▶ Create a Root Catalog
- ▶ Create subcategories
- ▶ Manage attribute sets
- ▶ Create products
- ▶ Manage products in a catalog grid

Introduction

This chapter explains how to set up a vanilla Magento 2 store. If Magento 2 is totally new to you, then lots of new basic whereabouts are pointed out. Are you currently working with Magento 1? If so, not much has changed since then.

The new backend of Magento 2 is the biggest improvement of them all. The design is built responsively and has a great user experience. Compared to Magento 1, this is a great improvement. The menu is located vertically on the left of the screen and works great on desktop and mobile environments:

The screenshot shows the Magento 2 Admin Dashboard. On the left is a vertical sidebar with icons for Dashboard, Sales, Products, Customers, Marketing, Content, Reports, Stores, System, and Find Partners & Extensions. The main area is titled 'Dashboard' and includes sections for 'Lifetime Sales' (\$0.00), 'Average Order' (\$0.00), 'Last Orders' (No Data Found), 'Last Search Terms' (No Data Found), and 'Top Search Terms' (No Data Found). A 'Store View' dropdown is set to 'All Store Views'. A 'Reload Data' button is at the top right. At the bottom, there's a copyright notice for Magento Commerce Inc. and links for 'Magento ver. 2.0.0' and 'Report Bugs'.

Within this chapter, we will learn how to set up a website with multiple domains using different catalogs and products. Depending on the website, store, and store view setup, we can create different subcategories, URLs, and products for any domain name.

There are a number of different ways customers can browse your store, but one of the most effective is **layered navigation**. Layered navigation is located in your catalog and holds product features to sort or filter. We will learn how to create product attributes for use in layered navigation.

Every website benefits from great **search engine optimization (SEO)**. We will learn how to define catalog URLs for catalogs.

Without products, the most important element of the website is missing. We will be creating different types of product in our multi-website setup.

 Throughout this chapter we will cover the basics of how to set up a multi-domain setup. Additional tasks required to complete a production setup are beyond the scope of this chapter.

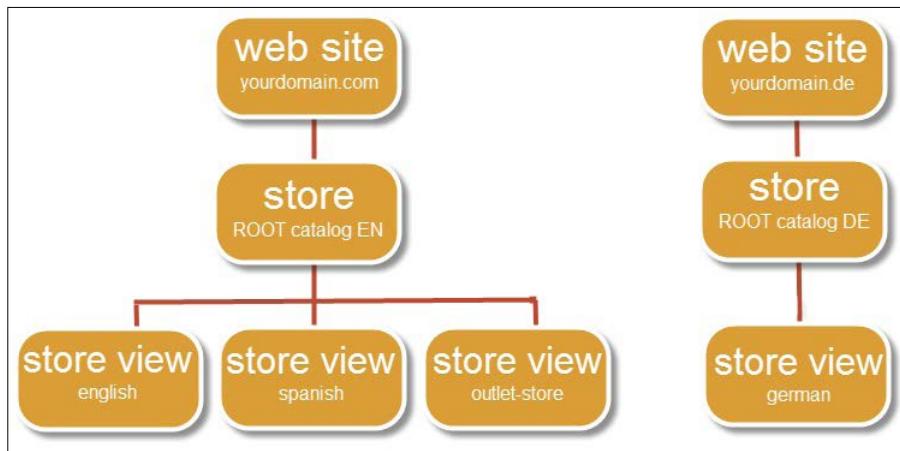
Create a Root Catalog

The first thing we need to do when setting up a vanilla Magento 2 website is define our **website**, **store**, and **store view** structure.

So what is the difference between website, store, and store view, and why is it important?

- ▶ Website is the top-level container and the most important of the three. It is the parent level of the entire store and used, for example, to define domain names, different shipping methods, payment options, customers, orders, and so on.
- ▶ Stores can be used to define, for example, different store views with the same information. A store is always connected to a Root Catalog that holds all the categories and subcategories. One website can manage multiple stores, but every store has a different Root Catalog. When using multiple stores, it is not possible to share one basket. The main reason for this has to do with the configuration setup, where shipping, catalog, customer, inventory, taxes, and payment settings are not shareable between different sites.
- ▶ Store view is the lowest level and mostly used to handle different localizations. Every store view can have a different language. Besides using store views just for localizations, they can also be used for **Business to Business (B2B)**, hidden private sales pages (with noindex andnofollow), and so on. The option where we use the Base Link URL, for example, (`yourdomain.com/myhiddenpage`) is easy to set up.

The website, store, and store view structure is shown in the following image:



Getting ready

To step through this recipe, you will use a Droplet created in *Chapter 2, Magento 2 System Tools*, at DigitalOcean (<https://www.digitalocean.com/>). We will be using an NGINX, PHP-FPM, Composer-based setup with Magento 2 preinstalled. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to create a multi-website setup including three domains (`yourdomain.com`, `yourdomain.de`, and `yourdomain.fr`) and separated Root Catalogs. The following steps will guide you through this:

1. First we need to update our NGINX. We need to configure the additional domains before we can connect them to Magento. Make sure that all domain names are connected to your server and DNS is configured correctly.

Go to `/etc/nginx/conf.d`, open the `default.conf` file, and include the following content at the top of your file:

```
map $http_host $magecode {  
    hostnames;  
    default base;  
    yourdomain.de de;  
    yourdomain.fr fr;  
}
```

2. Your configuration should look like this now:

```
map $http_host $magecode {  
    hostnames;  
    default base;  
    yourdomain.de de;  
    yourdomain.fr fr;  
}  
  
upstream fastcgi_backend {  
    server 127.0.0.1:9000;  
}  
server {  
    listen 80;  
    listen 443 ssl http2;  
  
    server_name yourdomain.com;  
  
    set $MAGE_ROOT /var/www/html;
```

```

set $MAGE_MODE developer;

ssl_certificate /etc/ssl/yourdomain-com.cert;
ssl_certificate_key /etc/ssl/yourdomain-com.key;

include /var/www/html/nginx.conf.sample;

access_log /var/log/nginx/access.log;
error_log /var/log/nginx/error.log;

location ~ /\.ht {
    deny all;
}
}
}

```

- Now let's go to the Magento 2 configuration file in `/var/www/html/` and open the `nginx.conf.sample` file. Go to the bottom and look for:

```
location ~ (index|get|static|report|404|503)\.php$
```

Now we add the following lines to the file under `fastcgi_pass fastcgi_backend;`:

```
fastcgi_param MAGE_RUN_TYPE website;
fastcgi_param MAGE_RUN_CODE $magecode;
```

- Your configuration should look like this now (this is only a small section of the bottom):

```

location ~ (index|get|static|report|404|503)\.php$ {
    try_files $uri =404;
    fastcgi_pass fastcgi_backend;

    fastcgi_param MAGE_RUN_TYPE website;
    fastcgi_param MAGE_RUN_CODE $magecode;

    fastcgi_param PHP_FLAG "session.auto_start=off \n
        suhosin.session.encryptua=off";
    fastcgi_param PHP_VALUE "memory_limit=256M \n
        max_execution_time=600";
    fastcgi_read_timeout 600s;
    fastcgi_connect_timeout 600s;
    fastcgi_param MAGE_MODE $MAGE_MODE;

    fastcgi_index index.php;
    fastcgi_param SCRIPT_FILENAME
        $document_root$fastcgi_script_name;
    include fastcgi_params;
}

```

The current setup uses the `MAGE_RUN_TYPE` website variable. You may change website to store, depending on your setup preferences. When changing the variable, you need your default .conf mapping codes as well.

- Now all you have to do is restart NGINX and PHP-FPM to use your new settings. Run the following command:

```
service nginx restart && service php-fpm restart
```

- Before we continue, we need to check if our web server is serving the correct codes. Run the following command in the Magento 2 web directory:

```
var/www/html/pub
```

```
echo "<?php header('Content-type: text/plain'); print_r($_SERVER); ?>" > magecode.php
```

Don't forget to update your `nginx.conf.sample` file with the new `magecode` code. It's located on the bottom of your file and should look like this:

```
location ~ (index|get|static|report|404|503|magecode) \.php$ {
```

Restart NGINX and open the file in your browser. The output should look as follows. As you can see, the created `MAGE_RUN` variables are available:

<code>[USER] => app</code>	<code>[USER] => app</code>	<code>[USER] => app</code>
<code>[HOME] => /home/app</code>	<code>[HOME] => /home/app</code>	<code>[HOME] => /home/app</code>
<code>[FCGI_ROLE] => RESPONDER</code>	<code>[FCGI_ROLE] => RESPONDER</code>	<code>[FCGI_ROLE] => RESPONDER</code>
<code>[MAGE_RUN_TYPE] => website</code>	<code>[MAGE_RUN_TYPE] => website</code>	<code>[MAGE_RUN_TYPE] => website</code>
<code>[MAGE_RUN_CODE] => base</code>	<code>[MAGE_RUN_CODE] => fr</code>	<code>[MAGE_RUN_CODE] => de</code>
<code>[PHP_FLAG] => session.auto_start=off</code>	<code>[PHP_FLAG] => session.auto_start=off</code>	<code>[PHP_FLAG] => session.auto_start=off</code>
<code>suhosin.session.encrypt=off</code>	<code>suhosin.session.encrypt=off</code>	<code>suhosin.session.encrypt=off</code>
<code>[PHP_VALUE] => memory_limit=256M</code>	<code>[PHP_VALUE] => memory_limit=256M</code>	<code>[PHP_VALUE] => memory_limit=256M</code>
<code>max_execution_time=600</code>	<code>max_execution_time=600</code>	<code>max_execution_time=600</code>
<code>[MAGE_MODE] => developer</code>	<code>[MAGE_MODE] => developer</code>	<code>[MAGE_MODE] => developer</code>

- Congratulations, you just finished configuring NGINX including additional domains. Now let's continue connecting them in Magento 2.
- Log in to the backend and go to **Stores | All Stores**. By default, Magento 2 has one **Website**, **Store**, and **Store View** setup. Now click on **Create Website** and commit the following details:

Name	My German Website
Code	de

Next, click on **Create Store** and commit the following details:

Website	My German Website
Name	My German Website
Root Category	Default Category (we will change this later)

Next, click on **Create Store View** and commit the following details:

Store	My German Website
Name	German
Code	de
Status	Enabled

Repeat the same steps for the French domain. Make sure that the **Code** in **Website** and **Store View** is **fr**.

- The next important step is to connect the websites with the domain name. Go to **Stores | Configuration | Web | Base URLs**. Change the **Store View** scope at the top to **My German Website**. You will be prompted when switching; press **OK** to continue. Now uncheck the checkbox called **Use Default** from the **Base URL** and **Base Link URL** fields and commit your domain name. Now click **Save Config** and continue the same procedure for the other website. The output should look like this:

Store View: My German Website ▾ ? Save Config

GENERAL ▾ Search Engine Optimization ⌂

General

Web

Design

Currency Setup

Store Email Addresses

Base URLs

Any of the fields allow fully qualified URLs that end with '/' (slash) e.g. http://example.com/magento/

Base URL Specify URL or {{base_url}} placeholder. Use Default [STORE VIEW]

Base Link URL May start with {{unsecure_base_url}} placeholder. Use Default [STORE VIEW]

- Save your entire configuration and clear your cache. Now go to **Products | Categories** and click on **Add Root Category** with the following data:

Name	Root German
Is Active	Yes
Page Title	My German Website

Perform the same steps for the French domain. You may add additional information here but it is not needed. Changing the current Root Category called **Default Category** to Root English is also optional but advised.

Save your configuration and go to **Stores | All Stores** and change all of the stores to the appropriate Root Catalog we just created. Every Root Category should now have a dedicated Root Catalog.

11. Congratulations, you just finished configuring Magento 2 including additional domains and dedicated Root Categories. Now let's open up a browser and surf to the domain names you created: `yourdomain.com`, `yourdomain.de`, and `yourdomain.fr`.

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 11, we created a multi-store setup for `.com`, `.de`, and `.fr` domains using a separate Root Catalog.

In steps 1 through 4, we configured the domain mapping in the NGINX `default.conf` file. Then we added the `fastcgi_param MAGE_RUN` code to the `nginx.conf.sample` file; this will manage which website or store view to request within Magento.

In step 6, we used an easy test method to check if all domains run the correct `MAGE_RUN` code.

In steps 7 through 9, we configure the website, store, and store view names and codes for the given domain names.

In step 10, we created additional Root Catalogs for the remaining German and French stores. They are then connected to the previously created store configuration. All stores have their own Root Catalog now.

There's more...

Are you able to buy additional domain names, but would like to try setting up a multi-store? Here are some tips to create one. Depending on whether you are using Windows, Mac OS, or Linux, the following options apply:

- ▶ **Windows:** Go to `C:\Windows\System32\drivers\etc` and open up the `hosts` file as an administrator. Add the following (change the IP and domain name accordingly):

123.456.789.0	<code>yourdomain.de</code>
123.456.789.0	<code>yourdomain.fr</code>
123.456.789.0	<code>www.yourdomain.de</code>
123.456.789.0	<code>www.yourdomain.fr</code>

Save the file and click on the **Start** button. Search then for `cmd.exe` and commit the following:

```
ipconfig /flushdns
```

- ▶ **Mac OS:** Go to the `/etc/` directory, open up the `hosts` file as a superuser, and add the following (change the IP and domain name accordingly):

```
123.456.789.0      yourdomain.de
123.456.789.0      yourdomain.fr
123.456.789.0      www.yourdomain.de
123.456.789.0      www.yourdomain.fr
```

Save the file and run the following command on the shell:

```
dscacheutil -flushcache
```

Depending on your Mac version, check out the different commands here:

<http://www.hongkiat.com/blog/how-to-clear-flush-dns-cache-in-os-x-yosemite/>

- ▶ **Linux:** Go to the `/etc/` directory, open up the `hosts` file as a root user, and add the following (change the IP and domain name accordingly):

```
123.456.789.0      yourdomain.de
123.456.789.0      yourdomain.fr
123.456.789.0      www.yourdomain.de
123.456.789.0      www.yourdomain.fr
```

Save the file and run the following command on the shell:

```
service nscd restart
```

Depending on your Linux version, check out the different commands here: <http://www.cyberciti.biz/faq/rhel-debian-ubuntu-flush-clear-dns-cache/>

Open up your browser and surf to the custom domains.

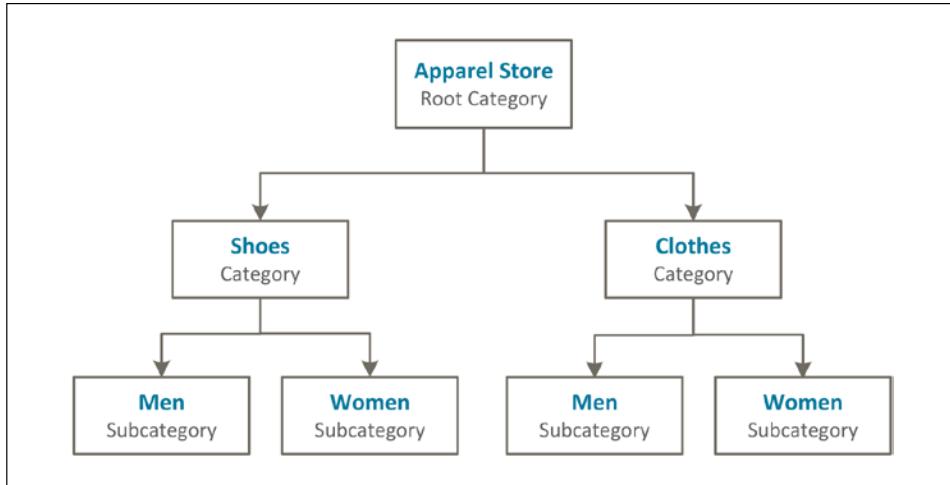


These domains only work on your PC. You can copy these IP and domain names on as many PC as you prefer. This method also works great when you are developing or testing and your production domain is not available on your development environment.

Create subcategories

After creating the foundation of the website, we need to set up a catalog structure. Setting up a catalog structure is not difficult but needs to be well thought out.

Some websites have an easy setup using two levels, while others sometimes use five or more subcategories. Always keep in mind user experience: your customer needs to crawl the pages easily. Keep it simple! The following image shows a simple catalog structure:



Getting ready

To step through this recipe, you will use a Droplet created in *Chapter 2, Magento 2 System Tools*, at DigitalOcean (<https://www.digitalocean.com/>). We will be using an NGINX, PHP-FPM, Composer-based setup with Magento 2 preinstalled. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to set up a catalog including subcategories. The following steps will guide you through this:

1. First, log in to the backend of Magento 2 and then go to **Products | Categories**.

Since we have already created Root Catalogs, we start with using the Root English catalog first.

2. Click on the **Root English** catalog on the left and then select the **Add Subcategory** button above the menu. Now commit the following and repeat all steps again for the other Root Catalogs:

Name	Shoes (Schuhe) (Chaussures)
Is Active	Yes
Page Title	Shoes (Schuhe) (Chaussures)

Name	Clothes (Kleider) (Vêtements)
Is Active	Yes
Page Title	Clothes (Kleider) (Vêtements)

3. Since we created the first level of our catalog, we can continue with the second level. Now click on the first level, which you need to extend with subcategories, and select the **Add Subcategory** button. Commit the following and repeat all steps again for the other Root Catalogs:

Name	Men (Männer) (Hommes)
Is Active	Yes
Page Title	Men (Männer) (Hommes)

Name	Women (Frau) (Femmes)
Is Active	Yes
Page Title	Women (Frau) (Femmes)

4. Congratulations, you just finished configuring subcategories in Magento 2. Now let's open up a browser and surf to the domain names you created earlier: yourdomain.com, yourdomain.de, and yourdomain.fr. Your categories should now look as follows:

The screenshot shows the 'Root English (ID: 2)' category configuration page. At the top, there are buttons for 'Add Root Category' and 'Add Subcategory'. Below these are tabs for 'General Information' (which is active), 'Display Settings', 'Custom Design', and 'Category Products'. On the left, a sidebar displays a tree structure of categories: Root English (0) has Shoes (0) and Clothes (0); Shoes (0) has Men (0) and Women (0); Clothes (0) has Men (0) and Women (0). Root German (0) has Schuhe (0) and Kleider (0); Schuhe (0) has Männer (0) and Frau (0); Kleider (0) has Männer (0) and Frau (0). Root French (0) has Chaussures (0) and Vêtements (0); Chaussures (0) has Hommes (0) and Femmes (0); Vêtements (0) has Hommes (0) and Femmes (0). To the right of the sidebar, the 'Name' field is set to 'Root English', the 'Is Active' field is set to 'Yes', and the 'Description' field contains a WYSIWYG editor. At the bottom, there is an 'Image' section with a placeholder 'Bestand kiezen' and a note 'Geen bestand gekozen'.

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 4, we created subcategories for the English, German, and French stores. In this recipe, we created a dedicated Root Catalog for every website. This way, every store can be configured using their own tax and shipping rules.

There's more...

In our example, we only submitted **Name**, **Is Active**, and **Page Title**. You may continue to commit the **Description**, **Image**, **Meta Keywords**, and **Meta Description** fields. By default, the URL key is similar to the **Name** field; you can change this depending on your SEO needs.

Every category or subcategory has a default page layout defined by the theme. You may need to override this. Go to the **Custom Design** tab and click the **Page Layout** drop-down menu. We can choose from the following options: **1 column**, **2 columns with left bar**, **2 columns with right bar**, **3 columns**, and **Empty layout**.

Manage attribute sets

Every product has a unique DNA; some, such as shoes, could have different colors, brands, and sizes, while a snowboard could have weight, length, torsion, manufacturer, and style.

Setting up a website with all the attributes does not make sense. Depending on the products you sell, you should create attributes specific to each website.

When creating products for your website, attributes are the key element and need to be thought through. What and how many attributes do you need? And how many values do you need? These are all types of question that could have a great impact on your website; and don't forget performance. Creating an attribute such as color and having 100,000 of different key values stored will not improve your overall speed and user experience. Always think things through.

After creating the attributes, we combine them in attribute sets, which can be picked when starting to create a product. Some attributes can be used more than once, while others are unique to one product or attribute set.

Getting ready

To step through this recipe, you will use a Droplet created in *Chapter 2, Magento 2 System Tools*, at DigitalOcean (<https://www.digitalocean.com/>). We will be using an NGINX, PHP-FPM, Composer-based setup with Magento 2 preinstalled. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to create product attributes and sets. The following steps will guide you through them:

1. First, log in to the backend of Magento 2 and go to **Stores | Products**.

Since we are using a vanilla setup, only system attributes and one attribute set are installed. Now click on **Add New Attribute** and commit the following data in the **Properties** tab:

Attribute Properties	
Default label	shoe_size
Catalog Input Type for Store Owners	Drop-down
Values Required	No

Manage Options (values of your attribute)			
English	Admin	French	German
4	4	35	35
4.5	4.5	35	35
5	5	35-36	35-36
5.5	5.5	36	36
6	6	36-37	36-37
6.5	6.5	37	37
7	7	37-38	37-38
7.5	7.5	38	38
8	8	38-39	38-39
8.5	8.5	39	39

Advanced Attribute Properties	
Scope	Global
Unique Value	No
Add to Column Options	Yes
Use in Filter Options	Yes

 Since we have already set up a multi-website selling shoes and clothes, we will stick with this. The attributes we need for selling shoes are: `shoe_size`, `shoe_type`, `width`, `color`, `gender`, and `occasion`.

Continue the rest of the chart accordingly (<http://www.shoessizingcharts.com>).

- Click on **Save and Continue Edit** now and continue on the **Manage Labels** tab with the following information:

Manage Titles (Size, Color, and so on)		
English	French	German
Size	Taille	Größe

- Click on **Save and Continue Edit** now and continue on the **Storefront Properties** tab with the following information:

Storefront Properties	
Use in Search	No
Comparable in Storefront	No
Use in Layered Navigation	Filterable (with result)
Use in Search Result Layered Navigation	No

Storefront Properties	
Position	0
Use for Promo Rule Conditions	No
Allow HTML Tags on Storefront	Yes
Visible on Catalog Pages on Storefront	Yes
Used in Product Listing	No
Used for Sorting in Product Listing	No

4. Click on **Save Attribute** now and clear the cache. Depending on whether you set up index management accordingly through the Magento 2 cronjob, it will automatically update the newly created attribute.
5. The configuration for the additional shoe_type, width, color, gender, and occasion attributes can be downloaded at <https://github.com/mage2cookbook/chapter4>.
6. After creating all of the attributes, we combine them in an attribute set called **Shoes**. Go to **Stores | Attribute Set**, click **Add Attribute Set**, and commit the following data:

Edit Attribute Set Name	
Name	Shoes
Based On	Default

7. Now click in the **Groups** section, click on the **Add New** button, and commit the group name called **Shoes**.
8. The newly created group is now located at the bottom of the list. You may need to scroll down before you see it. It is possible to drag and drop the group higher up in the list.
9. Now drag and drop the created shoe_size, shoe_type, width, color, gender, and occasion attributes in the group and save the configuration. The cronjob notice is automatically updated, depending on your settings.

10. Congratulations, you just finished creating attributes and attribute sets in Magento 2. These can be seen in the following screenshot:

The screenshot shows the 'Edit Attribute Set Name' interface. The 'Name' field is set to 'Shoes'. Below it, there's a note 'For internal use'. In the 'Groups' section, there are two buttons: 'Add New' and 'Delete Selected Group'. A note says 'Double click on a group to rename it.' The 'Autosettings' group contains attributes like 'custom_design_to', 'custom_layout_update', 'page_layout', 'options_container', 'short_description', 'visibility', 'news_from_date', 'news_to_date', 'country_of_manufacture', 'gift_message_available', 'gift_wrapping_available', 'gift_wrapping_price', and 'is_returnable'. A newly created group named 'Shoes' is highlighted in yellow and contains attributes: 'occasion', 'width', 'shoe_size', 'shoe_type', 'color', and 'gender'. In the 'Unassigned Attributes' section, there is one attribute: 'manufacturer'.

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 10, we created attributes that will be used in an attribute set. The attributes and sets are the fundamentals for every website.

In steps 1 through 5, we created multiple attributes to define all details about the shoes and clothes we would like to sell. Some attributes are later used as configurable values on the frontend while others only indicate the gender or occasion.

In steps 6 through 9, we connect the attributes to the related attribute set. Thus, when creating a product, all the correct elements are available.

There's more...

After creating the Shoe attribute set, continue by creating an attribute set for Clothes.

Use the following attributes to create the set: color, occasion, apparel_type, sleeve_length, fit, size, length, and gender.

Follow the same steps we performed before to create a new attribute set. You may reuse the color, occasion, and gender attributes. Details of all the attributes can be found here: <https://github.com/mage2cookbook/chapter4#clothes-set>.

The following screenshot shows the **Clothes** attribute set:

Edit Attribute Set Name		Groups	Unassigned Attributes
Name *	<input type="text" value="Clothes"/>	Add New Delete Selected Group	<input type="checkbox"/> manufacturer <input type="checkbox"/> shoe_size <input type="checkbox"/> shoe_type <input type="checkbox"/> width
For internal use		Double click on a group to rename it.	
		<ul style="list-style-type: none"> <input type="checkbox"/> page_layout <input type="checkbox"/> options_container <input type="checkbox"/> Autosettings <ul style="list-style-type: none"> <input type="checkbox"/> short_description <input type="checkbox"/> visibility <input type="checkbox"/> news_from_date <input type="checkbox"/> news_to_date <input type="checkbox"/> country_of_manufacture <input type="checkbox"/> gift_message_available <input type="checkbox"/> gift_wrapping_available <input type="checkbox"/> gift_wrapping_price <input type="checkbox"/> is_returnable <input checked="" type="checkbox"/> Clothes <ul style="list-style-type: none"> <input type="checkbox"/> color <input type="checkbox"/> occasion <input type="checkbox"/> apparel_type <input type="checkbox"/> sleeve_length <input type="checkbox"/> fit <input type="checkbox"/> size <input type="checkbox"/> length <input type="checkbox"/> gender 	

Create products

Eventually, after creating attributes and sets, it comes down to adding products. Magento 2 uses the same types of product as Magento 1. This also includes Magento 2 Enterprise.

The product types we can choose from are: Simple Product, Configurable Product, Grouped Product, Virtual Product, Bundle Product, Downloadable Product, and, for the **Enterprise Edition (EE)**, Gift Card.

Depending on the products you would like to sell, you may use one or two of the types. The most used types are Simple and Configurable Products because they rely on one another when you sell shoes, for example.

Product type definitions are as follows:

- ▶ **Simple Product:** A Simple Product in Magento is a physical product. There are no options such as size or color that the end user can pick during the order. One example of a Simple Product type is a broom or umbrella.
- ▶ **Configurable Product:** A combination of Simple Products organized with different colors, sizes, or other attributes is called a Configurable Product. One example of a Configurable Product is a shoe or shirt.
- ▶ **Grouped Product:** A Grouped Product is a collection of Simple Products related to one another. Each Simple Product could be sold separately, but is cheaper as a set. One example of a Grouped Product is a camera plus a photo bag and memory card. This set might offer a special price.
- ▶ **Virtual Product:** A Virtual Product is a non-physical product. One example of a Virtual Product is a service warranty for your computer or a membership.
- ▶ **Bundle Product:** A Bundle Product is an extension of a Grouped Product. A Grouped Product does not have the option to configure different choices. But this can be managed using a Bundle Product. One example of a Bundle Product is a building a computer; a customer can choose from a set of different hard disks, monitors, CPUs, memory, and so on.
- ▶ **Downloadable Product:** A Downloadable Product is a non-physical product. One example of a Downloadable Product is software or an eBook; you can download them online.
- ▶ **Gift Card (EE only):** A Gift Card can be a physical, virtual, or combined product. This is used as a store credit and can be sold as a gift.

Besides the regular ones, it is possible you may need extra options, depending on your product base. In Magento 1, additional third-party modules, such as Configurable Bundles, Events, Training, Rental, and Recurring, could be bought and installed to manage this.

Using product types is now easier than ever. Magento 2 created a user-friendly flow for configuring Configurable Products on the fly.

Getting ready

To step through this recipe, you will use a Droplet created in *Chapter 2, Magento 2 System Tools*, at DigitalOcean (<https://www.digitalocean.com/>). We will be using an NGINX, PHP-FPM, Composer-based setup with a single Magento 2 website, Root Catalog, store view, categories, and attributes preinstalled. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to create Configurable Product for Magento 2. The following steps will guide you through them:

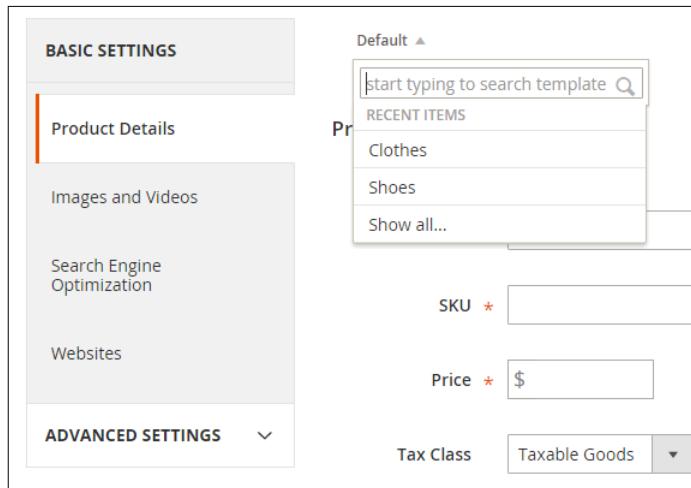
1. First, log in to the backend of Magento 2 and go to **Products | Catalog**. Click on the **Add Product** button and continue with the following information:

Product Details	
Name	Ellis Flat
SKU	shw005
Price	250.00
Tax Class	Taxable Goods
Images and Videos	Download the images here: https://github.com/mage2cookbook/chapter4
Quantity	100
Weight	Yes
Categories	Shoes
Description	Suede upper. Rubber 0.5" heel. Domestic.

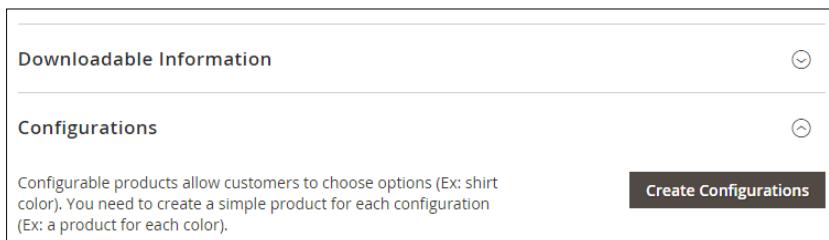
2. Now hit the **Save** button. As you can see, you stay in the same screen while your data is being saved. When you want to **Save & Close** then choose from the drop-down arrow and continue.
3. Open a new tab in your browser, click the drop-down arrow in the top-right corner, and choose **Customer View**. This trick will open up a new tab and launch your home page.
4. Go to your **Shoes** menu and the newly created product should be visible. In some situations, it is best to clear your cache first if you are having issues.

- Congratulations, you just finished creating a Simple Product in Magento 2.

Now let's go back to our backend and open up the newly created product. Next we are going to set our attribute set we created earlier in this chapter. This option is located above the **Product Details** title. Here we choose the **Shoes** attribute set, as shown in the following screenshot:



- After choosing the option, you will see that Magento 2 has loaded a new menu called **Shoes** under the **Search Engine Optimization** menu.
- Next scroll to the bottom and open up the **Configurations** drop-down menu. Click the **Create Configurations** button. This is shown in the following screenshot:



8. Next we will use one of the brand new features of Magento 2. This workflow helps to create Configurable Product on the fly. Depending on the attributes, the list of attributes could be long. For now, we pick the **shoe_size** option. Continue to the next step by hitting the **Next** button in the top-right corner, as shown in the following screenshot:

The screenshot shows the 'Create Product Configurations' interface. At the top, there's a yellow info bar stating: 'When you remove or add an attribute, we automatically update all configurations and you will need to manually recreate the current configurations.' Below this is a progress bar with four steps: 'Select Attributes' (highlighted), 'Attribute Values', 'Bulk Images & Price', and 'Summary'. To the right are 'Cancel', 'Back', and 'Next' buttons.

Step 1: Select Attributes

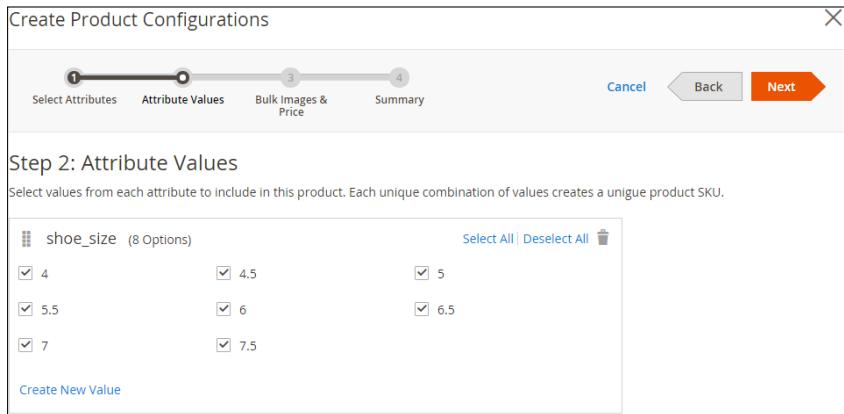
Selected Attributes: Size

Filters Default View Columns

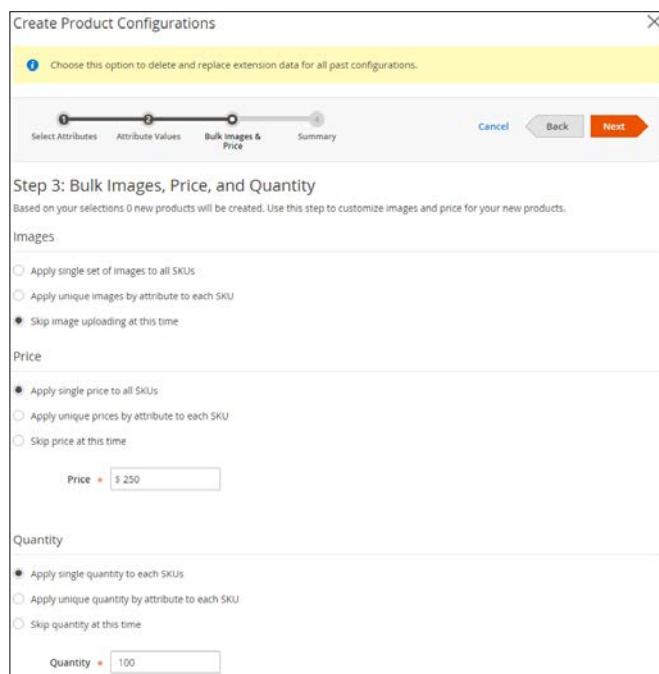
10 records found (1 selected) 20 per page 1 of 1

-	Use in Layered Navigation	Attribute Code	Attribute Label	Required	System	Visible	Scope	Searchable	Comparable
<input type="checkbox"/>	Filterable (with results)	apparel_type	apparel_type	Yes	Yes	Yes	Global	Yes	Yes
<input type="checkbox"/>	Filterable (with results)	color	Color	No	Yes	Yes	Global	Yes	Yes
<input type="checkbox"/>	Filterable (with results)	fit	fit	No	Yes	Yes	Global	Yes	No
<input type="checkbox"/>	No	gender	gender	No	Yes	Yes	Global	Yes	No
<input type="checkbox"/>	Filterable (with results)	length	length	No	Yes	Yes	Global	Yes	No
<input type="checkbox"/>	Filterable (with results)	occasion	occasion	No	Yes	Yes	Global	Yes	Yes
<input checked="" type="checkbox"/>	Filterable (with results)	shoe_size	shoe_size	No	Yes	Yes	Global	No	No
<input type="checkbox"/>	Filterable (with results)	shoe_type	shoe_type	Yes	Yes	Yes	Global	Yes	No
<input type="checkbox"/>	Filterable (with results)	size	size	No	Yes	Yes	Global	Yes	No
<input type="checkbox"/>	Filterable (with results)	sleeve_length	sleeve_length	No	Yes	Yes	Global	Yes	No

9. A list shows all the attribute values we created earlier. Click **Select All** and continue to the next step, as shown in the following screenshot:



10. **Step 3: Bulk Images, Price, and Quantity** to create a Configurable Product is an important step. Commit a price and quantity here, otherwise the product will not show up on the screen (adjusting the value later is straightforward using the same flow). Select **Apply single price to all SKUs** and **Apply single quantity to each SKU** and fill in 250 for the **Price** and 100 for the **Quantity**. Continue to the next step, as shown in the following screenshot:



11. Depending on the values in the attribute list, **Associated Products** are created. Now click the **Generate Products** button and all of them are created, as shown in the following screenshot:

Create Product Configurations X

① Select Attributes ② Attribute Values ③ Bulk Images & Price Summary

[Cancel](#) [Back](#) [Generate Products](#)

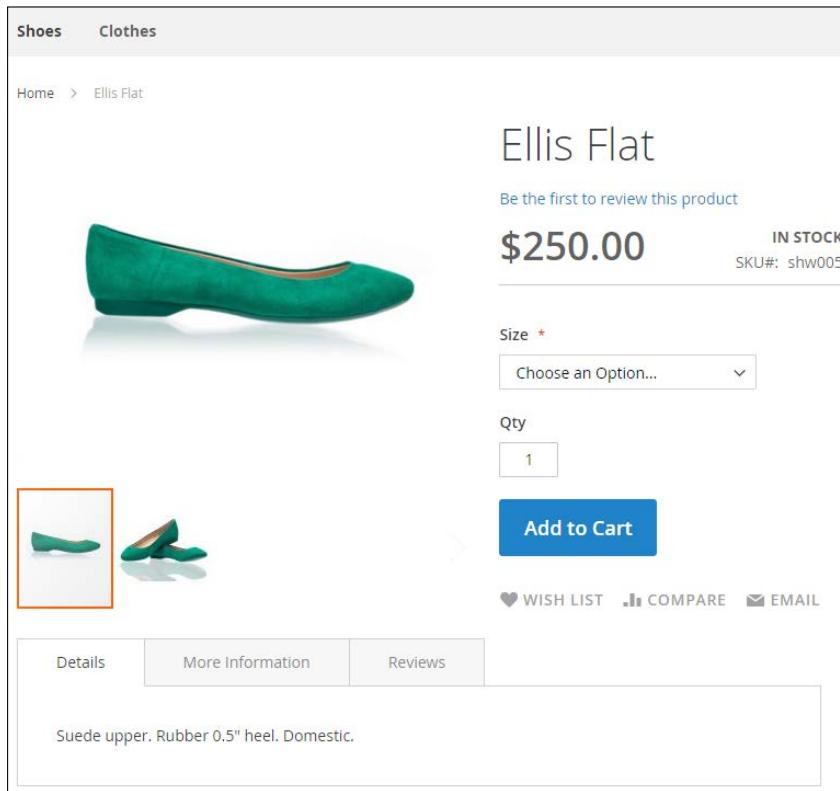
Step 4: Summary

Associated Products (1)

You created these products for this configuration.

Images	SKU	Quantity	shoe_size	Price
	shw005-4	100	4	\$ 250
	shw005-4.5	100	4.5	\$ 250
	shw005-5	100	5	\$ 250
	shw005-5.5	100	5.5	\$ 250
	shw005-6	100	6	\$ 250
	shw005-6.5	100	6.5	\$ 250
	shw005-7	100	7	\$ 250
	shw005-7.5	100	7.5	\$ 250

12. Now save the new setup and check in your browser; the result should be as shown in the following screenshot:



13. Congratulations, you just finished creating a Configurable Product in Magento 2.
14. Now go to <https://github.com/mage2cookbook/chapter4#creating-products> and continue to the rest of the **Shoes** and **Clothes** products. After creating a minimum of two products, our catalog navigation filter (layered navigation) pops up and looks like this:

The screenshot shows a Magento storefront for 'Shoes'. At the top, there's a navigation bar with 'LUMA' logo, 'Shoes', and 'Clothes' links. Below it, a breadcrumb trail shows 'Home > Shoes'. The main title 'Shoes' is centered above a green notification bar that says 'You added product Ellis Flat to the comparison list.' with a checkmark icon. To the left, a 'Shopping Options' sidebar lists filters for 'PRICE', 'COLOR', 'SHOE TYPE', and 'OCCASION'. To the right, there are two shoe products displayed: a green flat and a leopard-print pump. Below the products is a 'Compare Products' section showing two items: 'Ellis Flat' and 'Broadway Pump'. Each item has a small image, the name, and a price (\$250.00 for Ellis Flat, \$410.00 for Broadway Pump). At the bottom of this section are 'Compare' and 'Clear All' buttons.

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 13, we created a Configurable Product using the new Magento workflow.

In steps 1 through 4, we created a Simple Product and connected it to the attribute set. This product will be the starting point for creating a Configurable Product.

In steps 8 through 11, we used the configurations workflow to select the attributes related to this product and set its values. Depending on the setup, we can apply a single price, image, or quantity to all of the Simple Products created during this process.

There's more...

By default, when creating a product, Magento 2 starts with a Virtual Product. When changing settings such as **Weight** (*Does this have a weight?*) to **Yes**, the product switches to a Simple Product. The same goes for Downloadable and Configurable Product.

This new technique is stunning and a great benefit for store owners. Everybody can now create and change products on the fly.

Bundle and Grouped Products, on the other hand, are more like Magento 1. You first need to choose the product type using the drop-down arrow in the **Add Product** button and then continue the same flow, as shown in the following screenshot:



Manage products in a catalog grid

Managing products on a daily basis may not be one of the most entertaining tasks. The product grid in Magento 1 is, out of the box, not the best tool. Lots of merchants install a third-party extension for better use and configuration. Depending on the extension, it is possible to tune the grid accordingly.

Now the new Magento 2 catalog grid is better than ever. Every backend user can create their own view and select the appropriate attributes for the best view ever.

Besides all the fancy features, the product grid loads asynchronously, which means that the page refreshes its data in the background. This is great for performance and user experience.

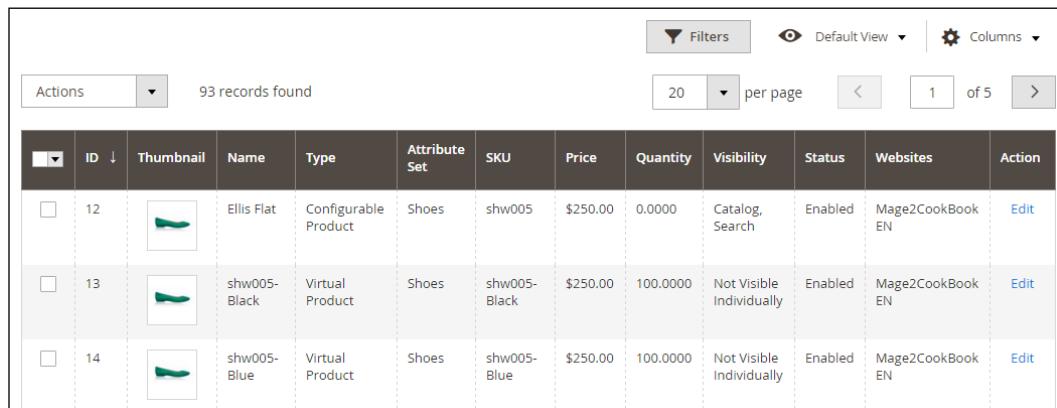
Getting ready

To step through this recipe, you will use a Droplet created in *Chapter 2, Magento 2 System Tools*, at DigitalOcean (<https://www.digitalocean.com/>). We will be using an NGINX, PHP-FPM, Composer-based setup including a single Magento 2 website, Root Catalog, store view, categories, and attributes and four configurable products preinstalled. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to create a custom product grid in Magento 2. The following steps will guide you through this:

- First, log in to the backend of Magento 2 and go to **Products | Catalog**. Depending on the previous recipe, you will have a list which looks as follows:



The screenshot shows the Magento 2 Admin Product Grid interface. At the top, there are buttons for 'Filters', 'Default View', and 'Columns'. Below that, there are dropdowns for 'Actions' (set to 'Actions'), 'per page' (set to 20), and 'of 5' (page 1). The grid itself has columns for ID, Thumbnail, Name, Type, Attribute Set, SKU, Price, Quantity, Visibility, Status, Websites, and Action. The first row contains a checkbox, ID 12, thumbnail of a green shoe, name 'Ellis Flat', type 'Configurable Product', attribute set 'Shoes', SKU 'shw005', price '\$250.00', quantity '0.0000', visibility 'Catalog, Search', status 'Enabled', websites 'Mage2CookBook EN', and an 'Edit' button. The second row contains a checkbox, ID 13, thumbnail of a green shoe, name 'shw005-Black', type 'Virtual Product', attribute set 'Shoes', SKU 'shw005-Black', price '\$250.00', quantity '100.0000', visibility 'Not Visible Individually', status 'Enabled', websites 'Mage2CookBook EN', and an 'Edit' button. The third row contains a checkbox, ID 14, thumbnail of a green shoe, name 'shw005-Blue', type 'Virtual Product', attribute set 'Shoes', SKU 'shw005-Blue', price '\$250.00', quantity '100.0000', visibility 'Not Visible Individually', status 'Enabled', websites 'Mage2CookBook EN', and an 'Edit' button.

Actions	ID	Thumbnail	Name	Type	Attribute Set	SKU	Price	Quantity	Visibility	Status	Websites	Action
<input type="checkbox"/>	12		Ellis Flat	Configurable Product	Shoes	shw005	\$250.00	0.0000	Catalog, Search	Enabled	Mage2CookBook EN	Edit
<input type="checkbox"/>	13		shw005-Black	Virtual Product	Shoes	shw005-Black	\$250.00	100.0000	Not Visible Individually	Enabled	Mage2CookBook EN	Edit
<input type="checkbox"/>	14		shw005-Blue	Virtual Product	Shoes	shw005-Blue	\$250.00	100.0000	Not Visible Individually	Enabled	Mage2CookBook EN	Edit

- Click in the top-right corner on the arrow to the right of **Columns**. By default, Magento lists 12 options. Now select **URL key**, and deselect the **SKU**, **Visibility**, and **Websites** checkboxes. Then point your mouse at the grid again. Be aware there is no **Save** button. The grid should look as follows:

The screenshot shows the 'Columns' configuration dialog for a grid. It lists 10 out of 43 visible columns. The checked columns are: ID, Type, Price, Status, Special Price, Cost, Meta Keywords, Thumbnail, Attribute Set, Quantity, Websites, Weight, and Meta Description. The unchecked columns are: SKU, Visibility, Short Description, and Meta Title. At the bottom of the dialog are 'Reset' and 'Cancel' buttons.

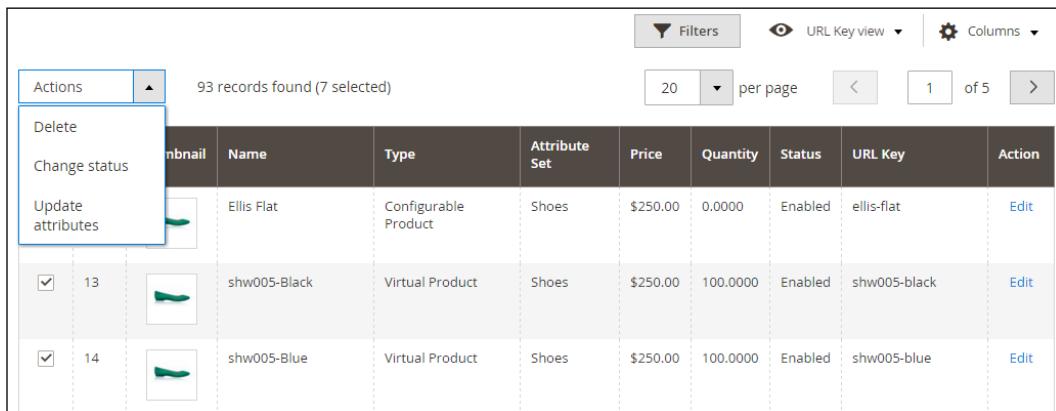
ID	Thumbnail	Name	Type
12		Ellis Flat	Configurable Product
13		shw005-Black	Virtual Product
14		shw005-Blue	Virtual Product
15		shw005-Brown	Virtual Product

- Now click on the **Default View** button, and click **Save View As....**. Pick a name and click the right arrow to save your work. The grid view should now look like this:

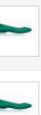
The screenshot shows the grid with a new 'Default View' applied. The columns are: ID, Thumbnail, Name, Type, Attribute Set, Price, Quantity, Status, URL Key, and Action. The first two rows of data are identical to the previous screenshot.

ID	Thumbnail	Name	Type	Attribute Set	Price	Quantity	Status	URL Key	Action
12		Ellis Flat	Configurable Product	Shoes	\$250.00	0.0000	Enabled	ellis-flat	Edit
13		shw005-Black	Virtual Product	Shoes	\$250.00	100.0000	Enabled	shw005-black	Edit

4. You can create as many views as necessary. Keep in mind that all created views only apply to the user who has created them. Currently, there is no shareable grid view option.
5. If you ever need to update the status of **Enabled** or **Disabled**, go to the left drop-down **Actions** menu and choose **Change status**.
6. Use the **Update attributes** option in the drop-down **Actions** menu to update one of the attributes. First select the products which you need to update. Then click on **Update attributes**, as shown in the following screenshot:



The screenshot shows a product grid with the following columns: Actions, Thumbnail, Name, Type, Attribute Set, Price, Quantity, Status, URL Key, and Action. There are 93 records found (7 selected). The 'Actions' dropdown menu is open, showing options: Delete, Change status, and Update attributes. The 'Update attributes' option is highlighted with a blue border. Two products are selected, indicated by checked checkboxes in the first column. The selected products are: Product ID 13, Name shw005-Black, Type Virtual Product, Attribute Set Shoes, Price \$250.00, Quantity 100.0000, Status Enabled, URL Key shw005-black; and Product ID 14, Name shw005-Blue, Type Virtual Product, Attribute Set Shoes, Price \$250.00, Quantity 100.0000, Status Enabled, URL Key shw005-blue.

Actions	Thumbnail	Name	Type	Attribute Set	Price	Quantity	Status	URL Key	Action	
Delete										
Change status										
Update attributes		Ellis Flat	Configurable Product	Shoes	\$250.00	0.0000	Enabled	ellis-flat	Edit	
<input checked="" type="checkbox"/>	13		shw005-Black	Virtual Product	Shoes	\$250.00	100.0000	Enabled	shw005-black	Edit
<input checked="" type="checkbox"/>	14		shw005-Blue	Virtual Product	Shoes	\$250.00	100.0000	Enabled	shw005-blue	Edit

7. Next go to **shoe_type**, mark checkbox and select the appropriate option, and click **Save**.
8. Congratulations, you just finished managing the product grid catalog in Magento 2.

How it works...

Let's recap and find out what we did throughout this recipe. In steps 1 through 8, we created a custom grid for the catalog view. You can create as many grids as you like and select your preferred attributes in it. Saving the new grid views is straightforward.

There's more...

If you like managing attributes in your grid or columns, go to **Stores | Product** and select one of the attributes. In the **Properties** tab, look for **Advanced Attribute Properties**. At the bottom, change **Add to Column Options** and **Use in Filter Options** appropriately, as shown in the following screenshot:

Advanced Attribute Properties

Attribute Code	color
This is used internally. Make sure you don't use spaces or more than 30 symbols.	
Scope	Global ▾
Declare attribute value saving scope	
Unique Value	No ▾
Not shared with other products	
Input Validation for Store Owner	None ▾
Add to Column Options	Yes ▾
Select "Yes" to add this attribute to the list of column options in the product grid.	
Use in Filter Options	Yes ▾
Select "Yes" to add this attribute to the list of filter options in the product grid.	

5

Managing Your Store

In this chapter, we will cover the basic tasks related to creating a catalog and products in Magento 2. You will learn the following:

- ▶ Creating shipping and tax rules
- ▶ Managing customer groups
- ▶ Configuring inventories
- ▶ Configuring currency rates
- ▶ Managing advanced pricing

Introduction

Although we've created categories and products, we are not yet ready to go online. Depending on the country or state we live in and ship to, we have to levy additional charges such as VAT and shipping fees.

These shipping fees and tax rates need to be configured correctly according to the website or store view. Shipping options can be straightforward from free shipping to advanced calculations. But keep in mind that, depending on the situation, it is not as easy as it looks and could take some time to set up.

Besides shipping and tax, we should not forget the inventory. Without the correct inventory setup, we could create issues when it comes to stock management. Magento 2 uses the same inventory setup as Magento 1, and is straightforward to configure out of the box.

Are you selling products overseas and need to use different currency rates? Magento 2 is your best friend. This functionality in a multi-store setup is easy to configure.

Some products may depend on special prices related to customer groups. Creating B2C or B2B groups is straightforward and can be connected to advanced pricing within the product types. These prices will be shown after login or store view.



Lots of system configuration features are basically the same as in Magento 1. Within this chapter we will cover the basics and show Magento 2-specific features when they apply.

Creating shipping and tax rules

The topic of shipping is so huge that you could write a book about it. The options related, for example, to width, length and height, breakable, edible, and so on, are endless.

After configuring these attributes for a product, we can start relating them to our shipping setup and shipping vendor. Magento has a huge selection of shipping vendors to choose from. It's important to choose the right vendor and the correct Magento extension. This could be challenging. Do not immediately pick the cheapest shipping vendor. Check the quality of their service, their specialty when it comes to shipping your products, and their Magento extension.

Creating the correct tax rules is not for the fainthearted. Are you in the USA, Europe, Asia, or somewhere else? Every country has its own tax rules. Always check with the local authorities to find out which rules to apply.

Getting ready

To step through this recipe, you will use a Droplet created in *Chapter 2, Magento 2 System Tools* at DigitalOcean <https://www.digitalocean.com/>. We will be using an NGINX, PHP-FPM, Composer-based setup including sample data. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to create shipping and tax rules for the European Union. The shipping rules apply to a Table Rates setup using a local shipping vendor. The following steps will guide you through them.

1. First we start setting up the shipping rates. Go to **Stores | Configuration | Sales**. We have three menus to choose from. Let's start with the **Shipping Settings** first. Click on the **Menu** tab. You see two drop-down menus called **Origin** and **Shipping Policy Parameters**.

Now complete the entire field set related to your company. This is the starting point. When using the **Shipping Policy**, just mark it **Yes** and commit your policy. Here is an example of how a policy could look:



Please be assured that your items will ship out within two days of purchase. We determine the most efficient shipping carrier for your order. The carriers that may be used are: TNT, DHL, United Parcel Service (UPS), or FedEx. Sorry but we cannot ship to P.O. Boxes.

Origin

Country	<input type="text" value="Netherlands"/>	[WEBSITE]
Region/State	<input type="text"/>	[WEBSITE]
ZIP/Postal Code	<input type="text" value="1011AC"/>	[WEBSITE]
City	<input type="text" value="Amsterdam"/>	[WEBSITE]
Street Address	<input type="text" value="My Street 123"/>	[WEBSITE]
Street Address Line 2	<input type="text"/>	[WEBSITE]

Shipping Policy Parameters

Apply custom Shipping Policy	<input type="text" value="Yes"/>	[WEBSITE]
Shipping Policy	<input type="text" value="Please be assured that your items will ship out within two days of purchase. We determine the most efficient shipping carrier for your order. The carriers that may be used are: U.S. Postal Service"/> <div style="float: right; width: 20px; height: 20px; background-color: #ccc; margin-top: -10px;"></div> [STORE VIEW]	

2. Now continue to the **Multishipping Settings** menu. By default, we stay with the **Allow Shipping to Multiple Addresses** option.
3. Next, we click **Shipping Methods**. The default shipping options in Magento 2 are: **Free Shipping, Flat Rate, Table Rates, UPS, USPS, FedEx, and DHL**.

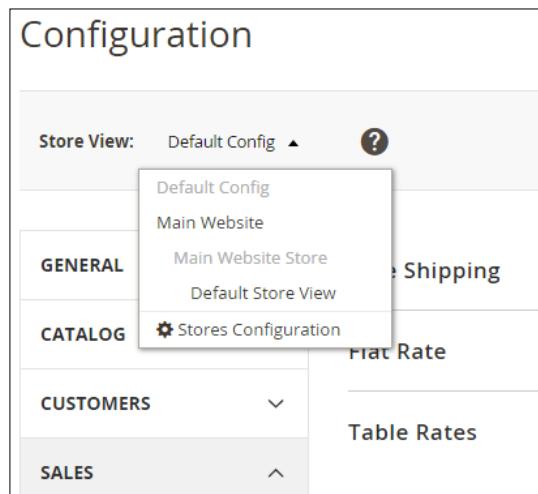
Since the scope of this recipe is **Table Rates**, using **Free Shipping** and **Flat Rate** is pretty straightforward.

Now click on the **Table Rates** drop-down arrow, and commit the following information:

Table Rates		
Enabled	Yes	[WEBSITE]
Title	Best Way	[STORE VIEW]
Method Name	Table Rate	[STORE VIEW]
Condition	Weight vs. Destination	[WEBSITE]
Include Virtual Products in Price Calculation	No	[WEBSITE]
Calculate Handling Fee	Fixed	[WEBSITE]
Handling Fee		[WEBSITE]
Displayed Error Message	This shipping method is not available. To use this shipping method, please contact us.	[STORE VIEW]
Ship to Applicable Countries	All Allowed Countries	[WEBSITE]
Ship to Specific Countries	Afghanistan Åland Islands Albania Algeria American Samoa Andorra Angola Anguilla Antarctica Antigua and Barbuda	[WEBSITE]
Show Method if Not Applicable	No	[WEBSITE]
Sort Order		[WEBSITE]

In the **Condition** drop-down menu we use the **Weight vs. Destination** option. Beside this option we also can choose from **Price vs. Destination** or **# (number) of items vs. Destination**. Depending on your needs, pick one of them. Since we are using the **Weight** option, we need to make sure that our entire product set has the correct weight configured.

4. For the purpose of this recipe, disable the **Flat Rate** option in the menu.
5. Now click **Save Config** and update your cache.
6. Next we need to switch to the correct website using the **Store View** switcher in the **Stores | Configuration** menu. Click the drop-down arrow in the top-left menu, and select **Main Website** (or the name of your website):



7. Confirm the pop-up window to continue and check the **Table Rates** options. Now we have two new options visible. The **Export CSV** gives us a comma-separated file called `tablesrates.csv` that we need to complete. Download the file and open up a spreadsheet editor, such as MS Excel, OpenOffice Calc, or Google Docs Spreadsheet.

Since we are using the **Weight vs. Destination** option, the CSV schema looks as follows:

Country	Region/State	Zip/Postal Code	Weight (and above)	Shipping Price
NLD	*	*	0	6.95
NLD	*	*	50	9.95
NLD	*	*	100	14.50
DEU	*	*	0	10.50
DEU	*	*	50	17.50
DEU	*	*	100	22.50

Country	Region/State	Zip/Postal Code	Weight (and above)	Shipping Price
FRA	*	*	0	10.50
FRA	*	*	50	17.50
FRA	*	*	100	22.50

In this example, we use a wildcard for the **Region/State** and **Zip/Postal Code**. You can replace this wildcard with the appropriate value. Upload your saved `tablesrates.csv` file in the **Import** section and click **Save Config**, and clean the cache:

The screenshot shows a configuration interface for shipping rates. It includes fields for 'Condition' (set to 'Weight vs. Destination'), 'Use Default' checkboxes for 'Weight vs. Destination' and 'Include Virtual Products in Price Calculation', an 'Export CSV' button, an 'Import' section with a dropdown for 'Bestand kiezen' (selected to 'Geen bestand gekozen'), and a 'WEBSITE' link.

- Before we can verify it is working, we need to update the weight of the product we want to sell. Go to **Products | Catalog** and update your grid using the weight attribute. Check out the *Manage products in a catalog grid* recipe of Chapter 4, *Creating Catalogs and Categories* for how to do this.
- Now let's edit **Joust Duffle Bag** from the sample data. Set the **Weight** to 50 and click **Save & Close**. Do the same for **Strive Shoulder Pack** (49) and **Crown Summit Backpack** (51). Your product grid should now look as follows:

ID	Thumbnail	Name	Type	Price	Quantity	Status	Weight	Action
1		Joust Duffle Bag	Simple Product	\$34.00	100.0000	Enabled	50.0000	Edit
2		Strive Shoulder Pack	Simple Product	\$32.00	100.0000	Enabled	49.0000	Edit
3		Crown Summit Backpack	Simple Product	\$38.00	100.0000	Enabled	51.0000	Edit

10. Finally, we can test if the checkout and shipping fee are configured correctly. Open up a browser, add the **Joust Duffle Bag** to you basket, and check it out. Complete your personal data and check the shipping **Table Rate** at the bottom.

We only used German, French, and Dutch codes in this example. If you want to have your country shipping fees in the **Table Rate** CSV file, update them accordingly:

Shipping Address

Email Address *

 [?](#)

You can create an account after checkout.

First Name *

Last Name *

Company

Street Address *

City *

State/Province *

 [▼](#)

Zip/Postal Code *

Country *

 [▼](#)

Phone Number *

 [?](#)

Order Summary

1 Item in Cart [^](#)

	Joust Duffle Bag	\$34.00
	Qty: 1	

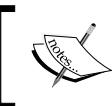
Shipping Methods

[See our Shipping Policy](#)

\$17.50 Table Rate Best Way

[Next](#)

11. Congratulations, you just finished configuring shipping rules in Magento 2.
12. Next we continue to configure the appropriate tax rules. Since we cannot cover all the different tax rules worldwide, we will stick for now with the European Union. Import to the following **tax_rates.csv** file to **System | Import/Export Tax Rates**. The file can be downloaded from <https://github.com/mage2cookbook/chapter5>.
13. To check if all tax rates are created, go to **Stores | Tax Zone and Rates**. You see a large list of all rates and countries.



All rates apply to the current tax regulation of the European Union with effect from the 1st of January 2015. The calculated tax is based on the country of the seller.

14. Now go to **Stores | Tax Rules** and click on **Rule 1**. For this example we change the **Name** to **EU Customers**. Now let's select all the EU countries with the (standard) tax Rate and click **Save Rule**:

Tax Rule Information

Name *	<input type="text" value="EU Customers"/>
Tax Rate *	<input type="text" value="US-CA-* -Rate 1"/> <input type="text" value="US-NY-* -Rate 1"/> <input type="text" value="US-MI-* -Rate 1"/> <input checked="" type="checkbox"/> Austria (standard) Austria (reduced: food / books / pharma / hotels) <input checked="" type="checkbox"/> Belgium (standard) Belgium (reduced: restaurants) Belgium (reduced: food / books / pharma / medical / hotels) <input checked="" type="checkbox"/> Bulgaria (standard)
Add New Tax Rate	

In this example we set the default rule to the high tax rate. Create a new rule when you are selling services or products that have a lower tax rate. But don't forget to create a new **Product Tax Class** in the **Additional Settings**. This class can then be used in every product type where it applies.

15. Next we need to configure the tax system setup. Go to **Stores | Configuration | Sales | Tax**. Depending on your production setup, using a multi domain with different shipping vendors and warehouse configuration may change. Always use the **Store View** switcher on the top to change the settings according to the domain or country you are selling in. The following examples will give you an overview of a basic setup created in Magento 2 Enterprise Edition. In the Magento 2 Community Edition some features, such as Gift Wrapping and Printed Card Prices, will not be shown:

Tax Classes

Tax Class for Shipping	None	[WEBSITE]
Tax Class for Gift Options	None	[WEBSITE]
Default Tax Class for Product	Taxable Goods	[GLOBAL]
Default Tax Class for Customer	Retail Customer	[GLOBAL]

Calculation Settings

Tax Calculation Method Based On	Row Total	[WEBSITE]
Tax Calculation Based On	Shipping Origin	[WEBSITE]
Catalog Prices	Excluding Tax	[WEBSITE]
This sets whether catalog prices entered from Magento Admin include tax.		
Shipping Prices	Excluding Tax	[WEBSITE]
This sets whether shipping amounts entered from Magento Admin or obtained from gateways include tax.		
Apply Customer Tax	After Discount	[WEBSITE]
Apply Discount On Prices	Including Tax	[WEBSITE]
Apply discount on price including tax is calculated based on store tax if "Apply Tax after Discount" is selected.		
Apply Tax On	Custom price if available	[WEBSITE]
Enable Cross Border Trade	No	[WEBSITE]
When catalog price includes tax, enable this setting to fix the price no matter what the customer's tax rate.		

Default Tax Destination Calculation

Default Country	<input type="text" value="Netherlands"/> [STORE VIEW]
Default State	<input type="text" value="*"/> [STORE VIEW]
Default Post Code	<input type="text"/> [STORE VIEW]

Price Display Settings

Display Product Prices In Catalog	<input type="text" value="Excluding Tax"/> [STORE VIEW]
Display Shipping Prices	<input type="text" value="Excluding Tax"/> [STORE VIEW]

Shopping Cart Display Settings

Display Prices	<input type="text" value="Excluding Tax"/> [STORE VIEW]
Display Subtotal	<input type="text" value="Excluding Tax"/> [STORE VIEW]
Display Shipping Amount	<input type="text" value="Excluding Tax"/> [STORE VIEW]
Display Gift Wrapping Prices	<input type="text" value="Excluding Tax"/> [STORE VIEW]
Display Printed Card Prices	<input type="text" value="Excluding Tax"/> [STORE VIEW]
Include Tax In Order Total	<input type="text" value="Yes"/> [STORE VIEW]
Display Full Tax Summary	<input type="text" value="No"/> [STORE VIEW]
Display Zero Tax Subtotal	<input type="text" value="No"/> [STORE VIEW]

Orders, Invoices, Credit Memos Display Settings		
Display Prices	Including Tax	[STORE VIEW]
Display Subtotal	Including Tax	[STORE VIEW]
Display Shipping Amount	Including Tax	[STORE VIEW]
Display Gift Wrapping Prices	Including Tax	[STORE VIEW]
Display Printed Card Prices	Including Tax	[STORE VIEW]
Include Tax In Order Total	Yes	[STORE VIEW]
Display Full Tax Summary	No	[STORE VIEW]
Display Zero Tax Subtotal	No	[STORE VIEW]

Fixed Product Taxes		
Enable FPT	No	[WEBSITE]
Display Prices In Product Lists	Including FPT and FPT description	[WEBSITE]
Display Prices On Product View Page	Including FPT and FPT description	[WEBSITE]
Display Prices In Sales Modules	Including FPT and FPT description	[WEBSITE]
Display Prices In Emails	Including FPT and FPT description	[WEBSITE]
Apply Tax To FPT	No	[WEBSITE]
Include FPT In Subtotal	No	[WEBSITE]

16. Change the setting and click **Save Config**.
17. Since we are using the Magento 2 sample data, we are ready to perform the test. Every product is configured with the **Taxable Goods Tax Class**.

18. Finally, we can test if the checkout and tax rates are configured correctly. Open up a browser and add the **Joust Duffle Bag** to your basket and check it out. Complete your personal data and check the **Review & Payments** step on the right in the checkout. The **Order Summary** should now look like this:

Order Summary

Cart Subtotal	€ 34,00
Shipping Best Way - Table Rate	€ 9,95
Tax	€ 7,14
Order Total Incl. Tax	€ 51,09
Order Total Excl. Tax	€ 43,95

1 Item in Cart ^

	Joust Duffle Bag	€ 34,00
	Qty: 1	

Ship To:

Ray NL Bogman
Mijn Straat 1
Amsterdam, 1000 AA
Nederland
1234567890

Shipping Method:

Best Way - Table Rate

19. Congratulations, you just finished configuring tax rules in Magento 2.

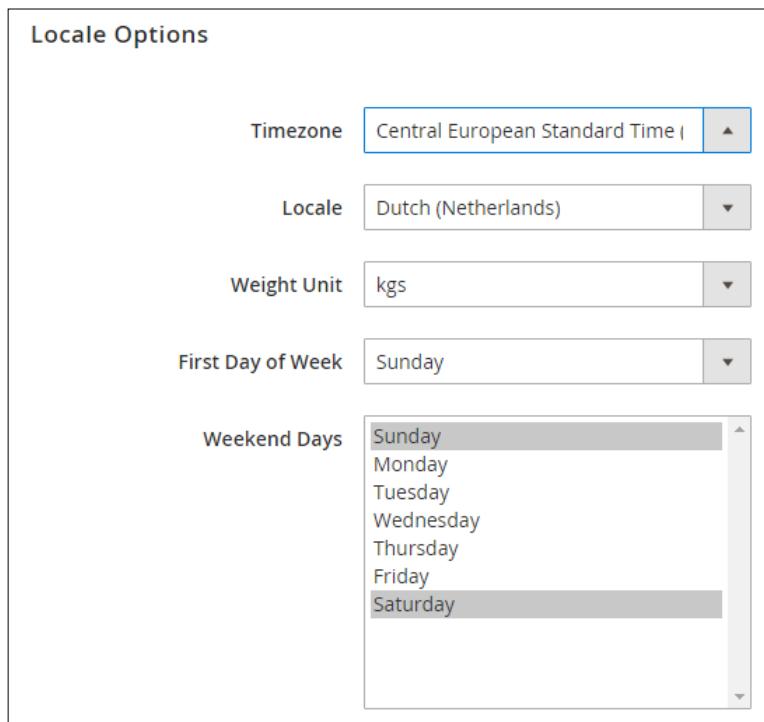
How it works...

Let's recap and find out what we did throughout the preceding recipe. In Steps 1 through 10, we configured a shipping method called **Table Rates** to handle all the shipping. We used the **Weight vs. Destination** option. Using this option we needed to update all our products with the correct weight attribute value.

In Steps 12 through 18, we configured tax rules for the European Union using a **tax_rates.csv** file from GitHub. By using this file, it was easy to configure the appropriate tax rule. In Step 15, we gave an example of how a system configuration for a store view could look.

There's more...

Depending on where you live or are sending products to, using the correct measuring units in Magento 2 is important. This new feature helps us to configure whether we calculate the weight in **lbs** (pounds) or **kgs** (kilograms). We can find this new option in **Stores | Configuration | General | Locale Options**. Here is an example showing the **Weight Unit** field:



Managing customer groups

Everybody knows that every website is different. Some sites only sell products to **Business to Consumer (B2C)** while others sell to **Business to Business (B2B)** or both.

In the B2C market it is pretty common to only have one customer group called **Retailer**. While the B2B market is related to the **Wholesaler**. Beside these two, we could also have groups such as Platinum, Gold, Silver, Special members, or groups based on their location. So the options are unlimited. Setting up the correct infrastructure for your customers helps you to segment these groups and offer them different prices.

Magento offers by default the following groups: General, NOT LOGGED IN, Retailer, and Wholesale.



By default, all groups are related to the Tax Class: Retail Customer. You may need to change this depending on your locale regulations.



Getting ready

To step through this recipe, you will use a Droplet created in *Chapter 2, Magento 2 System Tools at DigitalOcean* <https://www.digitalocean.com/>. We will be using an NGINX, PHP-FPM, Composer-based setup including sample data. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to add additional customer groups. We want to create the following groups: Platinum, Gold, and Silver. The following steps will guide you through this.

1. First login to the backend of Magento, and go to **Stores | Other Settings | Customer Groups**.
2. Click on the **Add New Customer Group** button and create three new groups for Platinum, Gold, and Silver. Connect them to the Tax Class: Retail Customer for now and click **Save Customer Group**.
3. Make sure to update your indexers. When you have your crontab configured correctly, you do not have to worry about this. Magento 2 will update this for you every minute.
4. Now click on **Customers | All Customers** in the default menu. Select all the customers you want to upgrade to another group by marking them on the left of the grid.

5. Click on the drop-down **Actions** menu and choose **Assign a Customer Group**. A new menu will be listed where we can pick one of the newly created groups. Select one of the options, and click **OK** in the pop-up window:

Name	Group
Veronica Costello	General
Mary NL Bogman	General
General	
Wholesale	
Retailer	
Platinum	Inc. All rights
Gold	
Silver	
Special members	

6. To confirm the changes are OK, click the edit link on the right. In the **Customer View** tab we see the **Personal Information** and the listed **Customer Group**:

Customer Information		Personal Information	
Customer View Account Information Addresses		Last Logged In: Never (Offline) Confirmed email: Confirmed Account Created: Jan 22, 2016, 7:55:28 PM Account Created in: Default Store View Customer Group: Platinum	Default Billing Address Veronica Costello 6146 Honey Bluff Parkway Calder, Michigan, 49628-7978 United States T: (555) 229-3326

7. Congratulations, you have just finished configuring Customer Groups in Magento 2.

How it works...

Let's recap and find out what we did throughout the preceding recipe. In Steps 1 through 6, we configured different Customer Groups and connected them to the appropriate customer.

There's more...

Creating customers groups basically does nothing. It is the relation to marketing or segmentation that makes the difference. In Magento 2, we got **Catalog** and **Cart Price Rules**. When combining them with the **Customers Group** we are able to create member discounts.

Go to **Marketing | Promotions | Catalog Price Rules**, or **Cart Price Rules** and click **Add New Rule**. In the **General Information** section we see the **Customer Groups** list.

Now go ahead and create a new rule and use one of the groups. Here is an example. Create a new rule called Platinum 25%, and select the **Customer Groups Platinum**. Now go to the **Actions** tab menu and choose the **Apply rule: Percent of product price discount**. In the Discount Amount commit 25. This is the amount of percent the Platinum member gets discounted from the total of his shopping cart.

Make sure when testing that the customer login is in this group.

Configuring inventories

When selling physical products, it makes sense that every product in the warehouse is related to inventory or stock management. Configuring and managing stock is a day-to-day job. A basic inventory setup in Magento 2 is straightforward and comparable to Magento 1. With growth, a **product inventory management (PIM)** system will be needed.

Getting ready

To step through this recipe, you will use a Droplet created in *Chapter 2, Magento 2 System Tools* at DigitalOcean <https://www.digitalocean.com/>. We will be using an NGINX, PHP-FPM, Composer-based setup including sample data. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to configure an inventory for a Magento 2 setup. The following steps will guide you through this.

1. First, log in to the backend of Magento and go to **Stores | Configuration | Catalog | Inventory**. You will see two menu options: **Stock Options** and **Product Stock Options**.

In **Stock Options** you can configure the following:

Stock Options		
Set Items' Status to be In Stock When Order is Cancelled	<input type="text" value="Yes"/>	[GLOBAL]
Decrease Stock When Order is Placed	<input type="text" value="Yes"/>	[GLOBAL]
Display Out of Stock Products	<input type="text" value="No"/>	[GLOBAL]
Products will still be shown by direct product URLs.		
Only X left Threshold	<input type="text" value="0"/>	[WEBSITE]
Display products availability in stock on Storefront.	<input type="text" value="Yes"/>	[STORE VIEW]

All options are self-explanatory. One of the more interesting options is **Display Out of Stock Products**. This can be used to show the product even when it is currently not available. Some websites have a limited stock amount; by using this, the product is not always removed from the web, which is bad for **search engine optimization (SEO)**.

2. In **Product Stock Options** you can configure the following. All options are self-explanatory. Interesting options here are **Out-of-Stock Threshold**, **Minimum Qty Allowed in Shopping Cart** and **Automatically Return Credit Memo Item to Stock**.

The **Out-of-Stock Threshold** is set as Global, which means that it is related to all the products. **Minimum Qty Allowed in Shopping Cart** is related to the **Customer Groups** we created in the *Managing customer groups* recipe. Configuring the correct **Minimum Qty** amount helps to set the default before the discount rule applies. **Automatically Return Credit Memo Item to Stock** is helpful in managing credit orders. After creating a credit order, all items are updated in the current stock:

Product Stock Options

Please note that these settings apply to individual items in the cart, not to the entire cart.

Manage Stock	<input type="text" value="Yes"/>	[GLOBAL]									
Changing can take some time due to processing whole catalog.											
Backorders	<input type="text" value="No Backorders"/>	[GLOBAL]									
Changing can take some time due to processing whole catalog.											
Maximum Qty Allowed in Shopping Cart	<input type="text" value="10000"/>	[GLOBAL]									
Out-of-Stock Threshold	<input type="text" value="0"/>	[GLOBAL]									
Minimum Qty Allowed in Shopping Cart	<table border="1"> <thead> <tr> <th>Customer Group</th> <th>Minimum Qty</th> <th>Action</th> </tr> </thead> <tbody> <tr> <td><input type="text" value="ALL"/></td> <td><input type="text" value="1"/></td> <td></td> </tr> <tr> <td colspan="3"><input type="button" value="Add"/></td> </tr> </tbody> </table>	Customer Group	Minimum Qty	Action	<input type="text" value="ALL"/>	<input type="text" value="1"/>		<input type="button" value="Add"/>			[GLOBAL]
Customer Group	Minimum Qty	Action									
<input type="text" value="ALL"/>	<input type="text" value="1"/>										
<input type="button" value="Add"/>											
Notify for Quantity Below	<input type="text" value="1"/>	[GLOBAL]									
Enable Qty Increments	<input type="text" value="No"/>	[GLOBAL]									
Automatically Return Credit Memo Item to Stock	<input type="text" value="No"/>	[GLOBAL]									

- After configuring all elements, click **Save Config** and update your cache.
- Now go to **Products | Catalog** and click edit on the **Joust Duffle Bag** product. On the product page we have two options to change the current stock. The first option is **Quantity** in the **Product Detail** tab in **Basic Settings**. This option is straightforward. The second, more detailed option is in the **Advanced Settings** tab called **Advanced Inventory**. These options here only relate to this product. Depending on the configuration, you are able to override the **Config Setting** of the system.

In **Advanced Inventory** you can configure the following. All options are self-explanatory. Interesting options are **Qty Uses Decimals** and **Allow Multiple Boxes for Shipping**.

The **Qty Uses Decimals** setting determines whether decimals will be used in the quantity field for the product (for example, 3.5 yards) or not.

The **Allow Multiple Boxes for Shipping** setting determines whether multiple boxes will be used for the product during shipping (for example, custom build computer incl. monitor) or not:

Advanced Inventory		
Manage Stock	<input type="button" value="Yes"/> <input type="button" value="No"/>	[GLOBAL]
	<input checked="" type="checkbox"/> Use Config Settings	
Qty	<input type="text" value="99"/>	[GLOBAL]
Out-of-Stock Threshold	<input type="text" value="0"/>	[GLOBAL]
	<input checked="" type="checkbox"/> Use Config Settings	
Minimum Qty Allowed in Shopping Cart	<input type="text" value="0"/>	[GLOBAL]
	<input checked="" type="checkbox"/> Use Config Settings	
Maximum Qty Allowed in Shopping Cart	<input type="text" value="10000"/>	[GLOBAL]
	<input checked="" type="checkbox"/> Use Config Settings	
Qty Uses Decimals	<input type="button" value="No"/> <input type="button" value="Yes"/>	[GLOBAL]
Allow Multiple Boxes for Shipping.	<input type="button" value="No"/> <input type="button" value="Yes"/>	[GLOBAL]
Backorders	<input type="text" value="No Backorders"/> <input type="button" value="▼"/>	[GLOBAL]
	<input checked="" type="checkbox"/> Use Config Settings	
Notify for Quantity Below	<input type="text" value="1"/>	[GLOBAL]
	<input checked="" type="checkbox"/> Use Config Settings	
Enable Qty Increments	<input type="button" value="No"/> <input type="button" value="Yes"/>	[GLOBAL]
	<input checked="" type="checkbox"/> Use Config Settings	
Stock Availability	<input type="text" value="In Stock"/> <input type="button" value="▼"/>	[GLOBAL]

- Congratulations, you just finished configuring inventory management in Magento 2.

How it works...

Let's recap and find out what we did throughout the preceding recipe. In Steps 1 through 4, we configured an inventory on a global system level and an advanced product inventory level. Depending on your system setup, you can change them in line with the website or store view.

There's more...

Depending on your setting, it is best to check your low stock on a daily basis. Go to **Reports | Products | Low Stock**. All products with a low stock setting will be listed here.

Configuring currency rates

Every website selling products uses some kind of currency. Without a valid currency things would start to get messy.

Configuring these currencies is pretty straightforward. The most interesting part is the currency rate exchange. Every day this rate needs to be checked and updated. Decreases in value are important on the product you sell. So choosing the correct default currency needs to be considered appropriately.

Getting ready

To step through this recipe, you will use a Droplet created in *Chapter 2, Magento 2 System Tools* at DigitalOcean <https://www.digitalocean.com/>. We will be using an NGINX, PHP-FPM, Composer-based setup including sample data. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to configure three currencies—US Dollar, Euro, and Pound Sterling—with Magento 2. The following steps will guide you through this.

1. First log in to the backend of Magento. Go to **Stores | Configuration | General | Currency Setup** and select the **Base Currency**. Now select **Default Display Currency**. When running a multi website or store view, you may need to switch to the appropriate website or store view to change the value.
2. Next select in the multi select window a currency from **British Pound Sterling, Euro, and US Dollar**. Hold down the Ctrl key as you select from the list and click **Save Config**. Make sure you update your cache.
3. Now go to **Stores | Currency Symbols** to check if they are correct. Make adjustments when needed.

- Now go to **Stores | Currency Rates** and click on the **Import** button. Depending on the default currency, the exchange rate between GBP, EUR, or USD is calculated. Check if the rates are correct and click **Save Currency Rates**. Make sure you update your cache:

EUR	GBP	USD
1.0000	0.7573	1.4150

- Next open up a new browser and go to your home page. In the top right corner there is a drop-down with all the listed currencies. Select one of them to test if the currencies are correct.
- Congratulations, you just finished configuring currency rates in Magento 2.

How it works...

Let's recap and find out what we did throughout the preceding recipe. In Steps 1 through 5, we configured additional currency for our website. Depending on the currencies you are selling, this can be updated in the backend. On the frontend, customers can easily switch and buy in their preferred currency.

There's more...

Would you like to update the currency rates on a daily, weekly, or monthly basis? Go to **Stores | Configuration | General | Currency Setup | Scheduled Import Setting** and **Enable** this depending on your needs. This service will update the rates on the selected time basis.

Managing advanced pricing

Selling products can be hard. Lots of websites are selling the same product. But how can we attract new customers and persuade them to buy our goods?

Many websites have the same default retail price shown on their site. When using advanced pricing we can show special prices within a particular date range, or display tier prices for our beloved platinum members.

Another option could be using a **minimum advertised price (MAP)**. This feature is useful if your product manufacturer has established a **Manufacturer's Suggested Retail Price (MSRP)**, and you want to sell the product at a price lower than the MSRP. What this feature basically does is hide the product price display in your catalog and only shows this during product purchase.

Getting ready

To step through this recipe, you will use a Droplet created in *Chapter 2, Magento 2 System Tools* at DigitalOcean <https://www.digitalocean.com/>. We will be using an NGINX, PHP-FPM, Composer-based setup including sample data. No other prerequisites are required.

How to do it...

For the purpose of this recipe, let's assume that we need to create advanced pricing for some of our products in Magento 2. The following steps will guide you through this.

1. First log in to the backend of Magento, go to **Products | Catalog** and open **Joust Duffle Bag**.
2. Now click on the **Advanced Pricing** menu on the left and set a new **Special Price** from the date range you prefer.
3. Now update your **Tier Price** using the Platinum, Gold, and Silver **Customer Groups**.
4. Finally, set the new price for the **Manufacturer's Suggested Retail Price** and choose the correct **Display Actual Price** option.

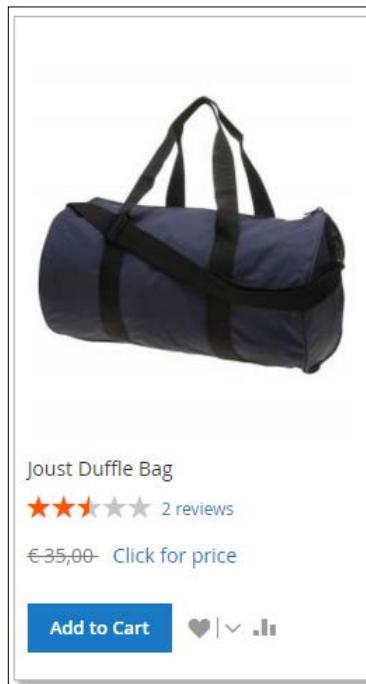
You can choose from **Use Config**, **On Gesture**, **In Cart**, and **Before Order Confirmation**.

- ❑ **Use Config:** The default configuration located in **Stores | Configuration | Sales | Sales | Minimum Advertised Price**.
- ❑ **On Gesture:** Product prices are shown in the catalog pages but only when the user clicks on the link, which shows the price in a pop-up.
- ❑ **In Cart:** Customers can see product prices when products have been added to the cart.

- **Before Order Confirmation:** Product prices are shown on the checkout page:

Advanced Pricing					Add All
Special Price	€ 30.00			[GLOBAL]	
Special Price From Date	1/24/2016		<input type="button" value="Calendar"/>	[WEBSITE]	
Special Price To Date			<input type="button" value="Calendar"/>	[WEBSITE]	
Cost	€			[GLOBAL]	
Tier Price	Web Site	Customer Group	Quantity	Item Price	Action
	All ▾	Gold	10 and above	22.50	<input type="button" value="Delete"/>
	All ▾	Plat	10 and above	20.00	<input type="button" value="Delete"/>
	All ▾	Silver	10 and above	25.00	<input type="button" value="Delete"/>
	<input type="button" value="Add Price"/>				
Manufacturer's Suggested Retail Price	€ 35.00			[GLOBAL]	
Display Actual Price	On Gesture			[WEBSITE]	
	Use config				
	On Gesture				
	In Cart				
	Before Order Confirmation				

5. Next open up a new browser and go to `http://yourdomain.com/joust-duffle-bag.html`. Depending on your configuration, it should look as follows:



Joust Duffle Bag

2 Reviews Add Your Review

€35,00 [Click for price](#) [What's this?](#)

IN STOCK
SKU#: 24-MB01

Our price is lower than the manufacturer's "minimum advertised price." As a result, we cannot show you the price in catalog or the product page. X

You have no obligation to purchase the product once you know the price. You can simply remove the item from your cart.

[Add to Cart](#)

6. Congratulations, you just finished managing advanced pricing in Magento 2.

How it works...

Let's recap and find out what we did throughout the preceding recipe. In Steps 1 through 5, we configured advanced pricing using a special price, a tier price, and the minimum advertised price in a product.

6

Creating a Magento 2 Theme

In this chapter, we will cover some basics on how to build your own theme based on the Magento 2 blank theme and add/change pages of built-in modules through layout XML through the following recipes:

- ▶ Creating a new theme
- ▶ Changing a layout XML of a Magento 2 module
- ▶ Adding CSS/JS to pages
- ▶ Using Grunt for CSS changes
- ▶ Adding static blocks to pages through layout XML
- ▶ Adding static blocks to pages through widgets
- ▶ Using a dynamic serving theme based on the client browser
- ▶ Creating theme-specific translations

Introduction

Theming is an important part of building your e-commerce website; making it easy for users to navigate and SEO-friendly will make sure that your conversion will be optimized.

In Magento 2, theming has changed in a big way; it's easier to optimize your theme with the more granular way of controlling what is outputted through the layout configurations.

In this chapter, we will see some basics on how to build your own theme extended from the Magento blank theme, which can also be used to extend any other theme you want to use as a starting point.

In order to understand the way theming is done, you should know how to work with Less, CSS, XML, and PHP.

Creating a new theme

Themes in Magento 2 are set up a bit differently than Magento 1. Some of these changes are as follows:

- ▶ Smaller layout files per layout handle
- ▶ Less (default) implementation with an internal Less preprocessor
- ▶ Extended layout methods to move and change blocks
- ▶ Magento UI library for default components, such as forms, buttons, and more
- ▶ Installable through Composer
- ▶ Fallback to module layout, templates, and other public files
- ▶ Static file generation to improve page load times

In this sample theme, the files are located in `app/design/frontend/<Vendor>/<Theme>`. When a theme is installed through Composer, it will be installed in the vendor directory.

Getting ready

In order to work with themes, you should have a basic knowledge of XML, HTML, CSS, and Less as these are used to build your theme.

How to do it...

The following are the steps to create a new theme:

1. First, we start by creating the theme definition file:

```
app/design/frontend/Genmato/default/theme.xml

<?xml version="1.0"?>
<theme xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:n
oNamespaceSchemaLocation="urn:magento:framework:Config/etc/theme.
xsd">
    <title>M2 Cookbook Sample Theme</title>
    <parent>Magento/blank</parent>
    <media>
        <preview_image>media/preview.png</preview_image>
    </media>
</theme>
```

2. Create preview.png for how your theme will look like. (This file needs to be present, but you can start with a blank file and replace this later when your theme is done.) Place this preview image under app/design/frontend/Genmato/default/media/.
3. In order to have it installable through Composer, we need to create a composer.json file:

```
app/design/frontend/Genmato/default/composer.json

{
    "name": "genmato/sample-theme",
    "description": "Genmato Sample Theme",
    "require": {
        "php": "~5.5.0|~5.6.0|~7.0.0",
        "magento/theme-frontend-blank": "100.0.*",
        "magento/framework": "100.0.*"
    },
    "type": "magento2-theme",
    "version": "1.0.0",
    "license": [
        "OSL-3.0",
        "AFL-3.0"
    ],
    "autoload": {
        "files": [
            "registration.php"
        ]
    }
}
```

4. In order to register the theme when loaded through Composer, it needs registration.php:

```
app/design/frontend/Genmato/default/registration.php

<?php
\Magento\Framework\Component\ComponentRegistrar::register(
\Magento\Framework\Component\ComponentRegistrar::THEME,
'frontend/Genmato/default',
__DIR__
);
```

5. Create a static file directory structure in your theme:

```
app/design/frontend/Genmato/default  
  web/  
    web/css/source/  
    web/fonts/  
    web/images/  
    web/js/
```

6. Define your logo file and size:

```
app/design/frontend/Genmato/default/Magento_Theme/layout/  
default.xml
```

```
<?xml version="1.0"?>  
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
      xsi:noNamespaceSchemaLocation="urn:magento:framework:  
      View/Layout/etc/page_configuration.xsd">  
  <body>  
    <referenceBlock name="logo">  
      <arguments>  
        <argument name="logo_file" xsi:type="string">  
          images/genmato.svg</argument>  
        <argument name="logo_img_width" xsi:type="number">  
          372</argument>  
        <argument name="logo_img_height" xsi:type="number">  
          84</argument>  
      </arguments>  
    </referenceBlock>  
    <referenceBlock name="report.bugs" remove="true"/>  
  </body>  
</page>
```

7. Place your logo file in app/design/frontend/Genmato/default/web/images/; in this example, an SVG is used but you can also define a PNG or JPG file.
8. It is possible to configure your own image sizes for the different images; when generating the static content, all images will be created in the width/height configured in this file:

```
app/design/frontend/Genmato/default/etc/view.xml
```

```
<?xml version="1.0"?>  
<view xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
      xsi:noNamespaceSchemaLocation="urn:magento:framework:  
      Config/etc/view.xsd">  
  <media>  
    <images module="Magento_Catalog">
```

```
<image id="category_page_grid" type="small_image">
<width>240</width>
<height>300</height>
</image>
</images>
</media>
</view>
```

9. Create your theme style (Less) file. In this file, all overrides for styles used in the blank theme and Magento UI framework can be specified. This can be used to change default colors in your theme:

```
app/design/frontend/Genmato/default/web/css/source/_theme.less

@page_background-color: @color-gray20;
@primary_color: @color-gray80;

@genmato_green: #009A4E;
@genmato_blue: #0089CF;

@link_color: @genmato_green;
@link_hover_color: darken(@link_color, 10%);

@button-primary_background: @genmato_green;
@button-primary_border: darken(@genmato_green, 40%);
@button-primary_color: @color-black;

@button-primary_hover_background: darken(@genmato_green, 10%);
@button-primary_hover_border: darken(@genmato_green, 50%);
@button-primary_hover_color: @color-black;

@navigation_background: @genmato_blue;
```

10. After all the files are uploaded to the Magento 2 installation, refresh the cache:

```
bin/magento cache:clean
```

11. Next, generate the static content:

```
bin/magento setup:static-content:deploy
```

You can rerun this command every time you make changes to your theme and need to regenerate the CSS from the Less files. Make sure that you remove your theme-generated files from the following locations:



pub/static/frontend/<Theme Vendor>

var/view_preprocessed/css/frontend/<Theme Vendor>

Otherwise, the changes in the Less files in your theme will not be used as the preprocessor checks if there is already a generated CSS file available. Check the *Using Grunt for CSS changes* recipe of this chapter on how to use live reloading of CSS changes without the need of recompiling.

12. The theme should now be available to select for your store. In order to change the theme, go to the following:

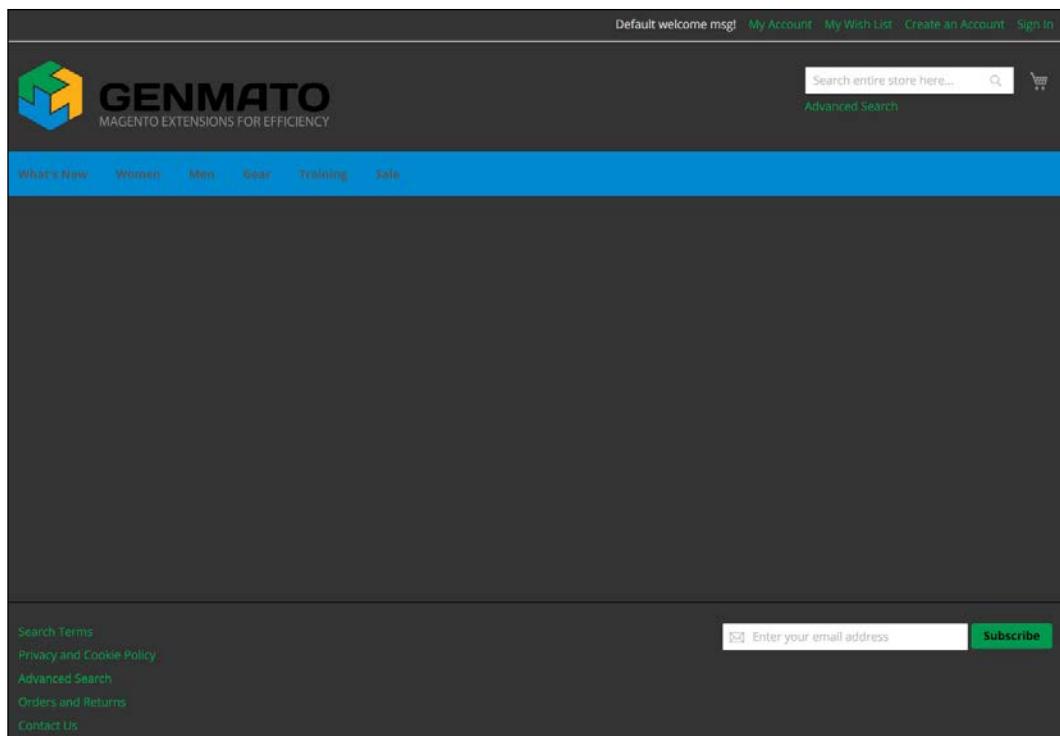
Stores | [General] Design | Design Theme

This can be seen in the following screenshot:

The screenshot shows the 'Design Theme' configuration page in the Magento Admin Panel. The left sidebar has tabs for General, Web, Design (which is selected and highlighted in orange), Currency Setup, and Store Email Addresses. The main area has sections for 'Design Theme' (with a dropdown menu showing 'M2 Cookbook Sample Theme' as the current selection), 'User-Agent Exceptions' (containing a search bar with the value '/^mozilla/i'), and 'Action' (with a 'Save' button). A tooltip for the User-Agent Exceptions section states: 'Search strings are either normal strings or regular exceptions (PCRE). They are matched in the same order as entered. Examples: Firefox /~mozilla/i'.

Choose the newly created theme from the drop-down list and save the configuration change.

13. Navigate to your store frontend and check whether the new theme is visible. The theme used in this example looks as follows:



How it works...

The theme configuration in Magento 2 is more powerful than in Magento 1, allowing you to have better control on changing elements that are available from installed modules. The way in which the fallback is configured makes it easier to create different variants of a theme by defining the parent theme only. During compilation of the theme, all the files are gathered from the right parent and merged into your theme. This improves page rendering as there is no more layout merging done. All static elements such as CSS and images are pregenerated. Building themes for distribution and changing them afterward is now also much easier and can be done without modifying the bought theme.

During the `bin/magento setup:static-content:deploy` command, the system collects all the Less files from the following:

- ▶ Current active theme
- ▶ The defined parent theme(s); this is done recursively for all parents until there is no parent defined
- ▶ The module files

Next, it will use the built-in `PHPLess` module to merge all these files into the configured CSS files. In this example, it generated a `styles-m.css` and `styles-l.css` as configured in the blank theme (`Magento/blank/Magento_Theme/default_head_blocks.xml`):

```
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="urn:magento:framework:
      View/Layout/etc/page_configuration.xsd">
    <head>
        <css src="css/styles-m.css" />
        <css src="css/styles-l.css" media="screen and (min-width:
        768px)"/>
        <css src="css/print.css" media="print" />
    </head>
</page>
```

The styles files are built from `styles-m.less` and `styles-l.less` (found in `Magento/blank/web/css/`) and define all the Less files that should be included. The two files are the definition files for the mobile and desktop versions of the layout. These external Less files are included through the default `@import` command used in Less. They also contain a special `@magento_import` command (which has to be commented out in order to avoid breaking the Less preprocessor). During compilation of the theme, Magento replaces these imports with a regular `@import` command but with a resolved path to the corresponding file location based on the fallback file found. During compilation, all files are stored at the `pub/static/frontend/<Vendor>/<theme>` location and served as static files to improve load times.

Adding theme variants

Creating a variant of a theme, for example, for seasonal promotions, is easy to add. Here, it is only necessary to create a new theme that has a parent to the default/normal theme; all separate themes need to be located in their own directories. The theme only needs the files that are different from the parent theme; this can be just CSS changes or static files used as backgrounds in the theme.

Layout files

In Magento 2, the layout files are split per layout handle; this makes it easier to modify a specific page only. A layout handle is a unique identifier for the layout definitions that are used to build the page. There are three different types of layout handles:

- ▶ page type layout handles: These identify the page based on the full action names of the controller (`customer_account_create`)
- ▶ page layout handles: These are added identifiers based on a product type shown (`catalog_product_view_type_downloadable`)
- ▶ custom handles: These are added custom identifiers not referencing to any page

Every file is located in the corresponding Module directory, so for the `customer_account_create` page handle, the layout file would be located in `[Theme Directory]/Magento_Customer/layout/customer_account_create.xml`. In the design and building of the pages, the layout is configured based on containers and defines the basic structure of a page (such as the header, footer, and columns—left, main, and right). In the containers, the content is added using blocks; every block has a template and block class assigned that is used to render the HTML for that block. It is possible to have multiple blocks assigned to a single container, allowing you to assign the order in which they must be shown. During generation, all layout files are merged together for each layout handle, allowing you to modify the output in the theme without the need of including the complete original files from the module.

Template files

Template files are `phtml` files containing the HTML and PHP code to build the specified content. In order to change or add data with a template in your theme, you will need to copy the original template file to your theme, for example, to change the account registration form, the `<customer-module-dir>/view/frontend/templates/form/register.phtml` file needs to be copied to `<theme-dir>/Magento_Customer/templates/form/register.phtml` and can be edited there.

Magento UI library

In Magento 2, there is a UI library available that includes basic interface CSS-class elements that can be used in the templates. The following components are available:

- ▶ actions-toolbar
- ▶ breadcrumbs
- ▶ buttons
- ▶ drop-downs
- ▶ forms
- ▶ icons
- ▶ layout
- ▶ loaders
- ▶ messages
- ▶ pagination
- ▶ popups
- ▶ ratings
- ▶ sections
- ▶ tabs and accordions
- ▶ tables

- ▶ tooltips
- ▶ typography
- ▶ list of theme variables

The CSS definition of all these elements can be altered to find out what variables to alter; check out `<magento-root>/lib/web/css/source/lib/variables`. In order to modify the way an element is rendered, you can look up the required variable and add your own definition in the `<theme-directory>/web/css/source/_theme.less` theme file.

Changing a layout XML of a Magento 2 module

In order to customize the layout to your own requirements, you can just add it to your theme and make the required changes. In order to change a layout from a Magento 2 module, you will need to locate the module and layout handle that you want to alter.

Getting ready

In this recipe, we will change the order of some elements on the product view page based on the theme created in the previous recipe.

How to do it...

The following steps will show you how to change elements defined in a layout file to match your desired design:

1. Create the layout handle file for the `Magento_Catalog` module:

```
app/design/frontend/Genmato/default/Magento_Catalog/layout/
catalog_product_view.xml
```

```
<?xml version="1.0"?>
<page layout="1column"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="urn:magento:framework:
      View/Layout/etc/page_configuration.xsd">
<body>
    <move element="product.info.stock.sku"
          destination="product.info.price"
          after="product.price.final"/>
    <move element="product.info.review"
          destination="product.info.main"
          before="product.info.price"/>
```

```
<remove name="report.bugs"/>  
</body>  
</page>
```

2. Upload the file to your Magento 2 installation and refresh the cache:

```
bin/magento cache:clean
```

3. Next, generate the static content:

```
bin/magento setup:static-content:deploy
```

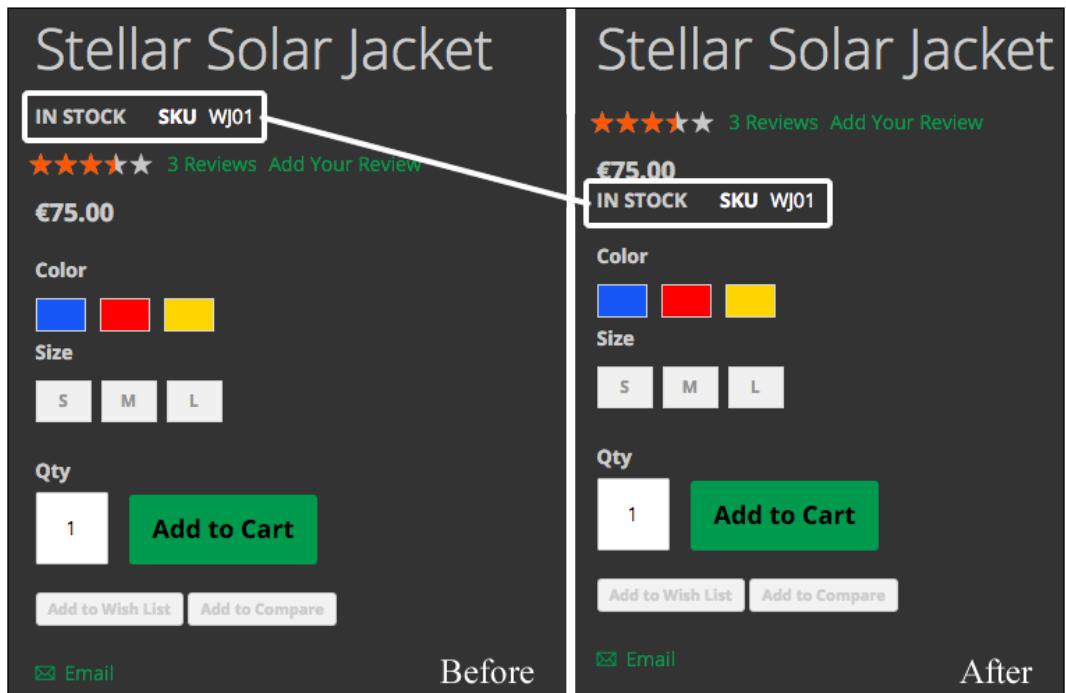
Make sure that you remove your theme-generated files from the following locations:

pub/static/frontend/<Theme Vendor>

var/view_preprocessed/css/frontend/<Theme Vendor>

Otherwise, the changes in the Less files in your theme will not be used as the preprocessor checks if there is already a generated CSS file available.

4. Navigate to a product page to see the changes:



How it works...

In the layout XML files, there are a few commands available to change the way in which a page is rendered. Here is a short description of every available command.

<container>

A container defines a structural layout block that does not produce its own content and can hold other blocks and/or containers. The output of the content generated by the children can be rendered in any valid HTML 5 tag with the option to specify an ID or class used for the element. Here is an example of the `product.info.stock.sku` container and the blocks that are added:

```
<container name="product.info.stock.sku" label="Product auxiliary
info" htmlTag="div" htmlClass="product-info-stock-sku">
<container name="product.info.type" before="-"/>
<block class="Magento\Catalog\Block\Product\View\Description"
name="product.info.sku" template=
"product/view/attribute.phtml" after="product.info.type">
<arguments>
<argument name="at_call" xsi:type="string">getSku</argument>
<argument name="at_code" xsi:type="string">sku</argument>
<argument name="css_class" xsi:type="string">sku</argument>
<argument name="at_label"
xsi:type="string">default</argument>
<argument name="add_attribute"
xsi:type="string">itemprop="sku"</argument>
</arguments>
</block>
</container>
```

A block generates content from the specified class and assigned template file. In the arguments of the block, it's possible to specify the load order using the `before` and `after` tags. Depending on the class specified, it's possible to pass information using the `<argument>` tag:

```
<block class="Magento\Catalog\Block\Product\View\Description"
name="product.info.overview" template=
"product/view/attribute.phtml" group="detailed_info"
after="product.info.extrahint">
<arguments>
<argument name="css_class"
xsi:type="string">overview</argument>
</arguments>
</block>
```

Arguments passed to the class can be the class methods or magic setters/getters and can be accessed in the template. The preceding `css_class` argument assigned can be requested in the template through `$this->getCssClass()`.

referenceContainer/referenceBlock

In order to add a block or container to an element specified in another layout file, you will need to reference this element. This way it's possible to add blocks to both the main content area and sidebar in the same layout file:

```
<referenceContainer name="product.info.type">
    <block class="Magento\Catalog\Block\Product\View\Type\Simple"
        name="product.info.simple" as="product_type_data"
        template="product/view/type/default.phtml"/>
    <container name="product.info.simple.extra"
        after="product.info.simple" as="product_type_data_extra"
        label="Product Extra Info"/>
</referenceContainer>
```

move

The move command allows you to change the location where the element is shown to another element. In this recipe, the SKU information was placed after the `product.price.final` block in the `product.info.price` container, where the default location would be the first block shown.

remove

The remove command will remove the block referenced with the name parameter; this will cause it not to be rendered on the page.

update

With the update instruction, it is possible to include a layout handle. This allows you to include a set of instructions defined once to multiple layouts. An example of this is the customer account; here, all account menu options for logged in users are defined in the `customer_account.xml` layout file:

```
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    layout="2columns-left" xsi:noNamespaceSchemaLocation=
    "urn:magento:framework:View/Layout/etc/page_configuration.xsd"
    label="Customer My Account (All Pages)"
    design_abstraction="custom">
    <body>
        <attribute name="class" value="account"/>
        <referenceContainer name="sidebar.main">
```

```
<block class="Magento\Framework\View\Element\Html\Links"
    name="customer_account_navigation" before="-"
    template="Magento_Customer::account/navigation.phtml">
<block class=
    "Magento\Framework\View\Element\Html\Link\Current"
    name="customer-account-navigation-account-link">
<arguments>
    <argument name="label" xsi:type="string"
        translate="true">Account Dashboard</argument>
    <argument name="path"
        xsi:type="string">customer/account</argument>
</arguments>
</block>
<block class=
    "Magento\Framework\View\Element\Html\Link\Current"
    name="customer-account-navigation-account-edit-link">
<arguments>
    <argument name="label" xsi:type="string"
        translate="true">Account Information</argument>
    <argument name="path"
        xsi:type="string">customer/account/edit</argument>
</arguments>
</block>
<block class=
    "Magento\Framework\View\Element\Html\Link\Current"
    name="customer-account-navigation-address-link">
<arguments>
    <argument name="label" xsi:type="string"
        translate="true">Address Book</argument>
    <argument name="path"
        xsi:type="string">customer/address</argument>
</arguments>
</block>
</block>
</referenceContainer>
</body>
</page>
```

This file is then included in the `customer_account_index.xml` layout (and other pages using this menu):

```
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    layout="2columns-left" xsi:noNamespaceSchemaLocation=
    "urn:magento:framework:View/Layout/etc/page_configuration.xsd">
<update handle="customer_account"/>
<body>
    <referenceBlock name="page.main.title">
```

```

<action method=" setPageTitle">
    <argument translate="true" name="title"
        xsi:type="string">My Dashboard</argument>
</action>
</referenceBlock>
<referenceContainer name="content">
    <block class="Magento\Framework\View\Element\Template"
        name="customer_account_dashboard_top" as="top"/>
    <block class="Magento\Customer\Block\Account\Dashboard\Info"
        name="customer_account_dashboard_info" as="info" template=
        "account/dashboard/info.phtml" cacheable="false"/>
    <block class="Magento\Customer\Block\Account\Dashboard\
        Address" name="customer_account_dashboard_address"
        as="address" template="account/dashboard/address.phtml"
        cacheable="false"/>
</referenceContainer>
</body>
</page>
```

Overriding template files

Templates are loaded in the following order:

1. Active theme, where it looks for the file in <theme>/<Module_Namespace>_<Module_name>/template/<requested template>.
2. Parent theme(s) (until no parent is found).
3. Module directory.

In order to override a template in your theme, it is possible to create your own version of the file. For example, to replace the left navigation file from the catalog module, copy the file from the original location <Magento_Catalog_path>/view/frontend/templates/navigation/left.phtml to your theme location <theme>/Magento_Catalog/templates/navigation/left.phtml. Here, you can apply your own changes necessary to suit your theme.

Another option is to change the template file assigned through a layout change; this can be useful if you want to change the template only for a specific product, category, or other page handle available. For this, you need to create a new layout file based on the file handle where you reference the block where you want to update the template. Using the <action> method, setTemplate, you can now specify the new template you want to use:

```

<referenceBlock name=" [blockname] ">
<arguments>
    <argument name="template" xsi:type="string">[Module
        name] :: [path to template]</argument>
</arguments>
</referenceBlock>
```

Adding CSS/JS to pages

Adding your own CSS or JS to every or specific page can also be done through a layout XML file. The source of a file that you want to include can come from the following:

- ▶ An external source hosted remotely
- ▶ Module-specific, which is mainly used by JavaScript
- ▶ Theme-specific, which is mainly used when adding CSS but can also be used to add custom JavaScript

Getting ready

In order to include CSS or JavaScript to a page, you first need to know where you want to include your files and where the file is located that you want to include. In this recipe, we will see how to include both an external CSS and JavaScript file and a theme JavaScript and CSS file.

How to do it...

This recipe is based on the theme created in the *Creating a new theme* recipe of this chapter.

1. To add a new source (CSS or JS) to all pages, the configuration is done through the `default_head_blocks.xml` file. It is also possible to add it to a single page only through the layout file for that layout handle:

```
app/design/frontend/Genmato/default/Magento_Theme/layout/
default_head_blocks.xml

<?xml version="1.0"?>
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="urn:magento:framework:
      View/Layout/etc/page_configuration.xsd">
    <head>
        <link src=
              'https://fonts.googleapis.com/css?family=Open+Sans'
              type='text/css' src_type="url"/>
        <link src='https://ajax.googleapis.com/ajax/libs/
                  jqueryui/1.11.4/jquery-ui.min.js' src_type="url"/>
        <link src="js/sample.js"/>
        <css src="css/sample.css" />
    </head>
</page>
```

2. Add the custom JavaScript code (just the basic file here):

```
app/design/frontend/Genmato/default/web/js/sample.js
```

```
require([
    'jquery',
], function ($) {
    jQuery(document).ready(function () {
        // Your jQuery code here
    });
});
```

3. Add your custom stylesheet Less file:

```
app/design/frontend/Genmato/default/web/css/sample.less
```

you can add your own Less code here!

4. Upload the file to your Magento 2 installation and refresh the cache:

```
bin/magento cache:clean
```

5. Next, generate the static content:

```
bin/magento setup:static-content:deploy
```

Make sure that you remove your theme-generated files from the following locations:

pub/static/frontend/<Theme Vendor>

var/view_preprocessed/css/frontend/<Theme Vendor>

Otherwise, the changes in the Less files in your theme will not be used as the preprocessor checks if there is already a generated CSS file available.

Refresh the page and make sure that the new CSS and JS files are added to the header of the source of your page.

How it works...

During the generation process, the XML tags that are added to the head component are converted depending on what is supplied in the configuration.

External files

When adding files from an external source, you need to specify `src_type=url`. Otherwise, the supplied source will be converted into a file on the local filesystem and result in an error loading the resource.

JavaScript files

Magento 2 uses RequireJS to manage all JavaScript and its dependencies; this means that it's not possible to just include an extra JS file and use functions that are available with jQuery. In order to use calls to jQuery functions, you need to add this dependency in your JS file; this will make sure that the jQuery library is loaded before your code is executed.

CSS files

When including a CSS file, the CSS is generated from a Less file, which is done automatically by the Magento preprocessor when the specified CSS isn't found and there is a `.less` file found with the same name.

Removing a file in the header

It is also possible to remove a file from the header; this can be useful if you want to remove an included resource from a parent theme. To remove a CSS, JavaScript, or font from the header, you can use the `<remove>` tag where you reference the path that you want to remove. For example, to remove the `styles-1.css` file from the page, you can use the following:

```
<theme>/Magento_Theme/layout/default_head_blocks.xml

<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="urn:magento:framework:
      View/Layout/etc/page_configuration.xsd">
    <head>
        <remove src="css/styles-1.css"/>
    </head>
</page>
```

Using Grunt for CSS changes

Making changes during the development of your theme can take some time while you are waiting on removing the compiled files, recompiling, and reloading. To optimize the workflow and speed up the reloading of the changes, it is possible to use the built-in client-side Less compiler based on a JavaScript compilation. Using this client compiler might not work fast enough for you and require manual reloads. Another option is to use the Grunt tool to watch changes made in the `.less` files. In this recipe, we will see how to make use of this tool and update CSS files without the manual removing of files and refreshing the browser.

Getting ready

Using Grunt should only be needed on your local development setup; for production systems, you should use the built-in Magento Less compiler. Grunt is built on Node.js; in order to use it, you should first install Node.js in your system.

How to do it...

This recipe will explain how to install, configure, and use Grunt to monitor file changes and recompile the CSS scripts. This recipe is based on the theme configured in the *Creating a new theme* recipe of this chapter. When you use it on your own theme, change the names/files according to your theme needs:

1. First, we need to install the Grunt command-line tool. As Grunt is a Node.js package, this installation is done through the **Node.js package manager (npm)**:

```
npm install -g grunt-cli
```

2. To install Grunt in your Magento project directory, navigate to the Magento 2 root directory of your installation and run the following:

```
npm install grunt --save-dev
```

3. After installing Grunt, the Node.js dependencies in your Magento project should be updated. To do so, run the following command from your Magento directory:

```
npm install  
npm update
```

4. To allow Grunt to recompile your own theme, it should be declared in the `dev/tools/grunt/configs/themes.js` file. This file configures the paths used to monitor file changes and the files that should be created. Add the following code to the file:

```
genmato: {  
    area: 'frontend',  
    name: 'Genmato/default',  
    locale: 'en_US',  
    files: [  
        'css/styles-m',  
        'css/styles-l',  
        'css/sample'  
    ],  
    dsl: 'less'  
},
```

5. Run the initial setup of your theme files; this will create symlinks from the `pub/static/frontend/Genmato/default` theme directory to the source theme files (located in `app/design/frontend/Genmato/default`):

```
grunt exec:genmato
```

The following output displays the processing done by Grunt:

6. Now that the symlinks are created, the Less files should be compiled; this will create the theme CSS files as they were configured in the `themes.js` file in step 4:

grunt less:genmato

This can be seen in the following screenshot:

```
mfp:html vladimir$ grunt less:genmoto
Running "less:genmoto" (less) task
File pub/static/frontend/Gemalto/default/en_US/css/styles-m.css created: 278.12 kB + 480.24 kB
File pub/static/frontend/Gemalto/default/en_US/css/styles-l.css created: 72.4 kB + 125.22 kB
File pub/static/frontend/Gemalto/default/en_US/css/sample.css created: 0 B + 0 B

Done, without errors.

Execution Time (2016-02-16 11:29:37 UTC)
loading tasks          186ms [██████████ 3%]
loading grunt-contrib-less 116ms [████ 2%]
less:genmoto           6.3s
Total 6.6s
```

- In order to display the CSS changes made automatically, install the **LiveReload** plugin for your browser from <http://livereload.com/extensions/>. When the plugin is installed, click on the icon to activate the live reloading of the CSS files. This should be done only on your local website URL.
 - When making changes to the Less files in your theme, the Grunt watcher should be activated:
`grunt watch`

This will detect the changes once you save a theme Less file and trigger the compilation of your Less code to a CSS file.

- Now you can make all your necessary changes, for example, we could change the color of the navigation background:

```
app/design/frontend/Genmato/default/web/css/source/_theme.less
@genmato__orange: #FCBF45;
@navigation__background: @genmato__orange;
```

After saving the file, the compilation of the CSS is triggered and a new CSS file is stored for your theme:

```
mbp:html vladimir$ grunt watch
Running "watch" task
Waiting...
>> File "pub/static/frontend/Genmato/default/en_US/css/source/_theme.less" changed.
Running "less:genmato" (less) task
File pub/static/frontend/Genmato/default/en_US/css/styles-m.css created: 278.12 kB → 480.24 kB
File pub/static/frontend/Genmato/default/en_US/css/styles-l.css created: 72.4 kB → 125.19 kB
File pub/static/frontend/Genmato/default/en_US/css/sample.css created: 0 B → 0 B

Done, without errors.

Execution Time (2016-02-16 11:44:02 UTC)
loading tasks      298ms [██████████] 4%
loading grunt-contrib-less 132ms [██████] 2%
less:genmato       7.8s [██████████] 95%
Total 8.2s

Completed in 9.265s at Tue Feb 16 2016 12:44:10 GMT+0100 (CET) - Waiting...
>> File "pub/static/frontend/Genmato/default/en_US/css/styles-m.css" changed.
>> File "pub/static/frontend/Genmato/default/en_US/css/sample.css" changed.
>> File "pub/static/frontend/Genmato/default/en_US/css/styles-l.css" changed.
Completed in 8.000s at Tue Feb 16 2016 12:44:10 GMT+0100 (CET) - Waiting...
```

- The live reload plugin will detect the changed CSS files and reload and apply them in your browser without reloading the page, which should now display the new color for the navigation background:



The Grunt watch command looks for changes made to the Less files in the template directory. When a change is detected, it will trigger the recompilation of the CSS files of the theme. To see what files Grunt is watching, you can run the watch command with the `-v` option:

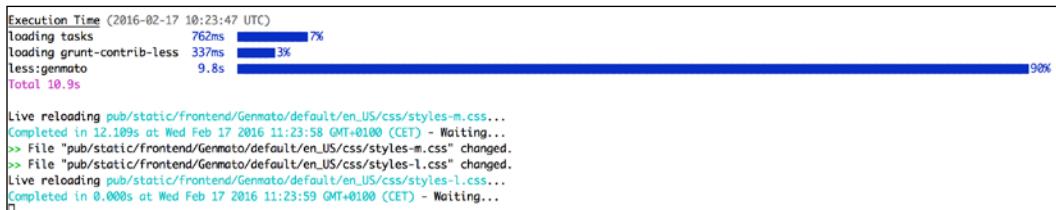
```
grunt watch -v
```

This will display all modules loaded and give you more debugging information for which file has been changed and the actions that it is running.

When using the Grunt method and deploy feature in Magento in the same setup, make sure that you run the command starting from step 5 again; otherwise, the file changes will not be detected.

LiveReload

When using the LiveReload plugin, it will create a web socket connection to the LiveReload service running on TCP port 35729. Through this socket, Grunt will notify when the compilation of the CSS file is completed and what file has changed so that the LiveReload plugin can reload that file and apply it in the browser:



Adding static blocks to pages through layout XML

With static blocks, it is possible to add content to pages that you can manage through the Magento backend. There are two ways to add a static block to your page. In this recipe, we will see how to add a static block through layout XML.

Getting ready

To add a static block to a page, you need to know on which page you want this block to be displayed and in what block or container.

How to do it...

This recipe shows you how to add a static block to the footer on all pages, based on the theme created in the *Creating a new theme* recipe of this chapter:

1. Create a new static block through the Magento backend. Go to the **Content** menu option and select **Blocks** under the **Elements** menu. Next, click on the **Add New Block** button to create a new block. Create the block with the content you want and click on **Save Block**:

General Information

Block Title * Sample Footer content

Identifier * footer-sample

Store View * All Store Views
Main Website
Main Website Store
Default Store View

Status * Enabled

Content * Show / Hide Editor

2. To add this block to the footer of your theme on all pages, it should be added to the `Magento_Theme default.xml` layout file:

```
app/design/frontend/Genmato/default/Magento_Theme/layout/
default.xml
```

```
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="urn:magento:framework:
      View/Layout/etc/page_configuration.xsd">
<body>
    <referenceBlock name="logo">
        <arguments>
            <argument name="logo_file"
                  xsi:type="string">images/genmato.svg</argument>
            <argument name="logo_img_width"
                  xsi:type="number">372</argument>
            <argument name="logo_img_height"
                  xsi:type="number">84</argument>
        </arguments>
    </referenceBlock>
```

```
<referenceBlock name="report.bugs" remove="true"/>
<referenceContainer name="footer">
    <block class="Magento\Cms\Block\Block"
        name="footer-sample">
        <arguments>
            <argument name="block_id"
                xsi:type="string">footer-sample</argument>
        </arguments>
    </block>
</referenceContainer>
</body>
</page>
```

3. After uploading the file to your Magento installation, refresh the cache:

```
bin/magento cache:clean
```

4. Open your browser and (re)load the website to check whether your content is now visible on the website.

How it works...

The code in this recipe adds the created block to the referenced footer container; here, we add a new block with the `Magento\Cms\Block\Block` class and pass an argument with `block_id` of the created block. The `block_id` specified must match the `Identifier` used when creating the block.

Adding a block to a single page

To add a block to a single page, for example, before the **Contact Us** form, create a new file, `app/design/frontend/Genmato/default/Magento_Contact/layout/contact_index_index.xml`, and add the following content:

```
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="urn:magento:framework:
    View/Layout/etc/page_configuration.xsd">
    <body>
        <referenceContainer name="content">
            <block class="Magento\Cms\Block\Block" name="contact-sample"
                before="contactForm">
                <arguments>
                    <argument name="block_id" xsi:type="string">[your block
                        identifier]</argument>
                </arguments>
            </block>
        </referenceContainer>
    </body>
</page>
```

In order to have the created block before the form, we reference the content area and add the block with specifying the `before` argument.

Adding static blocks to pages through widgets

Adding a static block to a page can also be done with the Magento widgets system; this allows you to add blocks to pages without the knowledge of how layout XML works.

How to do it...

In this recipe, we will see step by step how to add a static block to the home page:

1. Create a new static block through the Magento backend. Go to the **Content** menu option and select **Blocks** under the **Elements** menu. Next, click on the **Add New Block** button to create a new block. Create the block with the content you want and click on **Save Block**:

General Information

Block Title *

Identifier *

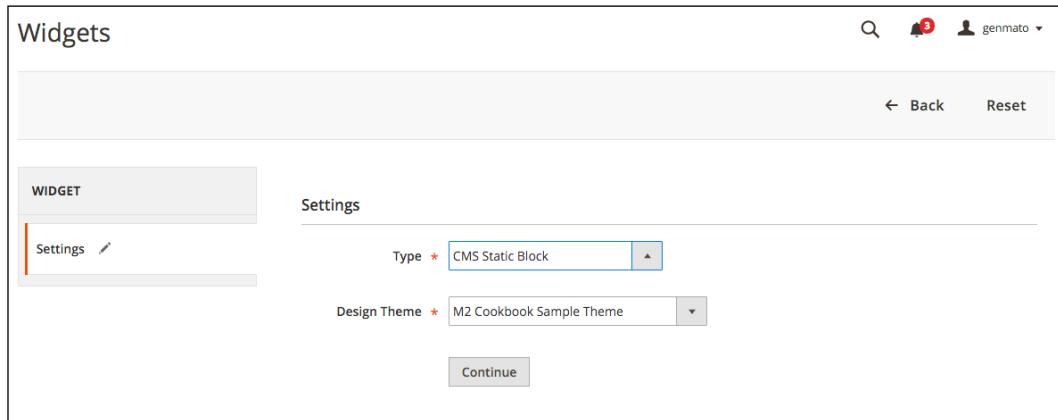
Store View * **Main Website** 

Status *

Content *


Sample homepage content

2. Create a new widget. Go to the **Content** menu option and select **Widgets** under the **Elements** menu. Next, click on the **Add Widget** button to create a new widget. Select the **CMS Static Block** option for the **Type** field and the theme that you want to apply this widget to under **Design Theme**. After this, click on the **Continue** button:



3. Specify the **Widget Title**, **Assign to Store Views**, and **Sort Order** properties:

The screenshot shows the 'Storefront Properties' page. It includes fields for 'Type' (set to 'CMS Static Block'), 'Design Package/Theme' (set to 'M2 Cookbook Sample Theme'), 'Widget Title' (set to 'Sample content'), 'Assign to Store Views' (set to 'All Store Views'), and 'Sort Order' (set to '1'). A tooltip for 'Assign to Store Views' lists 'All Store Views', 'Main Website', 'Main Website Store', and 'Default Store View'. Below the 'Sort Order' field is a note: 'Sort Order of widget instances in the same container'.

- Specify the layout options; here, we configure the page and container where the content must be visible. It is possible to select multiple locations for the same block:

Page	Container	Template
CMS Home Page	Main Content Area	CMS Static Block Default Template

Display on: Specified Page

Add Layout Update

- The last step is to select the block to display. For this, go to the **Widget Options** tab and click on the **Select Block** button. From the list shown, you can now select the block that you want to add:

Block: Sample Homepage

Select Block...

- Refresh the cache:

```
bin/magento cache:clean
```

- Refresh the home page and check whether the created block is shown.

How it works...

The widget created will build the layout XML for the details that you have selected and will be loaded from the database while generating the layout. It is possible to add multiple Layout Updates and select multiple locations where the widget should be shown.

Available widgets

By default, Magento ships with the following widgets:

CMS Page Link

The CMS Page Link widget will allow you to add a link to a page that you specify; this can be useful to add a link to the footer.

CMS Static Block

The CMS Static Block widget will add a static block to the location that you specify (as shown in this recipe).

Catalog Category Link

The Catalog Category Link widget adds a link to a specific category that you specify.

Catalog New Products List

The Catalog New Products List widget allows you to add a list of products to a page; here, you can select the amount of products that you want to show and if you want to display only new products or all products.

Catalog Product Link

With the Catalog Product Link widget, it is possible to create a link to a specific product.

Catalog Product List

To display a list of products on your page, you can use the Catalog Product List widget. With this widget, you can control the products shown based on your own conditions (product attributes).

Orders and Returns

This will add a block to allow customers to search for their orders and view the status or request for a return (Enterprise).

Recently Compared Products

This will add a block that shows the products that are added to the compare products list.

Recently Viewed Products

This will add a block that shows the products that have been viewed by the user.

Using a dynamic serving theme based on the client browser

Using a responsive theme allows you to build a single theme that has the same look and feel through all devices and will also generate more traffic and slower loads on devices with a smaller viewport if you don't want to show the same information to them. With dynamic serving, you can assign a custom theme with your own layout configuration based on the User-Agent string that is sent by the client browser. This allows you to show only the content that is relevant to that user's device; this can be useful for images shown on the home page and also with the way you display your product details.

Getting ready

In order to use dynamic serving, you must first know what devices you want to show a different layout for.

How to do it...

In this recipe, we will see how to create a new theme that depends on the theme created in the *Creating a new theme* recipe of this chapter. Remove the desktop definition CSS and show this theme to users of the iPhone only.

1. Create the theme definition file:

```
app/design/frontend/Genmato/mobile/theme.xml

<theme xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation=
"urn:magento:framework:Config/etc/theme.xsd">
    <title>M2 Cookbook Sample Mobile Theme</title>
    <parent>Genmato/default</parent>
    <media>
        <preview_image>media/preview.png</preview_image>
    </media>
</theme>
```

2. Create `preview.png` for how your theme will look. (This file needs to be present but you can start with a blank file and replace this later when your theme is done.) Place this preview image under `app/design/frontend/Genmato/mobile/media/`.
3. To install the theme through Composer, we need to create a `composer.json` file:

```
app/design/frontend/Genmato/mobile/composer.json

{
    "name": "genmato/mobile-theme",
```

```
"description": "Genmato Sample Mobile Theme",
"require": {
    "php": "~5.5.0|~5.6.0|~7.0.0",
    "genmato/default-theme": "1.0.*",
    "magento/framework": "100.0.*"
},
"type": "magento2-theme",
"version": "1.0.0",
"license": [
    "OSL-3.0",
    "AFL-3.0"
],
"autoload": {
    "files": [
        "registration.php"
    ]
}
}
```

4. In order to register the theme when loaded through Composer, it needs `registration.php`:

```
app/design/frontend/Genmato/mobile/registration.php
```

```
<?php
\Magento\Framework\Component\ComponentRegistrar::register(
    \Magento\Framework\Component\ComponentRegistrar::THEME,
    'frontend/Genmato/mobile',
    __DIR__
);
```

5. Create a static file directory structure in your theme:

```
app/design/frontend/Genmato/mobile
```

```
web/
web/css/source/
web/fonts/
web/images/
web/js/
```

-
6. Remove the desktop CSS file as specified by the parent blank theme:

```
app/design/frontend/Genmato/mobile/Magento_Theme/layout/
default_head_blocks.xml
```

```
<?xml version="1.0"?>
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:
View/Layout/etc/page_configuration.xsd">
<head>
    <remove src="css/styles-l.css"/>
</head>
</page>
```

7. Add a custom static block to the mobile home page:

```
app/design/frontend/Genmato/mobile/Magento_Cms/layout/cms_
index_index.xml
```

```
<?xml version="1.0"?>
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:
View/Layout/etc/page_configuration.xsd">
<body>
    <referenceContainer name="content">
        <block class="Magento\Cms\Block\Block"
name="homepage-content">
            <arguments>
                <argument name="block_id"
xsi:type="string">sample-homepage</argument>
            </arguments>
        </block>
    </referenceContainer>
</body>
</page>
```

8. After uploading the files to your Magento installation, refresh the cache to load the new theme:

```
bin/magento cache:clean
```

- In the Magento backend, navigate to the store theme configuration:

Stores | Configuration | [General] Design.

Here, we can create the design exception based on the user agent that is sent by the browser:

Design Theme M2 Cookbook Sample Theme [STORE VIEW]

User-Agent Exceptions

Search String iphone Add

Design Theme Action

-- No Theme --

✓ M2 Cookbook Sample Mobile Theme

M2 Cookbook Sample Theme

Magento Blank

Magento Luma

Search strings are either normal strings or regular exceptions (PCRE). They are matched in the same order as entered. Examples:
Firefox
/^mozilla/i

- After saving the configuration, the cache must be refreshed again:

```
bin/magento cache:clean
```

- Now, the theme shown should be different when the website is accessed through an iPhone and normal browser on your desktop.

How it works...

During the page load process, the theme for the request is selected. When a User-Agent exception is found, the `Magento\Framework\View\DesignExceptions` class' `getThemeByRequest` method will check whether it matches the User-Agent sent by the client. If there is a match found, the theme specified will be used; otherwise, the default theme will be used.

The mobile theme used in this recipe only has a small number of changes; you can further optimize your theme by optimizing every page and removing components that aren't necessary to be displayed on the device that the theme is designed for.

Creating theme-specific translations

Magento offers a powerful system to translate strings used in templates, e-mails, and other components. There are several locations where you can place translations; they are loaded in the following order:

1. Magento database (inline translations).
2. Theme translations located in <theme>/i18n/<locale>.csv.
3. Parent theme translations (until no further parent is specified).
4. Translation packages located in app/i18n/<locale>.
5. Module translations located in <module>/i18n/.

Getting ready

In this recipe, we will change a translation for the en_US language; if you want, you can add other languages to your theme also.

How to do it...

This recipe will use the theme created in the *Creating a new theme* recipe of this chapter, but you can apply it to your own custom theme also.

1. Create your local translations file:

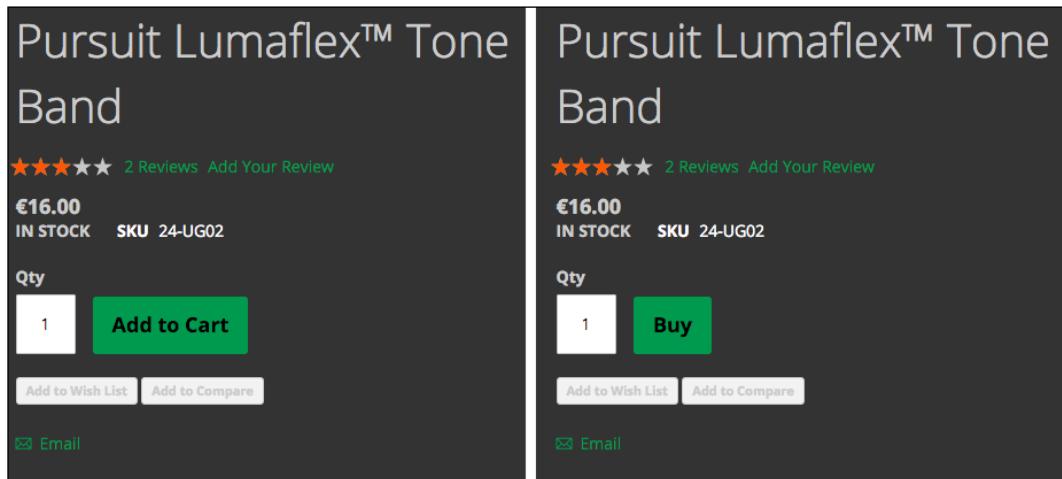
```
app/design/frontend/Genmato/default/i18n/en_US.csv
```

```
"Add to Cart", "Buy"
```

2. After uploading the file to your Magento installation, refresh the cache:

```
bin/magento cache:clean
```

3. If you now reload the page, the new translation should be visible:



How it works...

In Magento, all translatable strings are used in the following ways:

- ▶ Template files (.phtml):

```
<?php echo __('[text to translate]') ?>
```

- ▶ UI component templates:

```
<span data-bind="i18n: '[text to translate]'></span>
```

- ▶ XML files:

```
<item name="label" xsi:type="string" translate="true">[text to translate]</item>
```

- ▶ JavaScript files: (To use translations in JavaScript files, you need to include the mage/translate module through RequireJS.)

```
define(['jquery', 'mage/translate'], function ($) { ... });
```

- ▶ Next, you can use the translation in your script:

```
$.mage.__('<text to translate>');
```

The translation files in the theme are stored as a comma-separated file, where you can specify the original word/phrase and translation in the following format:

```
"[original string]", "[translation]"
```

It is important that you place only a single translation on a line. Additionally, the original string must match the case; otherwise, no translation can be done.

Some translations use dynamic values in the translation; these translations are created with %x (where x is a number) in the string:

```
"Quantity was recalculated from %1 to %2", "Quantity was  
recalculated from %1 to %2"
```

Generating a translation file

In Magento 2, there is now an easy option to generate a translation file for your theme or module. To generate the custom translations file for your theme, you can use the following command:

```
bin/magento i18n:collect-phrases -output="  
/app/design/frontend/Genmato/default/i18n/en_US.csv"  
app/design/frontend/Genmato/default
```

This command will output only translatable strings that are used in your template. It will not include files from your parent theme(s) or module template files. The generated file can now be edited and, if necessary, you can add extra lines with translations for phrases that are not used in your own theme but you still want to translate.

7

Creating Magento 2 Extensions – the Basics

In this chapter, we will cover the basics on how to create your own Magento 2 extensions with the following recipes:

- ▶ Initializing extension basics
- ▶ Working with database models
- ▶ Creating tables using setup scripts
- ▶ Creating a web route and controller to display data
- ▶ Creating system configuration fields
- ▶ Creating a backend data grid
- ▶ Creating a backend form to add/edit data

Introduction

Now that we have installed and configured Magento 2, we will see the basics on how to create a new Magento 2 extension. If you are familiar with creating an extension for Magento 1.x, you will notice that there are some concepts that look a lot like how it is done in Magento 1. However, as the code is a completely new framework, there are also lots of changes in the module structure and necessary files.

Some of the major changes that change the way a extension is created are as follows:

- ▶ **A new module structure:** The directory structure has changed, moving all parts of an extension into a single container, and files are no longer spread between different locations. This makes it easier to maintain an extension.

- ▶ **Configuration files:** In Magento 2, the configuration of an extension is split into multiple smaller files that are validated by an **XML Schema Definition (XSD)** schema. Additionally, some configuration files can be area-specific (adminhtml, frontend, cron, and api) to overwrite the general settings.
- ▶ **Namespaces and dependency injection:** In Magento 2, the code uses PHP namespaces to identify class names. Additionally, classes necessary are no longer loaded through `Mage::getModel()` (or similar functions), but injected into your class through a dependency injection.

Initializing extension basics

The way in which a new extension is built in Magento 2 is a bit different than it was in Magento 1. The major change is that all files are now included in the extension directory. This makes it easier to manage and remove.

The location of an extension is also different; there are no longer separate codepools as used in Magento 1 (core, community, and local). Depending on the way the extension is installed, the extension will be running from the `vendor` directory when installed through Composer. For project-specific extensions, it is also possible to place them in `app/code`.

Getting ready

When developing an extension, it is advised to run Magento 2 in developer mode as this will give better error messages explaining what went wrong and make debugging a lot easier.

To display all PHP errors and activate developer mode, activate the `display_errors` setting in `app/bootstrap.php` and run the following command:

```
bin/magento deploy:mode:set developer
```

The code used in this chapter is based on naming the module `Genmato_Sample` and all files will be placed in the following:

```
app/code/Genmato/Sample/
```

How to do it...

Follow these steps to initialize a new extension:

1. Create the module initialization file:

```
etc/module.xml:
```

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/
  XMLSchema-instance" xsi:noNamespaceSchemaLocation=
```

```
"urn:magento:framework:Module/etc/module.xsd">
<module name="Genmato_Sample" setup_version="0.1.3">
    <sequence>
        <module name="Magento_Store"/>
    </sequence>
</module>
</config>
```

2. Create the module registration file:

`registration.php`

```
<?php

\Magento\Framework\Component\ComponentRegistrar::register(
    \Magento\Framework\Component\ComponentRegistrar::MODULE,
    'Genmato_Sample',
    __DIR__
);
```

3. Create the module `composer.json` file:

`composer.json`

```
{
    "name": "genmato/sample",
    "description": "Genmato Magento2 Sample extension",
    "keywords": ["magento2", "genmato", "m2sample"],
    "type": "magento2-module",
    "license": "OSL-3.0",
    "require": {
        "php": "~5.5.0|~5.6.0|~7.0.0"
    },
    "autoload": {
        "files": [ "registration.php" ],
        "psr-4": {
            "Genmato\\Sample\\": ""
        }
    }
}
```

4. Enable the module with Magento. To do this, run the following command:

`bin/magento module:enable Genmato_Sample`

5. Run the upgrade command to register the module:

`bin/magento setup:upgrade`

How it works...

The module declaration in step 1 registers the module in Magento; here, the version number and dependencies are also specified to manipulate the load order. The following XML nodes are available:

- ▶ **Module name:** Genmato_Sample
- ▶ **Setup version:** 0.1.3 is used for the setup/upgrade scripts creating database schemas
- ▶ **Sequence:** Here we can define the modules that this extension is depending on

With the use of Composer, modules can be placed in two locations currently. In order for the autoloader to know what file to load when a class is instantiated, the path needs to be registered. Through `registration.php` from step 2, the (`__DIR__`) location is stored for a specified package. The registration can be done for the following types of packages:

- ▶ **MODULE:** This is for extensions/modules
- ▶ **LIBRARY:** This is for extensions that are used as a library
- ▶ **THEME:** This is for themes
- ▶ **LANGUAGE:** This is for language packs

In order to use Composer to install the module, it is required to add a `composer.json` file to the module. The most important elements in this file are as follows:

- ▶ **Name:** The extension package name is in the format of `<vendor name>/<extension name>`; it is important to only use lowercase letters. This name is also used to install the extension through Composer.
- ▶ **Type:** This defines the package type; the possible options are as follows:
 - `magento2-module`: Extensions/modules
 - `magento2-theme`: Themes
 - `magento2-language`: Language packages
- ▶ **require:** This defines the packages needed to be installed on the system for this package to work. During installation of the package, Composer will check the requirements and try to install the missing packages. When it's not possible to install them, the installation will fail.
- ▶ **Autoload:** This element is to specify information that needs to be loaded through the Composer autoloader.

There's more...

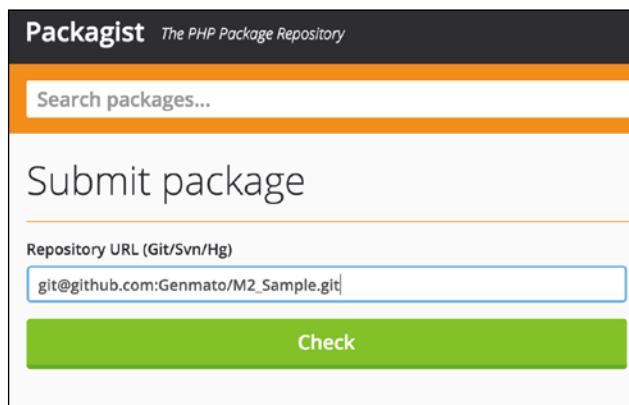
In order to install packages in your project, they have to be available for Composer to install. There are a few options available to install packages from:

- ▶ **Magento marketplace:** Currently, installing Magento 2 through Composer is done from the Magento repository. In the future, paid and free modules/packages bought through the marketplace will be available through the same repository through your account. This will make installing and managing packages easy.
- ▶ **From version control repository:** For private packages that you use in your project, it is possible to install them directly from your version control repository; for this, you need to add the following to the `composer.json` file located in your Magento 2 installation root:

```
{
  "repositories": [
    {
      "type": "vcs",
      "url": "https://github.com/[github
        account]/[package]"
    }
  ]
}
```

- ▶ **Packagist:** For freely available packages, it is also possible to register them on Packagist, which is the default repository that is available in Composer to install packages. To add a Magento 2 extension to Packagist, you will need to store the extension in a public repository, which can be GitHub (<http://www.github.com>) or BitBucket (<http://www.bitbucket.com>).

On Packagist, you can submit your package by specifying your extension repository URL:



Every package is now installable by running the following command in your Magento 2 installation root:

```
composer require <vendor-name>/<module-name>
```

Working with database models

Storing data in a database table is handled through a `Model` class; this model holds the data. While saving the data, a `ResourceModel` is used, and this class is the link between the `Model` and database table, and all CRUD operations go through the `ResourceModel`. When loading a set of records, a `Collection` is used; it is possible to apply filters to this collection.

Getting ready

Using database models requires that the module configuration is done correctly; otherwise, the autoloader won't be able to find the files to load.

How to do it...

The following steps in this recipe will add the database models to your module:

1. Create the `Model` class:

`Model/Demo.php`

```
<?php
namespace Genmato\Sample\Model;
use Magento\Framework\Model\AbstractModel;
class Demo extends AbstractModel
{
    /**
     * Initialize resource model
     * @return void
     */
    protected function _construct()
    {
        $this->_init('Genmato\Sample\Model\ResourceModel\
            Demo');
    }
}
```

2. Create the ResourceModel class:

Model/ResourceModel/Demo.php

```
<?php
namespace Genmato\Sample\Model\ResourceModel;
use Magento\Framework\Model\ResourceModel\Db\AbstractDb;
class Demo extends AbstractDb
{
    /**
     * Initialize resource model
     * @return void
     */
    protected function _construct()
    {
        $this->_init('genmato_demo', 'demo_id');
    }
}
```

3. Create the Collection class:

Model/ResourceModel/Demo/Collection.php

```
<?php
namespace Genmato\Sample\Model\ResourceModel\Demo;
use Magento\Framework\Model\ResourceModel\Db\Collection\AbstractCollection;
class Collection extends AbstractCollection
{
    /**
     * @var string
     */
    protected $_idFieldName = 'demo_id';

    /**
     * Define resource model
     * @return void
     */
    protected function _construct()
    {
        $this->_init('Genmato\Sample\Model\Demo',
            'Genmato\Sample\Model\ResourceModel\Demo');
    }
}
```

How it works...

The Model class specifies the used ResourceModel in the constructor through the `_init()` function. When the `save()` function is called on a Model, it will call the `save()` function on the ResourceModel; see the `save()` function in `AbstractModel` that is extended:

```
/**  
 * Save object data  
 *  
 * @return $this  
 * @throws \Exception  
 */  
public function save()  
{  
    $this->_getResource()->save($this);  
    return $this;  
}
```

Here, `_getResource()` returns an instance of the class specified in the `_init()` function.

In the ResourceModel class constructor, the `_init()` function is called to specify the `genmato_demo` database table and primary key in the `demo_id` table. This will be used when creating the queries necessary to load or save the data. Depending on the action performed (save new, save existing, or delete), a corresponding query is generated using an INSERT, UPDATE, or DELETE query. This is handled by the framework and is built (currently) on the `Zend_Db_Adapter_Pdo_Mysql` class.

In the Collection class, both Model and ResourceModel are specified in the `_init()` function. ResourceModel is necessary to connect to the database and load the records from the right database table, allowing you to use filters to select the records necessary. The collection is then represented as an array of Models to allow all functionality available to models (magic getters/setters, adding data, and delete/save).

Creating tables using setup scripts

When using database models, as explained in the previous recipe, the corresponding tables needs to be created during setup. These operations are placed in setup scripts and executed during the installation of an extension.

Getting ready

While running the installation of a module, there are four files executed to create schemas and insert data. To create schemas, the files used are as follows:

`Setup/InstallSchema.php`

`Setup/UpgradeSchema.php`

The installation file is executed only when there is no record in the `setup_module` table for the module. The upgrade file is executed only when the current version number in the `setup_module` table is lower than the version configured in your `etc/module.xml` file.

When it's necessary to insert default values into a table or new EAV attributes need to be created, these actions need to be configured in the following files:

`Setup/InstallData.php`

`Setup/UpgradeData.php`

Magento keeps track of which version is installed for an extension in the `setup_module` table; here, the current installed version for the schema and data is stored:

Objects				setup_module @magento2rc (L...)	
	module	schema_version	data_version		
▶	Genmato_Sample	0.1.3	0.1.3		
	Magento_AdminNotification	2.0.0	2.0.0		
	Magento_AdvancedPricingImportExport	2.0.0	2.0.0		
	Magento_Authorization	2.0.0	2.0.0		
	Magento_Authorizenet	2.0.0	2.0.0		
	Magento_Backend	2.0.0	2.0.0		

How to do it...

Follow these steps to create your database tables:

1. The following is the table schema installation:

`Setup/InstallSchema.php`

```
<?php
namespace Genmato\Sample\Setup;
```

```
use Magento\Framework\Setup\InstallSchemaInterface;
```

```
use Magento\Framework\Setup\ModuleContextInterface;
use Magento\Framework\Setup\SchemaSetupInterface;
use Magento\Framework\DB\Adapter\AdapterInterface;

class InstallSchema implements InstallSchemaInterface
{
    public function install(SchemaSetupInterface $setup,
                           ModuleContextInterface $context)
    {
        $installer = $setup;

        $installer->startSetup();

        /**
         * Create table 'genmato_demo'
         */
        $table = $installer->getConnection()->newTable(
            $installer->getTable('genmato_demo')
        )->addColumn(
            'demo_id',
            \Magento\Framework\DB\Ddl\Table::TYPE_SMALLINT,
            null,
            ['identity' => true, 'nullable' => false, 'primary' => true],
            'Demo ID'
        )->addColumn(
            'title',
            \Magento\Framework\DB\Ddl\Table::TYPE_TEXT,
            255,
            ['nullable' => false],
            'Demo Title'
        )->addColumn(
            'creation_time',
            \Magento\Framework\DB\Ddl\Table::TYPE_TIMESTAMP,
            null,
            [],
            'Creation Time'
        )->addColumn(
            'update_time',
            \Magento\Framework\DB\Ddl\Table::TYPE_TIMESTAMP,
            null,
            [],
            'Modification Time'
        )->addColumn(
```

```

'is_active',
\Magento\Framework\DB\Ddl\Table::TYPE_SMALLINT,
null,
['nullable' => false, 'default' => '1'],
'Is Active'
) ->addIndex(
    $setup->getIdxName(
        $installer->getTable('genmato_demo'),
        ['title'],
        AdapterInterface::INDEX_TYPE_FULLTEXT
    ),
    ['title'],
    ['type' => AdapterInterface::INDEX_TYPE_FULLTEXT]
) ->setComment(
    'Demo Table'
);
$installer->getConnection()->createTable($table);

$installer->endSetup();
}
}

```

2. Trigger the execution of the setup scripts:

bin/magento setup:upgrade

How it works...

In Magento 2, the running of the setup scripts is no longer triggered by the first request after flushing the cache; to initiate the running of these scripts, run the command specified in step 2. When running the upgrade command, all modules are evaluated on their current version and module version in the configuration file. First, all schema installations/updates are executed, and next, the data installations/updates are processed.

The `InstallData` and `InstallSchema` files are executed only when there is no prior registration of the extension in the `setup_module` table. To run the installation files during testing, it is possible to remove the module row from the table and run the `bin/magento setup:upgrade` command.

The available methods to create a new table are defined in the `Magento\Framework\DB\Adapter\AdapterInterface\Table` class and are as follows:

- ▶ `addColumn`: This adds a new column to the table; this method has the following parameters:
 - `name`: This is the name of the table
 - `type`: This is the table type; the available column types are defined as constants in the `Magento\Framework\DB\Adapter\AdapterInterface\Table` class as `TYPE_*`:
 - `TYPE_BOOLEAN`
 - `TYPE_SMALLINT`
 - `TYPE_INTEGER`
 - `TYPE_BIGINT`
 - `TYPE_FLOAT`
 - `TYPE_NUMERIC`
 - `TYPE_DECIMAL`
 - `TYPE_DATE`
 - `TYPE_TIMESTAMP`
 - `TYPE_DATETIME`
 - `TYPE_TEXT`
 - `TYPE_BLOB`
 - `TYPE_VARBINARY`
 - `size`: This specifies the size of the column
 - `options`: This is used to specify extra column options; the available options are as follows:
 - `unsigned`: This is only for number types; allows True/False (default: False)
 - `precision`: This is only for decimal and numeric types (default: calculated from size parameter or 0 if not set)
 - `scale`: This is only for decimal and numeric types (default: calculated from size parameter or 10 if not set)
 - `default`: The default value is used when creating a new record
 - `nullable`: In case a column is NULL (default: True)
 - `primary`: This makes a column a primary key
 - `primary_position`: This is only for primary keys and sets the sort order for the primary keys

- identity/auto_increment: This auto-increments a column on inserting a new record (used to identify a unique record ID)
- ❑ comment: This is the description of the column
- addForeignKey: This adds a foreign key relation to another table; the parameters allowed are as follows:
- ❑ fkName: This is the name of the foreign key
 - ❑ column: This is the column used as the foreign key
 - ❑ refTable: This is the table where the key references to
 - ❑ refColumn: This is the column name in the referenced table
 - ❑ onDelete: This sets the action to be performed when deleting a record; the available options are (constants as defined in Magento\Framework\DB\Adapter\AdapterInterface\Table):
 - ACTION CASCADE
 - ACTION RESTRICT
 - ACTION SET DEFAULT
 - ACTION SET NULL
 - ACTION NO ACTION
- addIndex: This adds a column to the search index; the available parameters are as follows:
- ❑ indexName: This is the name used for the index
 - ❑ fields: These are the column(s) used for the index (can be a single column or an array of columns)
 - ❑ options: This is an array with extra options; currently, only the option type is used to specify the index type

When changing an existing table, it is possible to use the following methods; these are the methods that can be used directly on the \$installer->getConnection() class:

- dropTable: This removes a table from the database; the available parameters are as follows:
- ❑ tableName: This is the name of the table to delete
 - ❑ schemaName: This is the optional schema name used

- ▶ `renameTable`: This renames a table from the database; the available parameters are as follows:
 - `oldTableName`: This is the current name of the table
 - `newTableName`: This is the new name for the table
 - `schemaName`: This is the optional schema name
- ▶ `addColumn`: This adds an extra column to a table; the available parameters are as follows:
 - `tableName`: This is the name of the table to alter
 - `columnName`: This is the name of the new column
 - `definition`: This is an array with the following parameters:
 - `Type`: Column type
 - `Length`: Column size
 - `Default`: Default value
 - `Nullable`: If a column can be NULL
 - `Identify/Auto_Increment`: Used as an identity column
 - `Comment`: Column description
 - `After`: Specify where to add the column
 - `schemaName`: This is the optional schema name
- ▶ `changeColumn`: This changes the column name and definition; the available parameters are as follows:
 - `tableName`: This is the name of the table to change
 - `oldColumnName`: This is the current column name
 - `newColumnName`: This is the new name for the column
 - `definition`: This is the table definition; see `addColumn` for available values
 - `flushData`: This flushes the table cache
 - `schemaName`: This is the optional schema name
- ▶ `modifyColumn`: This changes the column definition; the available parameters are as follows:
 - `tableName`: This is the name of the table
 - `columnName`: This is the column name to change
 - `definition`: This is the table definition; see `addColumn` for available values

- flushData: This flushes the table cache
 - schemaName: This is the optional schema name
- ▶ dropColumn: This removes a column from the table; the available parameters are as follows:
 - tableName: This is the name of the column
 - columnName: This is the name of the column to remove
 - schemaName: This is the optional schema name
- ▶ addIndex: This adds a new index; the available parameters are as follows:
 - tableName: This is the name of the table to change
 - indexName: This is the name of the index to add
 - fields: These are the columns to be used as the index
 - indexType: This is the type of index; the available options (constants defined in `(Magento\Framework\DB\Ddl\Table\AdapterInterface)`) are as follows:
 - INDEX_TYPE_PRIMARY
 - INDEX_TYPE_UNIQUE
 - INDEX_TYPE_INDEX
 - INDEX_TYPE_FULLTEXT
 - schemaName: This is the optional schema name
- ▶ dropIndex: This removes an index from a table; the available parameters are as follows:
 - tableName: This is the name of the column
 - indexName: This is the name of the index
 - schemaName: This is the optional schema name
- ▶ addForeignKey: This adds a new foreign key; the available parameters are as follows:
 - fkName: This is the name of the foreign key
 - tableName: This is the name of the table
 - columnName: This is the name of the column used in the foreign key
 - refTableName: This is the name of the referenced table
 - refColumnName: This is the name of the referenced column
 - onDelete: This is the action to perform on delete (see the preceding `addForeignKey` description for available options)

- ❑ `purge`: This removes invalid data (default: false)
- ❑ `schemaName`: This is the optional schema name
- ❑ `refSchemaName`: This is the option-referenced schema name

There's more...

When, in a later version of the extension, there are extra fields necessary (or the current fields need to be changed), this is handled through the `UpgradeSchema` function, `upgrade`:

`Setup/UpgradeSchema.php`

```
<?php  
namespace Genmato\Sample\Setup;  
  
use Magento\Framework\DB\Ddl\Table;  
use Magento\Framework\Setup\UpgradeSchemaInterface;  
use Magento\Framework\Setup\ModuleContextInterface;  
use Magento\Framework\Setup\SchemaSetupInterface;  
  
class UpgradeSchema implements UpgradeSchemaInterface  
{  
    public function upgrade(SchemaSetupInterface $setup,  
                           ModuleContextInterface $context)  
    {  
        $setup->startSetup();  
  
        if (version_compare($context->getVersion(), '0.1.1', '<')) {  
            $connection = $setup->getConnection();  
  
            $column = [  
                'type' => Table::TYPE_SMALLINT,  
                'length' => 6,  
                'nullable' => false,  
                'comment' => 'Is Visible',  
                'default' => '1'  
            ];  
            $connection->addColumn($setup->getTable('genmato_demo'),  
                                   'is_visible', $column);  
        }  
  
        $setup->endSetup();  
    }  
}
```

As this file is run every time the module version is different than the currently installed version, it is necessary to check the current version that is installed to execute only the updates necessary:

```
if (version_compare($context->getVersion(), '0.1.1', '<')) {
```

The preceding statement will make sure that the schema changes are executed only if the current version is less than 0.1.1.

Data installation

In order to provide default content during installation (this can be records in a table or adding extra attributes to some entity), the data installation function is used:

Setup/InstallData.php

```
<?php
namespace Genmato\Sample\Setup;

use Genmato\Sample\Model\Demo;
use Genmato\Sample\Model\DemoFactory;
use Magento\Framework\Setup\InstallDataInterface;
use Magento\Framework\Setup\ModuleContextInterface;
use Magento\Framework\Setup\ModuleDataSetupInterface;

class InstallData implements InstallDataInterface
{
    /**
     * Demo factory
     *
     * @var DemoFactory
     */
    private $demoFactory;

    /**
     * Init
     *
     * @param DemoFactory $demoFactory
     */
    public function __construct(DemoFactory $demoFactory)
    {
        $this->demoFactory = $demoFactory;
    }

    /**
     * {@inheritDoc}
     */
```

```
* @SuppressWarnings(PHPMD.ExcessiveMethodLength)
*/
public function install(ModuleDataSetupInterface $setup,
    ModuleContextInterface $context)
{
    $demoData = [
        'title' => 'Demo Title',
        'is_active' => 1,
    ];

    /**
     * Insert demo data
     */
    $this->createDemo()->setData($demoData)->save();

}

/**
 * Create demo
 *
 * @return Demo
 */
public function createDemo()
{
    return $this->demoFactory->create();
}
```

In this example, there is one record created in the table created during setup. For this, the `DemoFactory` class is injected through dependency injection into the constructor function of this class. `DemoFactory` is an automatically created class that allows you to instantiate a class (in this case, `Genmato\Sample\Model\Demo`) without injecting this directly into the constructor. Here, this is done in the `createDemo` function:

```
$this->demoFactory->create();
```

Similar to `SchemaUpgrade`, there is also a `DataUpgrade` option to insert data while upgrading to a newer version.

Creating a web route and controller to display data

In order to display data from your extension on the frontend (the public part of the website), the following is necessary:

- ▶ A configured route
- ▶ A controller handling the request
- ▶ A layout file to specify what to show
- ▶ The block class as specified in the layout file
- ▶ A template file (optional)

How to do it...

Follow these steps to extend your module with a frontend web route and output data from a template file:

1. Create a route in the frontend area:

```
etc/frontend/routes.xml

<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/
  XMLSchema-instance" xsi:noNamespaceSchemaLocation=
  "urn:magento:framework:App/etc/routes.xsd">
  <router id="standard">
    <route id="sample" frontName="sample">
      <module name="Genmato_Sample" />
    </route>
  </router>
</config>
```

2. Create the controller that handles the request and renders the output:

```
Controller/Index/Index.php

<?php
namespace Genmato\Sample\Controller\Index;
use Magento\Framework\App\Action\Action;

class Index extends Action
{
    /**
     * @var \Magento\Framework\View\Result\PageFactory
```

```
 */
protected $resultPageFactory;

/**
 * @param \Magento\Framework\App\Action\Context $context
 * @param \Magento\Framework\View\Result\PageFactory
 *      resultPageFactory
 */
public function __construct(
    \Magento\Framework\App\Action\Context $context,
    \Magento\Framework\View\Result\PageFactory
    $resultPageFactory
)
{
    $this->resultPageFactory = $resultPageFactory;
    parent::__construct($context);
}

/**
 * Renders Sample Index
 */
public function execute()
{
    return $this->resultPageFactory->create();
}
}
```

3. Create the layout file to specify what to display:

```
view/frontend/layout/sample_index_index.xml

<?xml version="1.0"?>
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    layout="1column" xsi:noNamespaceSchemaLocation=
    "urn:magento:framework:View/Layout/etc/
    page_configuration.xsd">
    <head>
        <title>Sample DemoList</title>
    </head>
    <body>
        <referenceContainer name="content">
            <block class="Genmato\Sample\Block\DemoList"
                name="demoList" template=
                "Genmato_Sample::list.phtml" />
        </referenceContainer>
    </body>
</page>
```

-
4. In order to show the data, the `Block` class is called to render the template specified:

`Block/DemoList.php`

```
<?php
namespace Genmato\Sample\Block;

use Magento\Framework\View\Element\Template;
use Genmato\Sample\Model\ResourceModel\Demo\Collection as
    DemoCollection;
use Magento\Store\Model\ScopeInterface;

class DemoList extends Template
{
    /**
     * Demo collection
     *
     * @var DemoCollection
     */
    protected $_demoCollection;

    /**
     * Demo resource model
     *
     * @var \Genmato\Sample\Model\ResourceModel\Demo\
        CollectionFactory
     */
    protected $_demoColFactory;

    /**
     * @param Template\Context $context
     * @param \Genmato\Sample\Model\ResourceModel\Demo\
        CollectionFactory $collectionFactory
     * @param array $data
     * @SuppressWarnings(PHPMD.ExcessiveParameterList)
     */
    public function __construct(
        Template\Context $context,
        \Genmato\Sample\Model\ResourceModel\Demo\
            CollectionFactory $collectionFactory,
        array $data = []
    ) {
        $this->_demoColFactory = $collectionFactory;
        parent::__construct(
            $context,
```

```
        $data
    );
}

/***
 * Get Demo Items Collection
 * @return DemoCollection
 */
public function getDemoItems()
{
    if (null === $this->_demoCollection) {
        $this->_demoCollection =
            $this->_demoColFactory->create();
    }
    return $this->_demoCollection;
}
}
```

5. In the template, the data collected in the Block class can be used to build the page:
- view/frontend/templates/list.phtml

```
<table>
<tr>
    <td>ID</td>
    <td>Title</td>
</tr>
<?php foreach($block->getDemoItems() as $item): ?>
<tr>
    <td><?php echo $item->getId(); ?></td>
    <td><?php echo $item->getTitle();?></td>
</tr>
<?php endforeach; ?>
</table>
```

6. Refresh the cache to update the configuration:

```
bin/magento cache:clean
```

In your browser, open `http://[your hostname]/sample/index/index/`. This will result in the following page when you access the URL:

The screenshot shows a web browser displaying a Magento storefront. At the top is a header with the LUMA logo and navigation links for 'What's New', 'Women', 'Men', 'Gear', 'Training', and 'Sale'. Below the header is a large, stylized title 'Sample DemoList'. Underneath the title is a table with one row, representing a product. The table has two columns: 'ID' and 'Title'. The value '1' is in the 'ID' column, and 'Demo Title' is in the 'Title' column.

ID	Title
1	Demo Title

 If you have not set the deploy mode to developer, it is possible that the accessed URL will not render. If this is the case, you need to run the compile command:
`bin/magento setup:di:compile`

How it works...

When a request is received in the application, the path is evaluated and executed in the following order:

1. A request is received by `index.php`.
2. The `index.php` file creates a bootstrap:

```
$bootstrap = \Magento\Framework\App\Bootstrap::create(BP, $_SERVER);
```

3. The bootstrap creates a new HTTP application:

```
$app = $bootstrap->createApplication('Magento\Framework\App\Http');
```

4. The bootstrap application is started:

```
$bootstrap->run($app);
```

5. In the `run` function of the `bootstrap` class, the created application is launched:

```
$response = $application->launch();
```

6. The application launch function will instantiate the `frontcontroller` and dispatch the request:

```
$frontController = $this->_objectManager->get(  
    'Magento\Framework\App\FrontControllerInterface');  
$result = $frontController->dispatch($this->_request);
```

7. The `frontcontroller` loops through all the available configured controllers and checks whether there is a match. When a match is found, the controller is instantiated and the `execute` method is called:

```
$result = $actionInstance->execute();
```

8. This will evaluate the layout file for the route loaded and render the blocks that are specified there.

To activate a route on the frontend, it needs to be configured by specifying the first part of the URL. This first part maps the request to the extension that will handle the request.

Here, `frontName` (`sample`) is mapped to the extension, `Genmato_Sample`, allowing it to handle all requests on the frontend.

In order to handle the request to a URL, there needs to be a controller. In Magento 2, the controller now handles only one action (whereas Magento 1 has a controller that has the option to handle multiple actions). Every request consists of three parts:

[route] / [controller] / [action]

[route]: This is the configured `frontName`

[controller]: This is the path to identify the controller

[action]: This is the action that is executed (the PHP class)

In this case, a request is made to the following:

`http://example.com/sample/`

This will result with a request to the following:

`http://example.com/sample/index/index`

This will load the following file:

`Genmato\Sample\Controller\Index\Index.php`

The executed controller relies on the layout file to render the page by including the specified blocks in the page. Just like the controllers, the layout files are now separate files per page handle. The name of the file is built in the same way as the controllers:

```
[route]_[controller]_[action].xml
```

This makes it easier to change a specific page handle in a theme.

The loaded `Block` class is the location to handle the collection of the data necessary to be shown in the template (just like in Magento 1). Here, we load the collection from the database so that it can be used in the template.

Creating system configuration fields

In Magento, it is possible to store configuration values for global/website or store in the backend. These values can be used to store simple module settings such as API -keys, module enable/disable options, or any setting that you might require for your module. The data is stored in the `core_config_data` table.

Getting ready

As the configuration fields are only accessible through the backend web pages, the configuration file is stored in the `etc/adminhtml` directory.

How to do it...

Create your own configuration options with the following step:

1. Create the system configuration file:

```
etc/adminhtml/system.xml
```

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Config:etc/system_file.xsd">
<system>
    <section id="sample" translate="label" type="text"
        sortOrder="2000" showInDefault="1" showInWebsite="1"
        showInStore="1">
        <label>Sample Configuration</label>
        <tab>general</tab>
        <resource>Genmato_Sample::config_sample</resource>
        <group id="demo" translate="label" type="text"
            sortOrder="100" showInDefault="1" showInWebsite="1"
            showInStore="1">
```

```
<label>Sample</label>
<field id="header" translate="label" type="text"
    sortOrder="1" showInDefault="1" showInWebsite="1"
    showInStore="1">
    <label>Header Title</label>
</field>
<field id="selectsample" translate="label"
    type="select" sortOrder="2" showInDefault="1"
    showInWebsite="1" showInStore="1">
    <label>Sample Select</label>
    <source_model>Genmato\Sample\Model\Config\Source\
        DemoList</source_model>
</field>
</group>
</section>
</system>
</config>
```

How it works...

The Magento store configuration is divided in separate sections and contains multiple groups of configuration fields. Every section is assigned to a main tab that is displayed above the sections.

Creating a new tab

When the configuration options that you want to add can't fit in one of the existing tabs, it is possible to add your own through the `system.xml` file. For this, add the following to your system configuration file:

```
<tab id="" translate="label"
    sortOrder="" class="">
    <label>[label]</label>
</tab>
```

In the configuration, you can use the following attributes:

- ▶ **id:** This is a unique identifier for the tab, which is also used to reference to the tab in your sections
- ▶ **translate:** This specifies the elements that need to be available for translation (in this case, only the label field is available)
- ▶ **sortOrder:** This is the numeric value to use to sort the tabs
- ▶ **class:** This is the optional class name for your tab

The label element is used as the visible name of the tab in the configuration; it is important to make it as descriptive as possible for customers to understand.

Creating a new section

Every section is shown below the tab referenced in the configuration. In every section, multiple groups can be configured that can hold multiple fields. Adding a new section can be done by adding the following to your `system.xml` file:

```
<section id="<unique_section_name>" translate="label"
    type="<text>" sortOrder="<sort order>" showInDefault="<0/1>"
    showInWebsite="<0/1>" showInStore="<0/1>">
    <class>[class]</class>
    <header_css>[header_css]</header_css>
    <label>[label]</label>
    <tab>[tab]</tab>
    <resource>[resource]</resource>
</section>
```

The attributes used are as follows:

- ▶ `id`: This is the unique section identifier
- ▶ `translate`: This specifies the elements that need to be available for translation (in this case, only the `label` field is available)
- ▶ `type`: This is the type of field used (normally text)
- ▶ `sortOrder`: This is the numeric value to use to sort the sections
- ▶ `showInDefault`: This shows sections in the default store configuration
- ▶ `showInWebsite`: This displays sections in the website configuration
- ▶ `showInStore`: This displays sections in the store configuration

The available elements in this configuration are as follows:

- ▶ `class`: This is the class used for the section
- ▶ `header_css`: This is the CSS class to use in the text header for the section
- ▶ `label`: This is the text to display in the section
- ▶ `tab`: This is the reference to the tab where the section should be added
- ▶ `resource`: This is the **access control list (ACL)** resource referenced, which is used to check whether the user logged in has access to the section

Creating a new group

In a section, it is possible to create multiple groups; they are displayed as separate fieldsets that can be expanded/collapsed and can hold one or multiple fields. To create a new group, add the following (in the section to hold the group) to the `system.xml` file:

```
<group id="<group_id>" translate="label" type="text"
    sortOrder="<sort order>" showInDefault="<0/1>"
    showInWebsite="<0/1>" showInStore="<0/1>">
    <label>[label]</label>
</group>
```

The attributes to configure a group are as follows:

- ▶ `id`: This is the unique ID for the group (in the section)
- ▶ `translate`: These are the fields that needs to be translated, in this case, only the `label` option
- ▶ `type`: This is the field type (text in this case)
- ▶ `sortOrder`: This is the numeric value used to sort the groups
- ▶ `showInDefault`: This shows a group in the default store configuration
- ▶ `showInWebsite`: This shows a group in the website configuration
- ▶ `showInStore`: This shows a group in the store configuration

The `label` element is shown as the heading of the group.

Creating a new field

Every group can have one or multiple fields. In the database, the configuration is stored as a path build, `<section>/<group>/<field>`. A new config field can be one of the following types: text input, textarea, select, multiselect, or a custom data renderer. To add a field, add the following to a group:

```
<field id="<field_id>" translate="label" type="<type>"
    sortOrder="<sort order>" showInDefault="<0/1>"
    showInWebsite="<0/1>" showInStore="<0/1>">
    <label>[comment]</label>
    <comment>[comment]</comment>
</field>
```

The attributes used for the field configuration are as follows:

- ▶ `id`: This is the unique ID of the field in the group
- ▶ `translate`: These are the fields that need to be translated (can be the `label`, `comment`, `tooltip`, or `hint` elements)

- ▶ **type:** This is the input type, which can be the following:
 - **text:** Default text input field
 - **textarea:** Text area input
 - **obscure:** Password input
 - **select:** Drop-down select

This needs a `source_model` element to specify the available options to select.
- multiselect:** This is the multiple item select box. This needs a `source_model` element to specify the available options to select.
- image:** This is the image upload with a preview if the file is uploaded. This needs different `backend_model` (to store the uploaded file), `upload_dir` (to specify where to save the file), and `base_url` (path used to build the URL to access from the frontend) elements.

- ▶ **sortOrder:** This is the numeric value used to sort the fields in the group
- ▶ **showInDefault:** This shows fields in the default store configuration
- ▶ **showInWebsite:** This shows fields in the website configuration
- ▶ **showInStore:** This shows fields in the store configuration

There are also some extra elements available to configure the field:

- ▶ **label:** This is the label, which shows in front of the input field
- ▶ **comment:** This is the comment shown below the input field
- ▶ **tooltip:** This is the text shown when hovering the question mark
- ▶ **frontend_class:** This is the class used for the field
- ▶ **frontend_model:** This is the custom frontend model (block) that will be used to render the input area
- ▶ **backend_model:** This is the backend model used when saving the data to process inserted values
- ▶ **source_model:** This is the data source used for select and multiselect fields
- ▶ **depends:** This is the option to hide/show a field based on a value used by another field, for example:

```
<depends>
  <field id="[field_name_to_check]">[value_to_show]</field>
</depends>
```

When `[field_name_to_check]` has the `[value_to_show]` value, the field will be visible, if it contains a different value, then the field is hidden for the user.

There's more...

To get the data stored in the sample/demo/header field, add the following to the template Block class in our template block:

Block/DemoList.php

```
/**  
 * Get Header from configuration value  
 * @return string  
 */  
public function getHeader()  
{  
    return $this->scopeConfig->getValue('sample/demo/header',  
        ScopeInterface::SCOPE_STORE);  
}
```

The _scopeConfig class is injected through **dependency injection (DI)** into the Block class in the constructor and allows us to read data stored in the configuration.

Next, we want to display the data in our template by adding the following line at the top of the file:

```
view/frontend/templates/list.phtml  
<p><?php echo $this->getHeader();?></p>
```

Creating a backend data grid

Adding a page to the backend requires, just like the frontend, a configured route, controller, and layout file. In order to display a grid page to show data from a table, there are currently three ways available:

- ▶ Creating a grid container and specifying the fields to display and data source to use in the grid class. This method is similar to how a grid is built in Magento 1 and is not really flexible/easy to extend. An example of how this is used can be found in the CMS Page module:

Magento\Cms\Block\Adminhtml\Page

Magento\Cms\Block\Adminhtml\Page\Grid

- ▶ Using this method, there is only a grid container `Block` class created. The grid fields and options are defined in the layout XML file. This makes it possible to extend the grid easily by adding extra fields to the XML. An example of this can be found in the Customer Group code in the following:

`Magento\Customer\view\adminhtml\layout\customer_group_index.xml`

- ▶ The last option is fully configured through XML and gives the option to specify the data source, quick search, available advanced filters, mass actions, row actions, and columns to show. It also gives you the option to customize the columns shown in the backend by the user.

In this example, we will use the last option to build a grid. As this option uses quite a large set of files and long XML listings, the complete code can be found on GitHub (https://github.com/Genmato/M2_Sample).

How to do it...

The following steps will describe how to add a backend data grid:

1. Configure routes for use in the `adminhtml` area:

`etc/adminhtml/routes.xml`

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/
  XMLSchema-instance" xsi:noNamespaceSchemaLocation=
  "urn:magento:framework:App/etc/routes.xsd">
  <router id="admin">
    <route id="sample" frontName="sample">
      <module name="Genmato_Sample"
        before="Magento_Backend" />
    </route>
  </router>
</config>
```

2. Create the Controller for the backend:

`Controller/Adminhtml/Demolist/Index.php`

```
<?php
namespace Genmato\Sample\Controller\Adminhtml\Demolist;

use Magento\Backend\App\Action\Context;
use Magento\Framework\View\Result\PageFactory;
```

```
use Magento\Backend\App\Action as BackendAction;

class Index extends BackendAction
{
    /**
     * @var PageFactory
     */
    protected $resultPageFactory;

    /**
     * @param Context $context
     * @param PageFactory $resultPageFactory
     */
    public function __construct(
        Context $context,
        PageFactory $resultPageFactory
    ) {
        parent::__construct($context);
        $this->resultPageFactory = $resultPageFactory;
    }
    /**
     * Check the permission to run it
     *
     * @return bool
     */
    protected function _isAllowed()
    {
        return $this->authorization->isAllowed(
            'Genmato_Sample::demolist');
    }

    /**
     * Index action
     *
     * @return \Magento\Backend\Model\View\Result\Page
     */
    public function execute()
    {
        /** @var \Magento\Backend\Model\View\Result\Page
         * $resultPage */
        $resultPage = $this->resultPageFactory->create();
        $resultPage->setActiveMenu('Genmato_Sample::demolist');
        $resultPage->addBreadcrumb(___('CMS')) , ___('CMS'));
    }
}
```

```

$resultPage->addBreadcrumb(__('Demo List'), __('Demo
List'));
$resultPage->getConfig()->getTitle()->prepend(__('Demo
List'));

return $resultPage;
}
}

```

3. Control the access to the page through the ACL:

etc/acl.xml

```

<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/
  XMLSchema-instance" xsi:noNamespaceSchemaLocation=
  "urn:magento:framework:Acl/etc/acl.xsd">
  <acl>
    <resources>
      <resource id="Magento_Backend::admin">
        <resource id="Magento_Backend::content">
          <resource id="Magento_Backend::content_elements">
            <resource id="Genmato_Sample::demolist"
              title="Demo List" sortOrder="10" />
          </resource>
        </resource>
      </resource>
    </resources>
  </acl>
</config>

```

4. The following is the layout file to specify the grid uiComponent used:

view/adminhtml/layout/sample_demolist_index.xml

```

<?xml version="1.0"?>
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="urn:magento:framework:
  View/Layout/etc/page_configuration.xsd">
  <update handle="styles"/>
  <body>
    <referenceContainer name="content">
      <uiComponent name="sample_demolist_listing"/>
    </referenceContainer>
  </body>
</page>

```

5. Create the uiComponent configuration:

The referenced uiComponent configuration is quite large. In the sample code (on GitHub), this file can be found at the following:

https://github.com/mage2cookbook/M2_Sample/blob/master/view/adminhtml/ui_component/sample_demo_list_listing.xml

6. Add resources used in uiComponent to the dependency injection configuration:

etc/di.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/
  XMLSchema-instance" xsi:noNamespaceSchemaLocation=
  "urn:magento:framework:ObjectManager/etc/config.xsd">
<preference for="Genmato\Sample\Model\DemoInterface"
  type="Genmato\Sample\Model\Demo" />

<type name="Magento\Framework\View\Element\UiComponent\
  DataProvider\CollectionFactory">
  <arguments>
    <argument name="collections" xsi:type="array">
      <item name="sample_demo_list_listing_data_source"
        xsi:type="string">Genmato\Sample\Model\
          ResourceModel\Demo\Grid\Collection</item>
    </argument>
  </arguments>
</type>
<type name="Genmato\Sample\Model\ResourceModel\Demo\
  Grid\Collection">
  <arguments>
    <argument name="mainTable" xsi:type="string">
      genmato_demo</argument>
    <argument name="eventPrefix" xsi:type="string">
      sample_demo_list_grid_collection</argument>
    <argument name="eventObject" xsi:type="string">
      sample_demo_list_collection</argument>
    <argument name="resourceModel" xsi:type="string">
      Genmato\Sample\Model\ResourceModel\Demo</argument>
  </arguments>
</type>
<virtualType name="DemoGridFilterPool" type="Magento\
  Framework\View\Element\UiComponent\DataProvider\
  FilterPool">
  <arguments>
    <argument name="appliers" xsi:type="array">
```

```

<item name="regular" xsi:type="object">
    Magento\Framework\View\Element\UiComponent\
        DataProvider\RegularFilter</item>
<item name="fulltext" xsi:type="object">
    Magento\Framework\View\Element\UiComponent\
        DataProvider\FulltextFilter</item>
</argument>
</arguments>
</virtualType>
<virtualType name="DemoGridDataProvider" type="Magento\
    Framework\View\Element\UiComponent\DataProvider\
    DataProvider">
<arguments>
    <argument name="collection" xsi:type="object"
        shared="false">Genmato\Sample\Model\ResourceModel\
            Demo\Collection</argument>
    <argument name="filterPool" xsi:type="object"
        shared="false">DemoGridFilterPool</argument>
</arguments>
</virtualType>
</config>

```

7. Add the mass action controller:

```

Controller/Adminhtml/Demolist/MassDelete.php

<?php
namespace Genmato\Sample\Controller\Adminhtml\Demolist;

use Magento\Framework\Controller\ResultFactory;
use Magento\Backend\App\Action\Context;
use Magento\Ui\Component\MassAction\Filter;
use Genmato\Sample\Model\ResourceModel\Demo\CollectionFactory;
use Magento\Backend\App\Action;

class MassDelete extends Action
{
    /**
     * @var CollectionFactory
     */
    protected $collectionFactory;

    /**
     * @param Context $context
     * @param Filter $filter
     */

```

```
* @param CollectionFactory $collectionFactory
*/
public function __construct(Context $context, Filter
    $filter, CollectionFactory $collectionFactory)
{
    $this->filter = $filter;
    $this->collectionFactory = $collectionFactory;
    parent::__construct($context);
}

/**
 * Execute action
 *
 * @return \Magento\Backend\Model\View\Result\Redirect
 */
public function execute()
{
    $collection = $this->filter->getCollection(
        $this->collectionFactory->create());
    $collectionSize = $collection->getSize();

    foreach ($collection as $item) {
        $item->delete();
    }

    $this->messageManager->addSuccess(__('A total of %1
record(s) have been deleted.', $collectionSize));

    /**
     * @var \Magento\Backend\Model\View\Result\Redirect
     */
    $resultRedirect = $this->resultFactory->create(
        ResultFactory::TYPE_REDIRECT);
    return $resultRedirect->setPath('*/*/');
}
}
```

8. Add options to the menu:

etc/adminhtml/menu.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/
  XMLSchema-instance" xsi:noNamespaceSchemaLocation=
  "urn:magento:module:Magento_Backend:etc/menu.xsd">
<menu>
```

```

<add id="Genmato_Sample::demolist" title="Demo List"
    module="Genmato_Sample" sortOrder="10"
    parent="Magento_Backend::content_elements"
    action="sample/demolist"
    resource="Genmato_Sample::demolist"/>
</menu>
</config>

```

How it works...

The controller and layout file are the same for the frontend, only the backend is protected through an ACL. This allows administrators to create specific user rules and allow access only to the selected pages.

The resource access is checked in the controller in the following function:

```
protected function _isAllowed()
```

The uiComponent configuration

As the complete configuration through uiComponent results in a large XML file, we will explain some parts on how they are configured and hook in to the system.

Data source

The data source is specified through the XML node:

```

<listing>
<dataSource name="sample_demolist_listing_data_source">
<argument name="dataProvider" xsi:type="configurableObject">
<argument name="class" xsi:type="string">
    DemoGridDataProvider</argument>

```

The specified data class, `DemoGridDataProvider`, is configured through dependency injection and specified in the `etc/di.xml` file. Here, `DemoGridDataProvider` corresponds to the `Genmato\Sample\Model\ResourceModel\Demo\Collection` collection. Additionally, `DemoGridFilterPool` and the collection are configured through this file to load the data.

Mass actions for grid

It is also easy to specify multiple mass actions to be used in the grid; you can specify the title, action to execute, and optional message alert box:

```

<massaction name="listing_massaction">
<argument name="data" xsi:type="array">
    <item name="config" xsi:type="array">

```

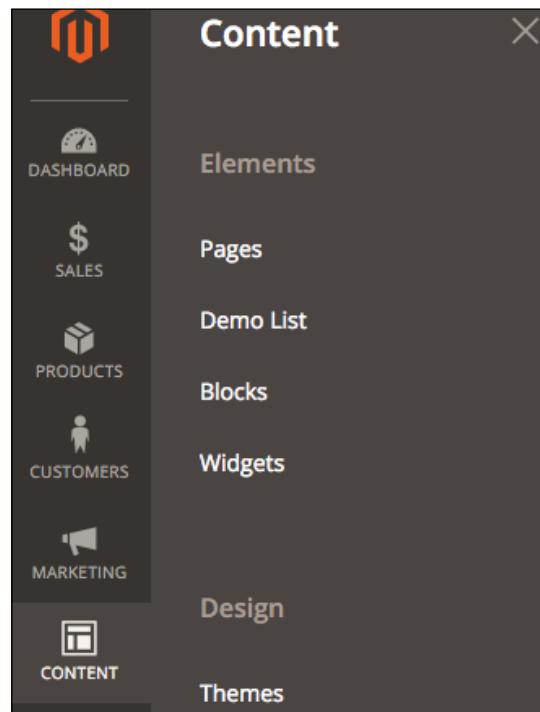
```
<item name="selectProvider" xsi:type="string">
    sample_demolist_listing.sample_demolist_listing.
    sample_demolist_columns.ids</item>
<item name="indexField" xsi:type="string">demo_id</item>
</item>
</argument>
<action name="delete">
<argument name="data" xsi:type="array">
<item name="config" xsi:type="array">
<item name="type" xsi:type="string">delete</item>
<item name="label" xsi:type="string" translate
    ="true">Delete</item>
<item name="url" xsi:type="url" path=
    "sample/demolist/massDelete"/>
<item name="confirm" xsi:type="array">
<item name="title" xsi:type="string"
    translate="true">Delete items</item>
<item name="message" xsi:type="string" translate="true">
    Are you sure you want to delete selected items?
    </item>
</item>
</item>
</argument>
</action>
</massaction>
```

In this example, we add an option to delete records from the database. In order to delete the records from the database, you need to create a controller that handles the removal of the records. You need to create a **Controller** class for every action you specify, this controller contains your custom code to be executed.

By adding the menu option in the **Content** menu below the **Pages** menu item, the page can be accessed through the backend. This link action will go to the specified `sample/demolist` URL that will result in the full URL:

`http://example.com/admin/sample/demolist/`

The `resource` argument defines the ACL that is used to show/hide the menu option depending on the user rights:



When clicking on the menu option, the resulting page will look as follows:

A screenshot of a "Demo List" page. The page has a header with the title "Demo List" and a search bar containing the keyword "demo". Below the search bar is a red button labeled "Add New Item". The main content area shows a table with one record. The table has columns for "ID" and "Title". The ID column contains a checkbox next to the value "1". The Title column contains the value "Demo Title". At the bottom of the table, there are buttons for "Select Items" and "Action". Above the table, there are filters, a "Default View" dropdown, and a "Columns" dropdown. Below the table, there are buttons for "20" records per page, navigation arrows, and a "Clear all" link.

ID	Title	Action
<input type="checkbox"/> 1	Demo Title	Select ▾

See also

For more information about uiComponents that are available, you can refer to the following:

<http://devdocs.magento.com/guides/v2.0/ui-components/ui-component.html>

Creating a backend form to add/edit data

If you want to add a new record or edit an existing record, it is possible to create a form to have a user friendly way to process the data. In this example, we will see how to create a form to edit an existing record. The path used to add a record will be as follows:

<http://example.com/admin/sample/demolist/new/>

This will require the controller name to be new, but as this is a reserved word in PHP, the class name used will be `newAction`. The execute function is not only used to add a new record, but it can also be used to edit an existing record. In the following code, only the execute action is shown; see the sample code for the complete source.

Getting ready

In this recipe, we will add the option to add or edit records; for this, the route used in the previous chapter is used, only new controllers and blocks are shown.

How to do it...

Follow these steps to add a form to your module:

1. Add the controller:

```
Controller/Adminhtml/Demolist/NewAction.php

<?php
namespace Genmato\Sample\Controller\Adminhtml\Demolist;
use Magento\Backend\App\Action;
class NewAction extends Action
{
    public function execute()
    {
        $demoId = $this->getRequest()->getParam('demo_id');
```

```

$this->_coreRegistry->register('current_demo_id',
    $demoId);

/** @var \Magento\Backend\Model\View\Result\Page
 * $resultPage */
$resultPage = $this->resultPageFactory->create();
if ($demoId === null) {
    $resultPage->addBreadcrumb(__('New DemoList'),
        __('New DemoList'));
    $resultPage->getConfig()->getTitle()->prepend(__('New
        DemoList'));
} else {
    $resultPage->addBreadcrumb(__('Edit DemoList'),
        __('Edit DemoList'));
    $resultPage->getConfig()->getTitle()->
        prepend(__('Edit DemoList'));
}
// Build the edit form
$resultPage->getLayout()->addBlock(
    'Genmato\Sample\Block\Adminhtml\Demo>Edit',
    'demolist', 'content')
->setEditMode((bool)$demoId);

return $resultPage;
}
}

```

2. Create the Form container:

Block/Adminhtml/Demo/Edit.php

```

<?php
namespace Genmato\Sample\Block\Adminhtml\Demo;
use Magento\Backend\Block\Widget\Form\Container;
class Edit extends Container
{
    /**
     * Remove Delete button if record can't be deleted.
     *
     * @return void
     */
    protected function _construct()
    {
        $this->_objectId = 'demo_id';
        $this->controller = 'adminhtml_demo';
    }
}

```

```
$this->_blockGroup = 'Genmato_Sample';
parent::__construct();
$demoId = $this->getDemoId();
if (!$demoId) {
    $this->buttonList->remove('delete');
}
}

/**
 * Retrieve the header text, either editing an existing
 * record or creating a new one.
 *
 * @return \Magento\Framework\Phrase
 */
public function getHeaderText()
{
    $demoId = $this->getDemoId();
    if (!$demoId) {
        return __('New DemoList Item');
    } else {
        return __('Edit DemoList Item');
    }
}

public function getDemoId()
{
    if (!$this->demoId) {
        $this->demoId=$this->coreRegistry->
            registry('current_demo_id');
    }
    return $this->demoId;
}
}
```

3. Build the form by defining the fields and types that are used:

Block/Adminhtml/Demo/Edit/Form.php

```
<?php
namespace Genmato\Sample\Block\Adminhtml\Demo>Edit;
use Magento\Customer\Controller\RegistryConstants;
use Magento\Backend\Block\Widget\Form\Generic;
class Form extends Generic
{
    /**
     *
```

```
* Prepare form for render
*
* @return void
*/
protected function _prepareLayout()
{
    parent::_prepareLayout();

/** @var \Magento\Framework\Data\Form $form */
$form = $this->_formFactory->create();

$demoId = $this->_coreRegistry->registry(
    'current_demo_id');
/** @var \Genmato\Sample\Model\DemoFactory $demoData */
if ($demoId === null) {
    $demoData = $this->demoDataFactory->create();
} else {
    $demoData = $this->demoDataFactory->create()->load(
        $demoId);
}

$yesNo = [];
$yesNo[0] = 'No';
$yesNo[1] = 'Yes';

$fieldset = $form->addFieldset('base_fieldset',
    ['legend' => __('Basic Information')]);

$fieldset->addField(
    'title',
    'text',
    [
        'name' => 'title',
        'label' => __('Title'),
        'title' => __('Title'),
        'required' => true
    ]
);

$fieldset->addField(
    'is_active',
    'select',
    [
        'name' => 'is_active',
```

```
'label' => __('Active'),
'title' => __('Active'),
'class' => 'required-entry',
'required' => true,
'values' => $yesNo,
]
);

$fieldset->addField(
    'is_visible',
    'select',
    [
        'name' => 'is_visible',
        'label' => __('Visible'),
        'title' => __('Visible'),
        'class' => 'required-entry',
        'required' => true,
        'values' => $yesNo,
    ]
);
}

if ($demoData->getId() !== null) {
    // If edit add id
    $form->addField('demo_id', 'hidden', ['name' =>
        'demo_id', 'value' => $demoData->getId()]);
}

if ($this->backendSession->getDemoData()) {
    $form->addValues($this->backendSession->
        getDemoData());
    $this->backendSession->setDemoData(null);
} else {
    $form->addValues(
        [
            'id' => $demoData->getId(),
            'title' => $demoData->getTitle(),
            'is_active' => $demoData->getIsActive(),
            'is_visible' => $demoData->getIsVisible(),
        ]
    );
}

$form->setUseContainer(true);
$form->setId('edit_form');
$form->setAction($this->getUrl('*/*/save'));
```

```

        $form->setMethod('post');
        $this->setForm($form);
    }
}

```

4. Add the **Save** action controller, which is called when pressing the **Submit** button:

Controller/Adminhtml/Demolist/Save.php

```

<?php
namespace Genmato\Sample\Controller\Adminhtml\Demolist;
use Magento\Backend\App\Action;
class Save extends Action
{
    /**
     * Save DemoList item.
     *
     * @return \Magento\Backend\Model\View\Result\Page|\Magento\Backend\Model\View\Result\Redirect
     */
    public function execute()
    {
        $id = $this->getRequest()->getParam('demo_id');
        $resultRedirect = $this->resultRedirectFactory->
            create();
        try {
            if ($id !== null) {
                $demoData = $this->demoFactory->create()->
                    load((int)$id);
            } else {
                $demoData = $this->demoFactory->create();
            }
            $data = $this->getRequest()->getParams();
            $demoData->setData($data)->save();

            $this->messageManager->addSuccess(__('Saved DemoList
                item.'));
            $resultRedirect->setPath('sample/demolist');
        } catch (\Exception $e) {
            $this->messageManager->addError($e->getMessage());
            $this->_getSession()->setDemoData($data);

            $resultRedirect->setPath('sample/demolist/edit',
                ['demo_id' => $id]);
        }
        return $resultRedirect;
    }
}

```

How it works...

In the controller, the `demo_id` parameter is stored in the Magento registry; this makes the data available to retrieve in the same request in other blocks/models to do further processing with it. Additionally, the `Genmato\Sample\Block\Adminhtml\Demo>Edit` class is loaded. This class will generate the container for the form and load the `Genmato\Sample\Block\Adminhtml\Demo\Form` class that will generate the form to be shown.

Building the form

Every form is built from the following:

```
$form = $this->_formFactory->create();
```

As it is not possible to add fields to a form directly, you need to create a fieldset first. It is possible to add multiple fieldsets to a single form and assign multiple form fields to a fieldset. To create a fieldset, use the following command:

```
$fieldset = $form->addFieldset(' [name] ', ['legend' => __([heading])]);
```

Adding a form field

Adding a form field to the fieldset is done as follows:

```
$fieldset->addField([elementId], [type], [config], [after]);
```

This method has the following parameters:

- ▶ `elementId`: This is the unique name of the form field element.
- ▶ `type`: This defines the type of element that is used as the input field. This can be your own class that implements the `Magento\Framework\Data\Form\Element\AbstractElement` class or one of the following:
 - ❑ button
 - ❑ checkbox
 - ❑ checkboxes
 - ❑ column
 - ❑ date
 - ❑ editablemultiselect
 - ❑ editor
 - ❑ fieldset
 - ❑ file
 - ❑ gallery

- hidden
- image
- imagefile
- label
- link
- multiline
- multiselect
- note
- obscure
- password
- radio
- radios
- reset
- select
- submit
- text
- textarea
- time

These input types are rendered from the classes found under the `Magento\Framework\Data\Form\Element` directory.

- ▶ **config:** This is an array of extra configuration data; some of the options are as follows:
 - name:** This is the name of the element from the data model that holds the data
 - label:** This is the text that is used as a label
 - title:** This is the text that is used as the field title parameter
 - class:** This is the optional class for the input field, which can be used for form validation
 - required:** This is optional. This is set if a field is required to have data.
 - value:** This is the value used for select/multiselect
- ▶ **after:** This is the optional parameter to specify where the form field needs to be placed; use `elementId` of the form field that you want to place this field after.

Loading the data

In order to display the data currently stored for the record that we want to edit, the stored `demo_id` is retrieved from the registry:

```
$demoId = $this->_coreRegistry->registry('current_demo_id');
```

Next, we check whether there is a valid value stored as `demoId` and load the data from the database (or just set an empty object):

```
/** @var \Genmato\Sample\Model\DemoFactory $demoData */
if ($demoId === null) {
    $demoData = $this->demoDataFactory->create();
} else {
    $demoData = $this->demoDataFactory->create()->load($demoId);
}
```

Now that the data is loaded, it is possible to set this data to the form fields:

```
$form->addValues(
[
    'id' => $demoData->getId(),
    'title' => $demoData->getTitle(),
    'is_active' => $demoData->getIsActive(),
    'is_visible' => $demoData->getIsVisible(),
]
);
```

Saving the data

When submitting the form to save the data, the action and method should be set to the URL that handles the save action:

```
$form->setAction($this->getUrl('*/*/save'));
$form->setMethod('post');
```

The URL `*/*/save` maps to the `Controller/Adminhtml/Demolist/Save` controller and will handle the save action.

In the controller, we first load the current record to load all the current values:

```
if ($id !== null) {  
    $demoData = $this->demoFactory->create()->load((int)$id);  
} else {  
    $demoData = $this->demoFactory->create();  
}
```

Next, we retrieve the submitted data, store it in the loaded object, and save the object:

```
$data = $this->getRequest()->getParams();  
$demoData->setData($data)->save();
```


8

Creating Magento 2 Extensions – Advanced

In this chapter, we will cover some of the more advanced features when building Magento 2 extensions in the following recipes:

- ▶ Using dependency injection to pass classes to your own class
- ▶ Modifying functions with the use of plugins – Interception
- ▶ Creating your own XML module configuration file
- ▶ Creating your own product type
- ▶ Working with service layers/contracts
- ▶ Creating a Magento CLI command option

Introduction

In the previous chapter, we explained the basics on how to create a Magento 2 extension/module, including how to store data in a database and create a frontend page and backend grid with a form to add/edit data. In this chapter, we will explain some more advanced functions that can be used while developing custom extensions. The first two recipes are informational and describe the basic usage of dependency injection and plugins. The other recipes are examples on how to add your custom XML configuration file and commands to the Magento CLI tool.

Using dependency injection to pass classes to your own class

In Magento 2, they introduced the usage of dependency injection, which is a well-known design pattern that changes the way you use resources in the code. Using dependency injection, all the required resources are created when the class is instantiated instead of creating an object (through the Magento 1.x Mage class) when necessary. The benefit of this is that it is easier to use unit testing as it is possible to mock the required objects.

Getting ready

In this example, we will see how to create a new record in the demolist model created in the previous chapter. The record is created using an observer on the `sales_order_place_after` event that is dispatched after a new order is saved.

How to do it...

Follow these steps on how to use dependency injection:

1. First, we declare the Observer to listen to the event that we want:

`etc/events.xml`:

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:Event/etc/events.xsd">
<event name="sales_order_place_after">
<observer name="sample" instance="Genmato\Sample\Observer\PlaceOrder"/>
</event>
</config>
```

2. Next, we create the Observer class as defined in step 1:

`Observer/PlaceOrder.php`:

```
<?php
namespace Genmato\Sample\Observer;

use Magento\Framework\Event\ObserverInterface;
use Magento\Framework\Event\Observer as EventObserver;
use Genmato\Sample\Model\DemoFactory;

class PlaceOrder implements ObserverInterface
{
```

```
/**
 * @var DemoFactory
 */
protected $demoFactory;

/**
 * @param DemoFactory $demoFactory
 */
public function __construct(DemoFactory $demoFactory)
{
    $this->demoFactory = $demoFactory;
}

/**
 * Add record to demoList when new order is placed
 *
 * @param EventObserver $observer
 * @return void
 */
public function execute(EventObserver $observer)
{
    /** @var \Magento\Sales\Model\Order $order */
    $order = $observer->getEvent()->getOrder();

    $demoList = $this->demoFactory->create();

    $demoList->setTitle(__('New order (%1) placed!', $order->getIncrementId()));

    try {
        $demoList->save();
    } catch (\Exception $ex) {
        // Process error here....
    }
}
```

3. Refresh the cache:

```
bin/magento cache:clean
```

How it works...

In the Observer class defined in step 2, the object that we need to create a new record in is injected into the constructor function:

```
public function __construct(DemoFactory $demoFactory)
```

During the instantiation of the class, the necessary classes are injected by the Magento 2 **dependency injection (DI)** framework. In this case, it adds the `Genmato\Sample\Model\DemoFactory` class to the constructor. This class is not a real existing class; this is because the class that we want to use is a non-injectable class. A non-injectable class is a class that you create yourself through the `new [classname]` command or the `Mage::getModel()` call in Magento 1.x. With the use of dependency injection, creating a new object should be handled by the object manager, but in order to be able to use unit testing, it is not advisable to create the object directly in the code. To solve this, the Magento framework uses autogenerated classes. In this example, the generated class is stored in `var/generation/Genmato/Sample/Model/DemoFactory.php`:

```
<?php
namespace Genmato\Sample\Model;

/**
 * Factory class for @see \Genmato\Sample\Model\Demo
 */
class DemoFactory
{
    /**
     * Object Manager instance
     *
     * @var \Magento\Framework\ObjectManagerInterface
     */
    protected $_objectManager = null;

    /**
     * Instance name to create
     *
     * @var string
     */
    protected $_instanceName = null;

    /**
     * Factory constructor
     *
     * @param \Magento\Framework\ObjectManagerInterface
     * $objectManager
     */
```

```

* @param string $instanceName
*/
public function __construct(\Magento\Framework\
    ObjectManagerInterface $objectManager, $instanceName =
        '\\Genmato\\Sample\\Model\\Demo')
{
    $this->_objectManager = $objectManager;
    $this->_instanceName = $instanceName;
}

/**
 * Create class instance with specified parameters
 *
 * @param array $data
 * @return \Genmato\Sample\Model\Demo
 */
public function create(array $data = array())
{
    return $this->_objectManager->create($this->_instanceName,
        $data);
}
}

```

In this case, the autogenerated class is a Factory class that has only the `create()` function available. In the `create` function, the object manager is used to instantiate the `Genmato\Sample\Model\Demo` class; this object can then be used to load an existing record or create a new record and save the data stored in the object.

Modifying functions with the use of plugins – Interception

One of the biggest problems in Magento 1.x was that changing the behavior of a function in a class that didn't have an event trigger had to be rewritten. This works fine if only a single rewrite for a class was used, but when using a large number of extensions, there is a risk that multiple extensions rewrite the same class, which can result in unpredictable results.

In Magento 2, this problem is partly solved by introducing Interception in the form of plugins. With the use of plugins, it is possible to modify a function in three places:

- ▶ **Before execution:** With the before plugin, it is possible to (pre)process any data given to the original function by modifying the original function arguments; this allows you to replace the original method or change the input variables and supply them to the original method.

- ▶ **Around execution:** In the around plugin, you can modify both the input and output values of the original function. In your own function, you decide what action to do before the original function call and what will be done after it.
- ▶ **After execution:** The after plugin takes the result generated from the original function, modifies the result, and the modified result is then returned to the caller of the original function.

Getting ready

The use of plugins is controlled through `di.xml`. It is possible to use the global from `etc/` or the area specific from `etc/ [area]`; in this case, we will use the global, which means that it is used in all areas.

How to do it...

Create the following files in this recipe to try the use of plugins:

1. To enable the plugins, add the following lines to `di.xml`:

`etc/di.xml`

```
<type name="Magento\Cms\Model\Page">
    <plugin name="sample_before" type=
        "Genmato\Sample\Plugin\BeforePage" sortOrder="1"/>
</type>

<type name="Magento\Catalog\Model\Product">
    <plugin name="sample_around" type=
        "Genmato\Sample\Plugin\AroundProduct" sortOrder="1"/>
</type>

<type name="Magento\Cms\Model\Page">
    <plugin name="sample_after" type=
        "Genmato\Sample\Plugin\AfterPage" sortOrder="1"/>
</type>
```

2. The following is the before plugin class:

`Plugin/BeforePage.php`

```
<?php
namespace Genmato\Sample\Plugin;

use Magento\CMS\Model\Page;

class BeforePage
```

```
{  
    public function beforeSetContent(Page $subject, $content)  
    {  
        return $subject->setContent('<!--' . $content . '-->');  
    }  
}
```

3. The following is the around plugin class:

Plugin/AroundProduct.php

```
<?php  
  
namespace Genmato\Sample\Plugin;  
  
use Magento\Catalog\Model\Product;  
  
class AroundProduct  
{  
    public function aroundSave(Product $subject, \Closure  
        $proceed)  
    {  
        $subject->setMyCustomAttribute('sample');  
  
        $return = $proceed();  
  
        $subject->setMyCustomAttribute('');  
  
        return $return;  
    }  
}
```

4. The following is the after plugin class:

Plugin/AfterPage.php

```
<?php  
namespace Genmato\Sample\Plugin;  
  
use Magento\CMS\Model\Page;  
  
class AfterPage  
{  
  
    public function aftergetTitle(Page $subject, $result)  
    {
```

```
        return 'SAMPLE: '.$result;
    }

}
```

5. To refresh the cache, execute the following command:

```
bin/magento cache:clean
```

How it works...

All configured interceptors/plugins are evaluated during initialization. When a call is made to a function that is extended through a plugin, all plugin functions are executed based on the configured `sortOrder` as configured in `di.xml`. If there are multiple plugins that extend the same original function, they are executed in the following sequence:

1. The before plugin with the lowest `sortOrder`
2. The around plugin with the lowest `sortOrder`
3. Other before plugins (from the lowest to highest `sortOrder`)
4. Other around plugins (from the lowest to highest `sortOrder`)
5. The after plugin with the highest `sortOrder`
6. Other after plugins (from the highest to lowest `sortOrder`)

There are some limitations on where you can use plugins; it is not possible to use plugins for the following:

- ▶ Final methods/classes
- ▶ Non-public methods
- ▶ Class methods (such as static methods)
- ▶ Inherited methods
- ▶ `__construct`
- ▶ Virtual types

If you need to modify one of these types listed, the only option to do this is to rewrite (preference) your class through `di.xml`.

Creating your own XML module configuration file

In Magento 1.x, it was possible to use the .xml file to include custom configuration options that might be necessary for an extension. This is no longer possible with Magento 2 because the XML files are all validated against a schema and anything other than predefined options are not allowed. To solve this, it is possible to generate your own custom XML file to set up the parameters that you need. This also allows other extensions to define settings as the output is generated from all modules that have this file configured.

Getting ready

In order to use your own XML configuration file, it is important that you generate a valid schema (XSD) file that will be used to validate the XML files when they are merged.

How to do it...

The following steps show you how to define a custom XML configuration file for your module:

1. First, we create the Reader for the XML file and define the name of the file that should be read from all modules:

Model/Sample/Reader.php

```
<?php
namespace Genmato\Sample\Model\Sample;

use Magento\Framework\Config\Reader\Filesystem;
use Magento\Framework\Config\FileResolverInterface;
use Magento\Framework\Config\ConverterInterface;
use Genmato\Sample\Model\Sample\SchemaLocator;
use Magento\Framework\Config\ValidationStateInterface;

class Reader extends Filesystem
{
    protected $_idAttributes = [
        '/table/row' => 'id',
        '/table/row/column' => 'id',
    ];

    /**
     * @param FileResolverInterface $fileResolver
     * @param ConverterInterface $converter

```

```
* @param SchemaLocator $schemaLocator
* @param ValidationStateInterface $validationState
* @param string $fileName
* @param array $idAttributes
* @param string $domDocumentClass
* @param string $defaultScope
*/
public function __construct(
    FileResolverInterface $fileResolver,
    ConverterInterface $converter,
    SchemaLocator $schemaLocator,
    ValidationStateInterface $validationState,
    $fileName = 'sample.xml',
    $idAttributes = [],
    $domDocumentClass = 'Magento\Framework\Config\Dom',
    $defaultScope = 'global'
) {
    parent::__construct(
        $fileResolver,
        $converter,
        $schemaLocator,
        $validationState,
        $fileName,
        $idAttributes,
        $domDocumentClass,
        $defaultScope
    );
}
}
```

2. To validate the schema, the Reader must know where to find the schema file:

Model/Sample/SchemaLocator.php

```
<?php
namespace Genmato\Sample\Model\Sample;

use Magento\Framework\Config\SchemaLocatorInterface;
use Magento\Framework\Config\Dom\UrnResolver;

class SchemaLocator implements SchemaLocatorInterface
{
    /** @var UrnResolver */

```

```
protected $urnResolver;

public function __construct(UrnResolver $urnResolver)
{
    $this->urnResolver = $urnResolver;
}

/**
 * Get path to merged config schema
 *
 * @return string
 */
public function getSchema()
{
    return $this->urnResolver->getRealPath(
        'urn:genmato:module:Genmato_Sample:/etc/sample.xsd');
}

/**
 * Get path to pre file validation schema
 *
 * @return string
 */
public function getPerFileSchema()
{
    return $this->urnResolver->getRealPath(
        'urn:genmato:module:Genmato_Sample:/etc/sample.xsd');
}
```

3. A single class is used to get the merged data and cache the XML:

Model/Sample/Data.php

```
<?php
namespace Genmato\Sample\Model\Sample;

use Magento\Framework\Config\Data\Scoped;
use Genmato\Sample\Model\Sample\Reader;
use Magento\Framework\Config\ScopeInterface;
use Magento\Framework\Config\CacheInterface;

class Data extends Scoped
{
    /**

```

```
* Scope priority loading scheme
*
* @var array
*/
protected $_scopePriorityScheme = ['global'];

/**
* @param Reader $reader
* @param ScopeInterface $configScope
* @param CacheInterface $cache
* @param string $cacheId
*/
public function __construct(
    Reader $reader,
    ScopeInterface $configScope,
    CacheInterface $cache,
    $cacheId = 'sample_config_cache'
) {
    parent::__construct($reader, $configScope, $cache,
        $cacheId);
}
}
```

4. Add the XSD schema file:

etc/sample.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema">
    <xss:element name="table">
        <xss:complexType>
            <xss:sequence>
                <xss:element name="row" maxOccurs="unbounded"
                    minOccurs="0">
                    <xss:complexType>
                        <xss:sequence>
                            <xss:element name="column"
                                maxOccurs="unbounded" minOccurs="0">
                                <xss:complexType>
                                    <xss:sequence>
                                        <xss:element type="xs:string"
                                            name="label">
                                            <xss:annotation>
```

```

<xs:documentation>from first xml
    from second xmlthey appear in
    both xmls with the same path
    and id and second one overrides
    the value for `attr1` from
    first xml  from first
    xml</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
<xs:attribute type="xs:string"
    name="id" use="optional"/>
<xs:attribute type="xs:byte"
    name="sort" use="optional"/>
<xs:attribute type="xs:string"
    name="attr1" use="optional"/>
<xs:attribute type="xs:string"
    name="disabled" use="optional"/>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute type="xs:string" name="id"
    use="optional"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

```

5. Add the configuration file for your module:

etc/sample.xml

```

<?xml version="1.0"?>





```

```
<column id="col2" sort="20" disabled="true">
    attr1="val2" >
        <label>Col 2</label>
    </column>
    <column id="col3" sort="15" attr1="val1">
        <label>Col 3</label>
    </column>
</row>
</table>
```

6. Get and display the merged data: (This is optional; in this example, we display the data through a frontend route.)

Controller/Index/Sample.php

```
<?php
namespace Genmato\Sample\Controller\Index;

use Magento\Framework\App\Action\Action;
use Magento\Framework\App\Action\Context;
use Magento\Framework\View\Result\PageFactory;
use Genmato\Sample\Model\Sample\DataFactory;

class Sample extends Action
{
    /**
     * @var PageFactory
     */
    private $resultPageFactory;

    /** @var DataFactory $dataReader */
    private $dataReader;

    /**
     * @param Context $context
     * @param PageFactory $resultPageFactory
     * @param DataFactory $dataReader
     */
    public function __construct(
        Context $context,
        PageFactory $resultPageFactory,
        DataFactory $dataReader
    )
    {
```

```
$this->dataReader = $dataReader;  
parent::__construct($context);  
}  
  
/**  
 * Renders Sample  
 */  
public function execute()  
{  
    $myConfig = $this->dataReader->create();  
    print_r($myConfig->get());  
}  
}
```

7. Refresh the cache using the following command:

```
bin/magento cache:clean
```

8. Now you can check the result data using the following command:

<http://example.com/sample/index/sample/>

How it works...

The configuration reader defines the file that is used in the (`$fileName = 'sample.xml'`) constructor. Make sure that the filename used is unique; otherwise, configuration data from another module will be merged and validation will fail as it won't match the schema that you defined. A solution could be to use `<vendor> <module>.xml` as the filename.

In the constructor, `SchemaLocator` is also defined; this will define the schema (XSD file) that is used to validate the XML. To be able to get the schema file independent from where the module is installed (vendor/ or app/code), the schema location is built from the defined URN: `urn:genmato:module:Genmato_Sample:/etc/sample.xsd`. This URN is parsed and translated to the directory where the module is installed, which is done through `ComponentRegistrar`, and the module location is registered in `registration.php` as described in *Chapter 7, Creating Magento 2 Extensions – the Basics*.

It is possible to get all data using the `read()` method on the `Reader` class, this will result in re-reading and merging all XML files, which will impact every request. This can delay the website; therefore, the `Data` class is added. Here, `Reader` is injected through the constructor. To get the data, you can call the `get()` method of the `Data` class. This will read and merge all XML files if they are not cached and return the cached version when available. If you don't supply an argument to the `get()` method, it will return all data, but it's also possible to specify a node that you would like.

The Data class can be added everywhere that you need to get your configuration data; for this, you add Genmato\Sample\Model\Sample\DataFactory to the constructor. This autogenerated class allows you to instantiate your configuration data class:

```
$myConfig = $this->dataReader->create();
```

This gets a value from the configuration:

```
$myConfig->get('<node>');
```

Creating your own product type

In Magento 2, it is easy to add your own product type, which can be useful when you want to add some special features that are not available through the default product types. In this recipe, we will see a minimal new product type that calculates the price based on the cost of the product; you can easily extend it further to fit your own needs.

Getting ready

Every product type has its own unique code specified, and it's important to use a short code to identify your product type.

How to do it...

The following steps in this recipe show you how to create a (minimal) new product type:

1. First, we need to declare the new product type:

```
etc/product_types.xml

<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="urn:
magento:module:Magento_Catalog:etc/product_types.xsd">
    <type name="demo" label="Demo Product"
        modelInstance="Genmato\Sample\Model\Product\Type\Demo"
        indexPriority="80" sortOrder="80">
        <priceModel instance=
            "Genmato\Sample\Model\Product\Type\Demo\Price" />
        <customAttributes>
            <attribute name="refundable" value="true"/>
            <attribute name="taxable" value="true"/>
        </customAttributes>
    </type>
</config>
```

2. Now, we create `modelInstance` as configured in the product type definition; this is the minimal product type code definition. You can add your own functions that would be necessary to your product here:

```
Model/Product/Type/Demo.php

<?php
namespace Genmato\Sample\Model\Product\Type;

use Magento\Catalog\Model\Product\Type\AbstractType;

class Demo extends AbstractType
{
    /**
     * Product-type code
     */
    const TYPE_CODE = 'demo';

    /**
     * Delete data specific for Simple product-type
     *
     * @param \Magento\Catalog\Model\Product $product
     * @return void
     */
    public function deleteTypeSpecificData(
        \Magento\Catalog\Model\Product $product)
    {
    }
}
```

3. To calculate the price based on the cost attribute, `priceModel` is specified. This class holds the code to get the price of a product. Here, we use a fixed value of `1.25*cost` attribute. It is also possible to get this value from another attribute or system configuration field:

```
Model/Product/Type/Demo/Price.php

<?php
namespace Genmato\Sample\Model\Product\Type\Demo;

use Magento\Catalog\Model\Product\Type\Price as
ProductPrice;

class Price extends ProductPrice
{

    /**

```

```
* Default action to get price of product
*
* @param Product $product
* @return float
*/
public function getPrice($product)
{
    return $product->getData('cost')*1.25;
}
```

4. By default, the cost attribute is only available for all product types; therefore, we need to add our product type to the `apply_to` field of the cost attribute. This will be done through an `UpgradeData` script:

`Setup/UpgradeData.php`

```
<?php
namespace Genmato\Sample\Setup;

use Magento\Eav\Setup\EavSetup;
use Magento\Eav\Setup\EavSetupFactory;
use Magento\Framework\Setup\UpgradeDataInterface;
use Magento\Framework\Setup\ModuleContextInterface;
use Magento\Framework\Setup\ModuleDataSetupInterface;
use Magento\Catalog\Model\Product;

/**
 * @codeCoverageIgnore
 */
class UpgradeData implements UpgradeDataInterface
{
    /**
     * EAV setup factory
     *
     * @var EavSetupFactory
     */
    private $eavSetupFactory;

    /**
     * Init
     *
     * @param EavSetupFactory $eavSetupFactory
     */
}
```

```
public function __construct(EavSetupFactory
    $eavSetupFactory)
{
    $this->eavSetupFactory = $eavSetupFactory;
}

/**
 * {@inheritDoc}
 * @SuppressWarnings(PHPMD.ExcessiveMethodLength)
 */
public function upgrade(ModuleDataSetupInterface $setup,
    ModuleContextInterface $context)
{
    if (version_compare($context->getVersion(), '0.8.4',
        '<')) {
        /** @var EavSetup $eavSetup */
        $eavSetup = $this->eavSetupFactory->create(['setup'
            => $setup]);

        $fieldList = [
            'price',
            'special_price',
            'special_from_date',
            'special_to_date',
            'minimal_price',
            'cost',
            'tier_price',
            'weight',
        ];
    }

    // make these attributes applicable to demo product
    foreach ($fieldList as $field) {
        $applyTo = explode(
            ',',
            $eavSetup->getAttribute(Product::ENTITY, $field,
                'apply_to')
        );
        if (!in_array('demo', $applyTo)) {
            $applyTo[] = 'demo';
            $eavSetup->updateAttribute(
                Product::ENTITY,
                $field,
                'apply_to',
            );
        }
    }
}
```

```
        implode(' ', $applyTo)
    );
}
}
}
}
```

5. Specify the available product types that can be used to create an order:

etc/sales.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi: noNamespaceSchemaLocation="urn:magento:module:Magento_Sales:etc/ sales.xsd">
<order>
<available_product_type name="demo"/>
</order>
</config>
```

6. Optionally, it is possible to specify a custom renderer to create the `Invoice` and `Creditmemo` PDF documents:

etc/pdf.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/
  XMLSchema-instance" xsi: noNamespaceSchemaLocation=
  "urn:magento:module:Magento_Sales:etc/pdf_file.xsd">
<renderers>
<page type="invoice">
<renderer product_type="demo">Magento\Sales\Model\
  Order\Pdf\Items\Invoice\DefaultInvoice</renderer>
</page>
<page type="creditmemo">
<renderer product_type="demo">Magento\Sales\Model\
  Order\Pdf\Items\Creditmemo\DefaultCreditmemo
</renderer>
</page>
</renderers>
</config>
```

- As we added a data upgrade script, it is necessary to increment `setup_version` in the module configuration. In this case, the version has been updated from 0.8.3 to 0.8.4. This is used in the upgrade script to execute only if the installed version is lower than the new version.

`etc/module.xml`

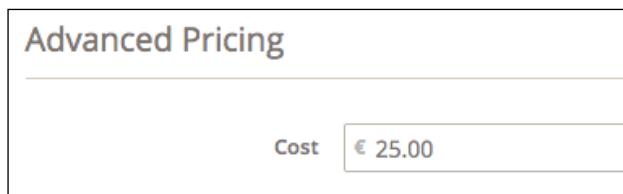
```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">
    <module name="Genmato_Sample" setup_version="0.8.4">
        <sequence>
            <module name="Magento_Store"/>
        </sequence>
    </module>
</config>
```

- After this, run the `upgrade` command to update the attributes specified in the `upgrade` command. The cache is flushed at the same time to read the updated configuration.

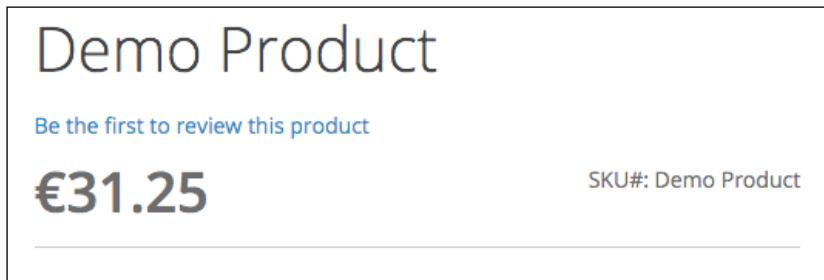
How it works...

The new defined product type is now available in the backend to create a new product. The attributes used are similar to a simple product, and you can add your own fields if necessary. The `priceModel` instance specified will calculate the product price on rendering on the frontend; in this case, the cost attribute is used and multiplied by 1.25.

While creating the product, it is possible to set the **Cost** attribute in the **Advanced Pricing** tab. In this example, we used a product cost of **€25.00**:



When the product is saved and requested on the frontend, the price will be calculated based on the preceding values and result in a final price of **€31.25**:



In the configuration of the new product type (in `product_types.xml`), it is also possible to specify the following items:

- ▶ `indexerModel`: This is to specify a custom indexer for your product type
- ▶ `stockIndexerModel`: This is to specify a custom indexer to manage the stock of your product type

Working with service layers/contracts

A service layer/contract is a fixed interface to get and store data without knowing the underlying layer. It is possible to swap the way the data is stored without changing the service layer.

A service layer consists of three interface types:

- ▶ **Data interface**: A data interface is a read-only presentation of a record, and therefore, this type of interface only has getters to represent a data record.
- ▶ **Repository interface**: A repository interface gives access to read and write (and delete) data. Every repository interface has the following methods:
 - `getList`: This returns a list of records based on the (optionally) provided search parameters
 - `get`: This loads the data from the database and returns a data interface for the specified ID
 - `save`: This saves the record specified in the data interface
 - `delete`: This deletes the record specified in the data interface
 - `deleteById`: This deletes the record specified by the ID
- ▶ **Management interface**: In a management interface, it is possible to specify special management functions that are not related to the repository.

Using a service layer also makes it easy to extend your module to access the web API; you only need to add a declaration in the appropriate XML to configure the link from the API command to the right interface.

How to do it...

In this recipe, we will add the option to read, create, or delete a record through a service layer contract:

1. Create a repository interface where the available commands are declared:

```
Api/DemoRepositoryInterface.php

<?php
namespace Genmato\Sample\Api;

interface DemoRepositoryInterface
{
    /**
     * Save demo list item.
     *
     * @api
     * @param \Genmato\Sample\Api\Data\DemoInterface $demo
     * @return \Magento\Customer\Api\Data\GroupInterface
     * @throws \Magento\Framework\Exception\InputException If
     *          there is a problem with the input
     * @throws \Magento\Framework\Exception\
     *          NoSuchEntityException If a group ID is sent but the
     *          group does not exist
     * @throws \Magento\Framework\Exception\State\
     *          InvalidTransitionException
     * If saving customer group with customer group code that
     *          is used by an existing customer group
     * @throws \Magento\Framework\Exception\LocalizedException
     */
    public function save(
        \Genmato\Sample\Api\Data\DemoInterface $demo);

    /**
     * Get demo list item by ID.
     *
     * @api
     * @param int $id
     * @return \Genmato\Sample\Api\Data\DemoInterface
     * @throws \Magento\Framework\Exception\
     *          NoSuchEntityException If $groupId is not found
    
```

```
* @throws \Magento\Framework\Exception\LocalizedException
*/
public function getById($id);

/**
 * Retrieve demo list items.
 *
 * The list of demo items can be filtered
 *
 * @api
 * @param \Magento\Framework\Api\SearchCriteriaInterface
 * $searchCriteria
 * @return \Genmato\Sample\Api\Data\
 * DemoSearchResultsInterface
 * @throws \Magento\Framework\Exception\LocalizedException
 */
public function getList(\Magento\Framework\Api\
    SearchCriteriaInterface $searchCriteria);

/**
 * Delete demo list item.
 *
 * @api
 * @param \Genmato\Sample\Api\Data\DemoInterface $demo
 * @return bool true on success
 * @throws \Magento\Framework\Exception\StateException If
 * customer group cannot be deleted
 * @throws \Magento\Framework\Exception\LocalizedException
 */
public function delete(
    \Genmato\Sample\Api\Data\DemoInterface $demo);

/**
 * Delete demolist by ID.
 *
 * @api
 * @param int $id
 * @return bool true on success
 * @throws \Magento\Framework\Exception\
 * NoSuchEntityException
 * @throws \Magento\Framework\Exception\StateException If
 * customer group cannot be deleted
 * @throws \Magento\Framework\Exception\LocalizedException
 */
public function deleteById($id);
}
```

2. Create the repository resource model. Here, the defined functions from the interface have the actual code:

```
Model/ResourceModel/DemoRepository.php

<?php
namespace Genmato\Sample\Model\ResourceModel;

use Genmato\Sample\Model\DemoRegistry;
use Genmato\Sample\Model\DemoFactory;
use Genmato\Sample\Api\Data\DemoInterface;
use Genmato\Sample\Api\Data\DemoInterfaceFactory;
use Genmato\Sample\Api\Data\DemoExtensionInterface;
use Genmato\Sample\Api\Data\
    DemoSearchResultsInterfaceFactory;
use Genmato\Sample\Model\Demo;
use Genmato\Sample\Model\ResourceModel\Demo as
    DemoResource;
use Genmato\Sample\Model\ResourceModel\Demo\Collection;
use Genmato\Sample\Api\DemoRepositoryInterface;

use Magento\Framework\Exception\CouldNotDeleteException;
use Magento\Framework\Exception\CouldNotSaveException;
use Magento\Framework\Exception\NoSuchEntityException;
use Magento\Framework\Api\SearchCriteriaInterface;
use Magento\Framework\Reflection\DataObjectProcessor;
use Magento\Framework\Api\ExtensionAttribute\
    JoinProcessorInterface;

class DemoRepository implements DemoRepositoryInterface
{

    /** @var DemoRegistry */
    private $demoRegistry;

    /** @var DemoFactory */
    private $demoFactory;

    /** @var DemoInterfaceFactory */
    private $demoDataFactory;

    /** @var Demo */
    private $demoResourceModel;

    /**
     * @var DataObjectProcessor
     */
}
```

```
private $dataObjectProcessor;

/**
 * @var DemoSearchResultsInterfaceFactory
 */
protected $searchResultsFactory;

/**
 * @var JoinProcessorInterface
 */
protected $extensionAttributesJoinProcessor;

/**
 * @param DemoRegistry $demoRegistry
 * @param DemoFactory $demoFactory
 * @param DemoInterfaceFactory $demoDataFactory
 * @param DemoResource $demoResourceModel
 * @param DataObjectProcessor $dataObjectProcessor
 * @param DemoSearchResultsInterfaceFactory
 * $searchResultsFactory
 * @param JoinProcessorInterface
 * $extensionAttributesJoinProcessor
 */
public function __construct(
    DemoRegistry $demoRegistry,
    DemoFactory $demoFactory,
    DemoInterfaceFactory $demoDataFactory,
    DemoResource $demoResourceModel,
    DataObjectProcessor $dataObjectProcessor,
    DemoSearchResultsInterfaceFactory
        $searchResultsFactory,
    JoinProcessorInterface
        $extensionAttributesJoinProcessor
) {
    $this->demoRegistry = $demoRegistry;
    $this->demoFactory = $demoFactory;
    $this->demoDataFactory = $demoDataFactory;
    $this->demoResourceModel = $demoResourceModel;
    $this->dataObjectProcessor = $dataObjectProcessor;
    $this->searchResultsFactory = $searchResultsFactory;
    $this->extensionAttributesJoinProcessor =
        $extensionAttributesJoinProcessor;
}
/**
 * {@inheritDoc}
 */
public function save(DemoInterface $demo)
```

```
{  
    /** @var Demo $demoModel */  
    $demoModel = $this->demoFactory->create();  
  
    if ($demo->getId()) {  
        $demoModel->load($demo->getId());  
    }  
    $demoModel  
        ->setTitle($demo->getTitle())  
        ->setIsVisible($demo->getIsVisible())  
        ->setIsActive($demo->getIsActive());  
  
    try {  
        $demoModel->save();  
    } catch (\Exception $exception) {  
        throw new CouldNotSaveException(__($exception->  
            getMessage()));  
    }  
    return $demoModel->getData();  
}  
  
/**  
 * {@inheritDoc}  
 */  
public function getById($id)  
{  
    $demoModel = $this->demoRegistry->retrieve($id);  
    $demoDataObject = $this->demoDataFactory->create()  
        ->setId($demoModel->getId())  
        ->setTitle($demoModel->getTitle())  
        ->setCreationTime($demoModel->getCreationTime())  
        ->setUpdateTime($demoModel->getUpdateTime())  
        ->setIsVisible($demoModel->getIsVisible())  
        ->setIsActive($demoModel->getIsActive());  
    return $demoDataObject;  
}  
  
/**  
 * {@inheritDoc}  
 */  
public function getList(SearchCriteriaInterface  
    $searchCriteria)  
{  
    $searchResults = $this->searchResultsFactory->create();  
    $searchResults->setSearchCriteria($searchCriteria);  
  
    /** @var Collection $collection */
```

```
$collection = $this->demoFactory->create() ->
    getCollection();
foreach ($searchCriteria->getFilterGroups() as
    $filterGroup) {
    foreach ($filterGroup->getFilters() as $filter) {
        $condition = $filter->getConditionType() ?: 'eq';
        $collection->addFieldToFilter($filter->getField(),
            [$condition => $filter->getValue()]);
    }
}
$searchResults->setTotalCount($collection->getSize());
$sortOrders = $searchCriteria->getSortOrders();
if ($sortOrders) {
    /** @var SortOrder $sortOrder */
    foreach ($sortOrders as $sortOrder) {
        $collection->addOrder(
            $sortOrder->getField(),
            ($sortOrder->getDirection() ==
                SortOrder::SORT_ASC) ? 'ASC' : 'DESC'
        );
    }
}
$collection->setCurPage($searchCriteria->
    getCurrentPage());
$collection->setPageSize($searchCriteria->
    getPageSize());
/** @var DemoInterface[] $demos */
$demos = [];
/** @var Demo $demo */
foreach ($collection as $demo) {
    /** @var DemoInterface $demoDataObject */
    $demoDataObject = $this->demoDataFactory->create()
        ->setId($demo->getId())
        ->setTitle($demo->getTitle())
        ->setCreationTime($demo->getCreationTime())
        ->setUpdateTime($demo->getUpdateTime())
        ->setIsVisible($demo->getIsVisible())
        ->setIsActive($demo->getIsActive());

    $demos[] = $demoDataObject;
}
$searchResults->setTotalCount($collection->getSize());
return $searchResults->setItems($demos);
}

/**
 * Delete demo list item.
```

```

*
* @param DemoInterface $demo
* @return bool true on success
* @throws \Magento\Framework\Exception\StateException If
*         customer group cannot be deleted
* @throws \Magento\Framework\Exception\LocalizedException
*/
public function delete(DemoInterface $demo)
{
    return $this->deleteById($demo->getId());
}

/**
* Delete demo list item by ID.
*
* @param int $id
* @return bool true on success
* @throws \Magento\Framework\Exception\
*         NoSuchEntityException
* @throws \Magento\Framework\Exception\StateException If
*         customer group cannot be deleted
* @throws \Magento\Framework\Exception\LocalizedException
*/
public function deleteById($id)
{
    $demoModel = $this->demoRegistry->retrieve($id);

    if ($id <= 0) {
        throw new \Magento\Framework\Exception\
            StateException(__('Cannot delete demo item.'));
    }

    $demoModel->delete();
    $this->demoRegistry->remove($id);
    return true;
}
}

```

3. Create the registry class; this stores the loaded records:

Model/DemoRegistry.php

```

<?php
/**
* Sample
*

```

```
* @package Genmato_Sample
* @author Vladimir Kerkhoff <support@genmato.com>
* @created 2015-12-23
* @copyright Copyright (c) 2015 Genmato BV,
    https://genmato.com.
*/
namespace Genmato\Sample\Model;

use Genmato\Sample\Api\Data\DemoInterface;
use Magento\Framework\Exception\NoSuchEntityException;

class DemoRegistry
{
    /**
     * @var array
     */
    protected $registry = [];

    /**
     * @var DemoFactory
     */
    protected $demoFactory;

    /**
     * @param DemoFactory $demoFactory
     */
    public function __construct(DemoFactory $demoFactory)
    {
        $this->demoFactory = $demoFactory;
    }

    /**
     * Get instance of the Demo Model identified by an id
     *
     * @param int $demoId
     * @return Demo
     * @throws NoSuchEntityException
     */
    public function retrieve($demoId)
    {
        if (isset($this->registry[$demoId])) {
            return $this->registry[$demoId];
        }
        $demo = $this->demoFactory->create();
```

```

$demo->load($demoId);
if ($demo->getId() === null || $demo->getId() != $demoId)
{
    throw NoSuchEntityException::singleField(
        DemoInterface::ID, $demoId);
}
$this->registry[$demoId] = $demo;
return $demo;
}

/**
 * Remove an instance of the Demo Model from the registry
 *
 * @param int $demoId
 * @return void
 */
public function remove($demoId)
{
    unset($this->registry[$demoId]);
}
}

```

4. Create a data interface; here, the available attributes (getters and setters) are defined:

Api/Data/DemoInterface.php

```

<?php
namespace Genmato\Sample\Api\Data;

use Genmato\Sample\Api\Data\DemoExtensionInterface;
use Magento\Framework\Api\ExtensibleDataInterface;

interface DemoInterface extends ExtensibleDataInterface
{

    const ID = 'id';
    const TITLE = 'title';
    const CREATION_TIME = 'creation_time';
    const UPDATE_TIME = 'update_time';
    const IS_ACTIVE = 'is_active';
    const IS_VISIBLE = 'is_visible';

    /**
     * Get id
     *

```

```
* @api
* @return int|null
*/
public function getId();

/**
 * Set id
 *
 * @api
 * @param int $id
 * @return $this
 */
public function setId($id);

/**
 * Get Title
 *
 * @api
 * @return string
 */
public function getTitle();

/**
 * Set Title
 *
 * @api
 * @param string $title
 * @return $this
 */
public function setTitle($title);

/**
 * Get Is Active
 *
 * @api
 * @return bool
 */
public function getIsActive();

/**
 * Set Is Active
 *
 * @api
 * @param bool $isActive
```

```
* @return $this
*/
public function setIsActive($isActive);

/**
 * Get Is Visible
 *
 * @api
 * @return bool
 */
public function getIsVisible();

/**
 * Set Is Active
 *
 * @api
 * @param bool $isVisible
 * @return $this
 */
public function setIsVisible($isVisible);

/**
 * Get creation time
 *
 * @api
 * @return string
 */
public function getCreationTime();

/**
 * Set creation time
 *
 * @api
 * @param string $creationTime
 * @return $this
 */
public function setCreationTime($creationTime);

/**
 * Get update time
 *
 * @api
 * @return string
 */

```

```
public function getUpdateTime();

/**
 * Set update time
 *
 * @api
 * @param string $updateTime
 * @return $this
 */
public function setUpdateTime($updateTime);

/**
 * Retrieve existing extension attributes object or create
 * a new one.
 *
 * @api
 * @return DemoExtensionInterface|null
 */
public function getExtensionAttributes();

/**
 * Set an extension attributes object.
 *
 * @api
 * @param DemoExtensionInterface $extensionAttributes
 * @return $this
 */
public function setExtensionAttributes(
    DemoExtensionInterface $extensionAttributes);
}
```

5. Create the data mapping class:

Model/Data/Demo.php

```
<?php
namespace Genmato\Sample\Model\Data;

use Magento\Framework\Api\AbstractExtensibleObject;
use Genmato\Sample\Api\Data\DemoInterface;
use Genmato\Sample\Api\Data\DemoExtensionInterface;

class Demo extends AbstractExtensibleObject implements
    DemoInterface
{
    /**

```

```
* Get id
*
* @return int|null
*/
public function getId()
{
    return $this->_get(self::ID);
}

/**
* Set id
*
* @param int $id
* @return $this
*/
public function setId($id)
{
    return $this->setData(self::ID, $id);
}

/**
* Get code
*
* @return string
*/
public function getTitle()
{
    return $this->_get(self::TITLE);
}

/**
* Set code
*
* @param string $title
* @return $this
*/
public function setTitle($title)
{
    return $this->setData(self::TITLE, $title);
}

/**
* Get Is Active
*
```

```
* @return bool
*/
public function getIsActive()
{
    return $this->_get(self::IS_ACTIVE);
}

/**
 * Set Is Active
 *
 * @param bool $isActive
 * @return $this
 */
public function setIsActive($isActive)
{
    return $this->setData(self::IS_ACTIVE, $isActive);
}

/**
 * Get Is Visible
 *
 * @return bool
 */
public function getIsVisible()
{
    return $this->_get(self::IS_VISIBLE);
}

/**
 * Set Is Active
 *
 * @param bool $isVisible
 * @return $this
 */
public function setIsVisible($isVisible)
{
    return $this->setData(self::IS_VISIBLE, $isVisible);
}

/**
 * Get creation time
 *
 * @return string
 */
```

```
public function getCreationTime()
{
    return $this->_get(self::CREATION_TIME);
}

/**
 * Set creation time
 *
 * @param string $creationTime
 * @return $this
 */
public function setCreationTime($creationTime)
{
    return $this->setData(self::CREATION_TIME,
        $creationTime);
}

/**
 * Get update time
 *
 * @return string
 */
public function getUpdateTime()
{
    return $this->_get(self::UPDATE_TIME);
}

/**
 * Set update time
 *
 * @param string $updateTime
 * @return $this
 */
public function setUpdateTime($updateTime)
{
    return $this->setData(self::UPDATE_TIME, $updateTime);
}

/**
 * {@inheritDoc}
 *
 * @return DemoExtensionInterface|null
 */
public function getExtensionAttributes()
```

```
{  
    return $this->_getExtensionAttributes();  
}  
  
/**  
 * {@inheritDoc}  
 *  
 * @param DemoExtensionInterface $extensionAttributes  
 * @return $this  
 */  
public function setExtensionAttributes(  
    DemoExtensionInterface $extensionAttributes)  
{  
    return $this->_setExtensionAttributes(  
        $extensionAttributes);  
}  
}
```

6. Create the search results interface; this is used in the getList command:

```
Api/Data/DemoSearchResultsInterface.php  
  
<?php  
namespace Genmato\Sample\Api\Data;  
  
use Genmato\Sample\Api\Data\DemoInterface;  
use Magento\Framework\Api\SearchResultsInterface;  
  
interface DemoSearchResultsInterface extends  
    SearchResultsInterface  
{  
    /**  
     * Get demo item list.  
     *  
     * @api  
     * @return DemoInterface[]  
     */  
    public function getItems();  
  
    /**  
     * Set demo item list.  
     *  
     * @api  
     * @param DemoInterface[] $items
```

```

    * @return $this
    */
    public function setItems(array $items);

}

```

7. Bind the interfaces through `di.xml` by adding the following lines:

`etc/di.xml`

```

<preference for="Genmato\Sample\Api\
    DemoRepositoryInterface"
    type="Genmato\Sample\Model\ResourceModel\
        DemoRepository" />

<preference for="Genmato\Sample\Api\Data\DemoInterface"
    type="Genmato\Sample\Model\Data\Demo" />
<preference for="Genmato\Sample\Api\Data\
    DemoSearchResultsInterface"
    type="Magento\Framework\Api\SearchResults" />

```

8. Add the Test controller; here, we test the working of the service layer: (This is optional.)

`Controller/Index/Test.php`

```

<?php
namespace Genmato\Sample\Controller\Index;

use Magento\Framework\App\Action\Action;
use Magento\Framework\App\Action\Context;
use Magento\Framework\View\Result\PageFactory;
use Magento\Framework\Api\SearchCriteriaBuilder;
use Genmato\Sample\Api\DemoRepositoryInterface;
use Genmato\Sample\Model\Data\DemoFactory;
use Genmato\Sample\Api\Data\DemoInterface;

class Test extends Action
{
    /**
     * @var PageFactory
     */
    private $resultPageFactory;

    /**
     * @var DemoRepositoryInterface */

```

```
private $demoRepository;

/** @var DemoFactory */
private $demo;

/**
 * @var SearchCriteriaBuilder
 */
private $searchCriteriaBuilder;
/**
 * @param Context $context
 * @param PageFactory $resultPageFactory
 * @param DemoRepositoryInterface $demoRepository
 * @param SearchCriteriaBuilder $searchCriteriaBuilder
 * @param DemoFactory $demoFactory
 */
public function __construct(
    Context $context,
    PageFactory $resultPageFactory,
    DemoRepositoryInterface $demoRepository,
    SearchCriteriaBuilder $searchCriteriaBuilder,
    DemoFactory $demoFactory
)
{
    $this->demoRepository = $demoRepository;
    $this->searchCriteriaBuilder = $searchCriteriaBuilder;
    $this->demo = $demoFactory;
    parent::__construct($context);
}

/**
 * Renders Sample
 */
public function execute()
{
    echo '<pre>';

    // Create new record through service layer/contract

    /** @var DemoInterface $demoRecord */
    $demoRecord = $this->demo->create();
    $demoRecord->setIsActive(1)
        ->setIsVisible(1)
```

```

->setTitle('Test through Service Layer');

$demo = $this->demoRepository->save($demoRecord);
print_r($demo);

// Get list of available records
$searchCriteria = $this->searchCriteriaBuilder->
    create();
$searchResult = $this->demoRepository->
    getList($searchCriteria);
foreach ($searchResult->getItems() as $item) {
    echo $item->getId().' => '.$item->getTitle(). '<br>';
}
}
}

```

9. Refresh the cache and generated data:

bin/magento setup:upgrade

10. Access the result of the test URL:

<http://example.com/sample/index/test/>

How it works...

The service layer/contract defines the methods and data format through these interfaces.

DemoRepositoryInterface

This interface describes the available methods, input, and output that is expected. The actual logic for the methods is done by the Model\ResourceModel\ DemoRepository class. In di.xml, there is a preference created that will use DemoRepository instead of the Interface class:

```
<preference for="Genmato\Sample\Api\DemoRepositoryInterface"
    type="Genmato\Sample\Model\ResourceModel\DemoRepository" />
```

DemoInterface

In this interface, the available getters and setters for the object are described. Just like RepositoryInterface, the actual processing of the data is mapped through di.xml to the Data\Demo class:

```
<preference for="Genmato\Sample\Api\Data\DemoInterface"
    type="Genmato\Sample\Model\Data\Demo" />
```

See also

In the next recipe, we will see how the `getById`, `deleteById`, `list`, and `save` methods can be used in your own code.

Creating a Magento CLI command option

With Magento 2, there is a **command-line interface (CLI)** available to run several tasks. The `bin/magento` command replaces the separate shell scripts that were used in Magento 1. This command is based on the **Symfony Console** component and looks just like `n98-magerun` that is available for Magento 1. Just like the rest of Magento 2, it's possible to extend the CLI tool with your own commands.

Getting ready

Adding commands to the CLI script requires some knowledge of the Symfony Console component. This recipe also uses the service layer created in the previous recipe.

How to do it...

In this recipe, we will add four options to the `bin/magento` CLI command with the following steps:

1. Create the `AddCommand` class; this is used to create a new record through the CLI:

```
Console/Command/AddCommand.php

<?php
namespace Genmato\Sample\Console\Command;

use Genmato\Sample\Api\DemoRepositoryInterface;
use Genmato\Sample\Model\Data\DemoFactory;
use Genmato\Sample\Api\Data\DemoInterface;

use Symfony\Component\Console\Input\InputOption;
use Symfony\Component\Console\Input\InputArgument;
use Symfony\Component\Console\Command\Command;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;
use Symfony\Component\Console\Question\ConfirmationQuestion;

class AddCommand extends Command
```

```
{

    /** @var DemoRepositoryInterface */
    private $demoRepository;

    /** @var DemoFactory */
    private $demoFactory;

    /**
     * AddCommand constructor.
     * @param DemoRepositoryInterface $demoRepository
     * @param DemoFactory $demoFactory
     * @param null $name
     */
    public function __construct(
        DemoRepositoryInterface $demoRepository,
        DemoFactory $demoFactory,
        $name = null)
    {
        parent::__construct($name);

        $this->demoRepository = $demoRepository;
        $this->demoFactory = $demoFactory;
    }

    /**
     * {@inheritDoc}
     */
    protected function configure()
    {
        $this->setName('demo:add')
            ->setDescription('Add demo record')
            ->addArgument('title', InputArgument::OPTIONAL,
                'Title')
            ->addOption('active', null, InputOption::VALUE_NONE,
                'Active')
            ->addOption('visible', null, InputOption::VALUE_NONE,
                'Visible')
        ;
    }

    /**
     * {@inheritDoc}
     */
}
```

```

protected function execute(InputInterface $input,
    OutputInterface $output)
{
    $title = $input->getArgument('title');
    $active = $input->getOption('active')? 1:0;
    $visible = $input->getOption('visible')? 1:0;
    if (!$title) {
        $dialog = $this->getHelper('dialog');

        $title = $dialog->ask($output, '<question>Enter the
            Title:</question> ',false);
        $active = $dialog->ask($output, '<question>Should
            record be active: [Y/n]</question> ','y');
        $active = (strtolower($active) == 'y') ? 1:0;
        $visible = $dialog->ask($output, '<question>Should
            record be visible: [Y/n]</question> ','y');
        $visible = (strtolower($visible) == 'y') ? 1:0;
    }

    /** @var DemoInterface $demoRecord */
    $demoRecord = $this->demoFactory->create();
    $demoRecord->setIsActive($active)
        ->setIsVisible($visible)
        ->setTitle($title);

    try {
        $demo = $this->demoRepository->save($demoRecord);
        $output->writeln('New record created (id='.$demo-
            getId().')');
    }catch (\Exception $ex) {
        $output->writeln('<error>' . $ex->
            getMessage() . '</error>');
    }
}
}

```

2. Create the delete option class; this is used to delete a record through the CLI:

Console/Command/DeleteCommand.php

```

<?php
namespace Genmato\Sample\Console\Command;

use Genmato\Sample\Api\DemoRepositoryInterface;

use Symfony\Component\Console\Input\InputOption;

```

```
use Symfony\Component\Console\Command\Command;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;
use Symfony\Component\Console\Question\ConfirmationQuestion;

class DeleteCommand extends Command
{

    /** @var DemoRepositoryInterface */
    private $demoRepository;

    public function __construct(
        DemoRepositoryInterface $demoRepository,
        $name = null)
    {
        parent::__construct($name);

        $this->demoRepository = $demoRepository;
    }

    /**
     * {@inheritDoc}
     */
    protected function configure()
    {
        $this->setName('demo:delete')
            ->setDescription('Delete demo record')
            ->addOption(
                'id',
                null,
                InputOption::VALUE_REQUIRED,
                'Demo record ID to delete'
            )
            ->addOption(
                'force',
                null,
                InputOption::VALUE_NONE,
                'Force delete without confirmation'
            );
    }

    /**
     * {@inheritDoc}
     */
}
```

```
/*
protected function execute(InputInterface $input,
    OutputInterface $output)
{
    $helper = $this->getHelper('question');

    $id = $input->getOption('id');

    try {
        if (!$input->getOption('force')) {
            $data = $this->demoRepository->getById($id);
            $output->writeln('Id : ' . $data->getId());
            $output->writeln('Title : ' . $data->
                getTitle());
            $question = new ConfirmationQuestion('Are you sure
                you want to delete this record? ', false);

            if (!$helper->ask($input, $output, $question)) {
                return;
            }
        }

        $data = $this->demoRepository->deleteById($id);
        if ($data) {
            $output->writeln('<info>Record deleted!</info>');
        } else {
            $output->writeln('<error>Unable to delete
                record!</error>');
        }
    } catch (\Exception $ex) {
        $output->writeln('<error>' . $ex->
            getMessage() . '</error>');
    }
}
```

3. Create the get option class; this is used to list a single record through the CLI:

Console/Command/GetCommand.php

```
<?php

namespace Genmato\Sample\Console\Command;

use Genmato\Sample\Api\DemoRepositoryInterface;

use Symfony\Component\Console\Command\Command;
```

```
use Symfony\Component\Console\Input\InputArgument;
use Symfony\Component\Console\Input\InputOption;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;

class GetCommand extends Command
{

    /** @var DemoRepositoryInterface */
    private $demoRepository;

    public function __construct(
        DemoRepositoryInterface $demoRepository,
        $name = null)
    {
        parent::__construct($name);

        $this->demoRepository = $demoRepository;
    }

    /**
     * {@inheritDoc}
     */
    protected function configure()
    {
        $this->setName('demo:get')
            ->setDescription('Get demo records')
            ->addOption(
                'id',
                null,
                InputOption::VALUE_REQUIRED,
                'Demo record ID to display'
            );
    }

    /**
     * {@inheritDoc}
     */
    protected function execute(InputInterface $input,
        OutputInterface $output)
    {
        $id = $input->getOption('id');

        try {

```

```
$data = $this->demoRepository->getById($id);

$table = $this->getHelper('table');
$table
    ->setHeaders(array(__('ID'), __('Title'),
        __('Created'), __('Updated'), __('Visible'),
        __('Active')));
    ->setRows([
        $data->getId(),
        $data->getTitle(),
        $data->getCreationTime(),
        $data->getUpdateTime(),
        $data->getIsVisible() ? __('Yes') : __('No'),
        $data->getIsActive() ? __('Yes') : __('No')
    ]);
$table->render($output);
} catch (\Exception $ex) {
    $output->writeln('<error>' . $ex->
        getMessage() . '</error>');
}
}
}
```

4. Create the list option class; this is used to list the available records through the CLI:

Console/Command>ListCommand.php

```
<?php
namespace Genmato\Sample\Console\Command;

use Magento\Framework\Api\SearchCriteriaBuilder;
use Genmato\Sample\Api\DemoRepositoryInterface;

use Symfony\Component\Console\Command\Command;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;

class ListCommand extends Command
{

    /** @var DemoRepositoryInterface */
    private $demoRepository;

    /**
     * @var SearchCriteriaBuilder
     */
}
```

```
private $searchCriteriaBuilder;

public function __construct(
    DemoRepositoryInterface $demoRepository,
    SearchCriteriaBuilder $searchCriteriaBuilder,
    $name = null)
{
    parent::__construct($name);

    $this->demoRepository = $demoRepository;
    $this->searchCriteriaBuilder = $searchCriteriaBuilder;
}

/**
 * {@inheritDoc}
 */
protected function configure()
{
    $this->setName('demo:list')->setDescription('List demo
        records');
}

/**
 * {@inheritDoc}
 */
protected function execute(InputInterface $input,
    OutputInterface $output)
{
    // Get list of available records
    $searchCriteria = $this->searchCriteriaBuilder->
        create();
    $searchResult = $this->demoRepository->
        getList($searchCriteria);
    $rows = [];
    foreach ($searchResult->getItems() as $item) {
        $rows[] = [$item->getId(), $item->getTitle()];
    }

    $table = $this->getHelper('table');
    $table
        ->setHeaders(array(__('ID'), __('Title')))
        ->setRows($rows)
    ;
    $table->render($output);
}
```

5. Register the commands so that they are available for the CLI by adding the following lines to the `di.xml` configuration file:

```
etc/di.xml

<type name="Magento\Framework\Console\CommandList">
    <arguments>
        <argument name="commands" xsi:type="array">
            <item name="demoadd" xsi:type="object">
                Genmato\Sample\Console\Command\AddCommand</item>
            <item name="demolist" xsi:type="object">
                Genmato\Sample\Console\Command\ListCommand</item>
            <item name="demoget" xsi:type="object">
                Genmato\Sample\Console\Command\GetCommand</item>
            <item name="demodelete" xsi:type="object">
                Genmato\Sample\Console\Command\DeleteCommand</item>
        </argument>
    </arguments>
</type>
```

6. Refresh the cache and generated data:

```
bin/magento setup:upgrade
```

7. Check whether the added commands are available by running the following command:

```
bin/magento
```

How it works...

Registering new commands works by registering new items through `di.xml` with the `Magento\Framework\Console\CommandList` class. In the XML file, every item that we want to add is listed with a unique name and the class that is used for this command.

In the class listed, there are two methods that are used:

- ▶ **configure:** In the `configure` method, the command is added with the following parameters:
 - `setName`: This is the option used for the command
 - `setDescription`: This is a short description of the command, which is shown in the command listing
 - `setArgument` (optional): This sets arguments necessary for the command
 - `setOption` (optional): This sets options necessary for the command
- ▶ **execute:** This is the actual method that is executed; here, the logic that you want to perform is located

Index

A

access control list (ACL) resource 243

advanced pricing

managing 176-179

Alternative PHP Cache (APC) 80

Apache

installing 2-5

URL 2

version 2, URL 4

attribute sets

managing 137-141

B

backend cache

Redis, configuring 80-88

backend data grid

creating 246-252

uiComponent configuration 253

backend form

adding 262, 263

creating 262

creating, to add/edit data 256-262

data, loading 264

data, saving 264

URL 256

BitBucket

URL 221

Business to Business (B2B) 127, 168

Business to Consumer (B2C) 168

C

catalog grid

products, managing 150-154

CloudFlare

about 80

Magento 2, configuring with 99-107

URL 100

command line

used, for installing Magento 2

sample data 53-56

used, for managing Magento 2 backup 65-67

used, for managing Magento 2 cache 59-64

used, for managing Magento 2 indexes 56-59

command-line interface (CLI command option)

creating 308-316

Community Edition (CE) 71

Composer 21, 43

connect portal

URL 54

controller

creating, to display data 235-241

CSS changes

Grunt, using for 198-200

CSS/JS

adding, to pages 196, 197

CSS files 198

external files 197

file in header, removing 198

JavaScript files 198

curl 71

currency rates

configuring 174, 175

customer groups

managing 168-170

D

data

displaying, web route and controller creation for 235-241

database models

working with 222-224

data interface 288

Data Migration Tool 71

DemolInterface 307

DemoRepositoryInterface 307

dependency injection (DI)

about 250

used, for passing classes to own class 268-271

Development Test Acceptance Production (DTAP) 68

DigitalOcean

URL 128, 134, 137

Docker

Magento 2, managing 38-41

URL 40

dynamic serving theme

using, client browser based 209-212

E

Enterprise Edition (EE)

about 141

version 71

Entity Attribute Value (EAV) 57

environmental variables

URL 56

Exchangeable Image File (EXIF) data 107

extension

basics, initializing 218-220

creation, ways 217, 218

packages, installing from 221, 222

F

Full Page Cache (FPC)

about 80

Varnish, configuring as 93-99

functions

modifying, plugins used 271-274

G

GitHub Data Migration Tool

URL 77

GitHub Gist

URL 92

GitHub token

URL 54

Google PageSpeed ranking 79

graphical user interface (GUI)

used, for installing Magento 2
sample data 47-52

Grunt

LiveReload plugin 202

using, for CSS language 198-202

H

HHVM (HipHop Virtual Machine)

installing 13-17

HTTP version 2 (HTTP/2)

about 2

Magento 2, configuring with 111-118

URL 117

Hypernode

Magento 2, installing 34-38

URL 34-38

Hypernode knowledge base

URL 37

Hypertext Transfer Protocol (HTTP) 2

I

ImageMagick library

URL 111

interception 271, 272

inventories

configuring 170-174

J

just-in-time (JIT) compiler 13

L

layered navigation 126

layout files 188

layout XML

<container> 192

changing, for Magento 2 module 190, 191

move command 193

referenceContainer/referenceBlock 193

remove command 193

template files, overriding 195

update command 193, 194
used, for adding static blocks
 to pages 202-204

Linux, Apache, MySQL, PHP (LAMP) 1

Linux, NGINX, MySQL, PHP (LEMP) 1

LiveReload plugin
 about 202
 URL 200

Luma 51

M

Magento
 setup, URL 38

Magento 1 database
 about 73
 transferring, to Magento 2 71-77

Magento 2
 about 126
 backup managing, via command line 64-67
 cache managing, via command line 59-64
 configuring, HTTP/2 used 111-117
 configuring, with CloudFlare 99-107
 Docker container, URL 41
 indexes managing, via command line 56-59
 indexes 56
 installing 21-33
 installing, on Hypernode 33-38
 managing, on Docker 38-41
 optimized images, configuring 107-111
 performance testing, configuring 118-123
 theme 182
 URL 23, 49

Magento 2 sample data
 installing, graphical user interface (GUI)
 used 47-52
 installing, via command line 53-56

Magento 2 set mode (MAGE_MODE)
 managing 68-71

management interface 288

Manufacturer's Suggested Retail Price (MSRP) 176

Memcached
 configuring, for session caching 89-93
 viewer 93

minimum advertised price (MAP) 176

mod_proxy module 8

mod_rewrite module 22

module
 layout XML, changing 190, 191

MySQL
 installing 18-21
 root user 20

N

Nexcess
 URL 93

NGINX
 installing 5, 6
 URL 5, 6

Node.js package manager (npm) 199

O

optimized images
 in Magento 2, configuring 107-111

P

page_cache 85

pages
 CSS/JS, adding 196, 197
 static blocks, adding to pages through layout
 XML 202-204

PageSpeed
 URL 110

performance testing
 configuring 118-123

PHP-FastCGI Process Manager (PHP-FPM)
 installing 7-13

PHP Next Generation (PHP NG) 7

PHP Redis
 URL 82

PHPRedMin
 URL 86

plugins
 used, for modifying functions 271-274

product inventory management (PIM) system 170

products
 bundle 142
 configurable 142
 creating 141-150
 downloadable 142

gift card 142
grouped 142
managing, in catalog grid 150-154
simple 142
virtual 142

product-type

creating 282-288

proxy_fcgi module 8

R

Rapido image

URL 108

Redis

configuring, for backend cache 80

repository interface 288

Root Catalog 127-133

S

sample data

URL 49

search engine optimization (SEO) 126, 171

segmentation fault 66

service layers/contracts

data interface 288

DemolInterface 307

DemoRepositoryInterface 307

management interface 288

repository interface 288

working with 288-307

session caching

Memcached, configuring 89-93

setup scripts

used, for creating tables 224-233

shipping and tax rules

creating 156-167

Software as a Service (SaaS) platform 2

software binaries 108

SSH key

URL 36

static blocks

adding, to pages through

layout XML 202-204

adding, to pages through widgets 205-207

adding, to single page 204, 205

subcategories

creating 134-136

Swiss army knife tool 43

Symfony Console component 308

system configuration fields

creating 241

new field, creating 244-246

new group, creating 244

new section, creating 243

new tab, creating 242

T

tables

creating, setup scripts used 224-233

data installation 233, 234

template files 189

theme

about 181

layout files 188

new theme, creating 182-188

template files 189

UI library 189

variants, adding 188

theme-specific translations

creating 213-215

translation file, generating 215

Transmission Control Protocol (TCP) 2

U

UberTheme

URL 77

Ubuntu

URL 7

uiComponent configuration

data source 253

mass actions, for grid 253-255

URL 250, 256

UI library 189

V

Varnish

configuring, as Full Page Cache 93-99

Varnish Configuration Language (VCL) file 93

virtual private server (VPS) 1

W

Web Application Firewall (WAF) 100

web route

creating, to display data 235-241

widgets

Catalog Category Link 208

Catalog New Products List 208

Catalog Product Link 208

Catalog Product List 208

CMS Page Link 208

CMS Static Block 208

Orders and Returns 208

Recently Compared Products 208

Recently Viewed Products 208

used, for adding static blocks

to pages 205-207

X

XML module configuration file

creating 275-281

XML Schema Definition (XSD) schema 218