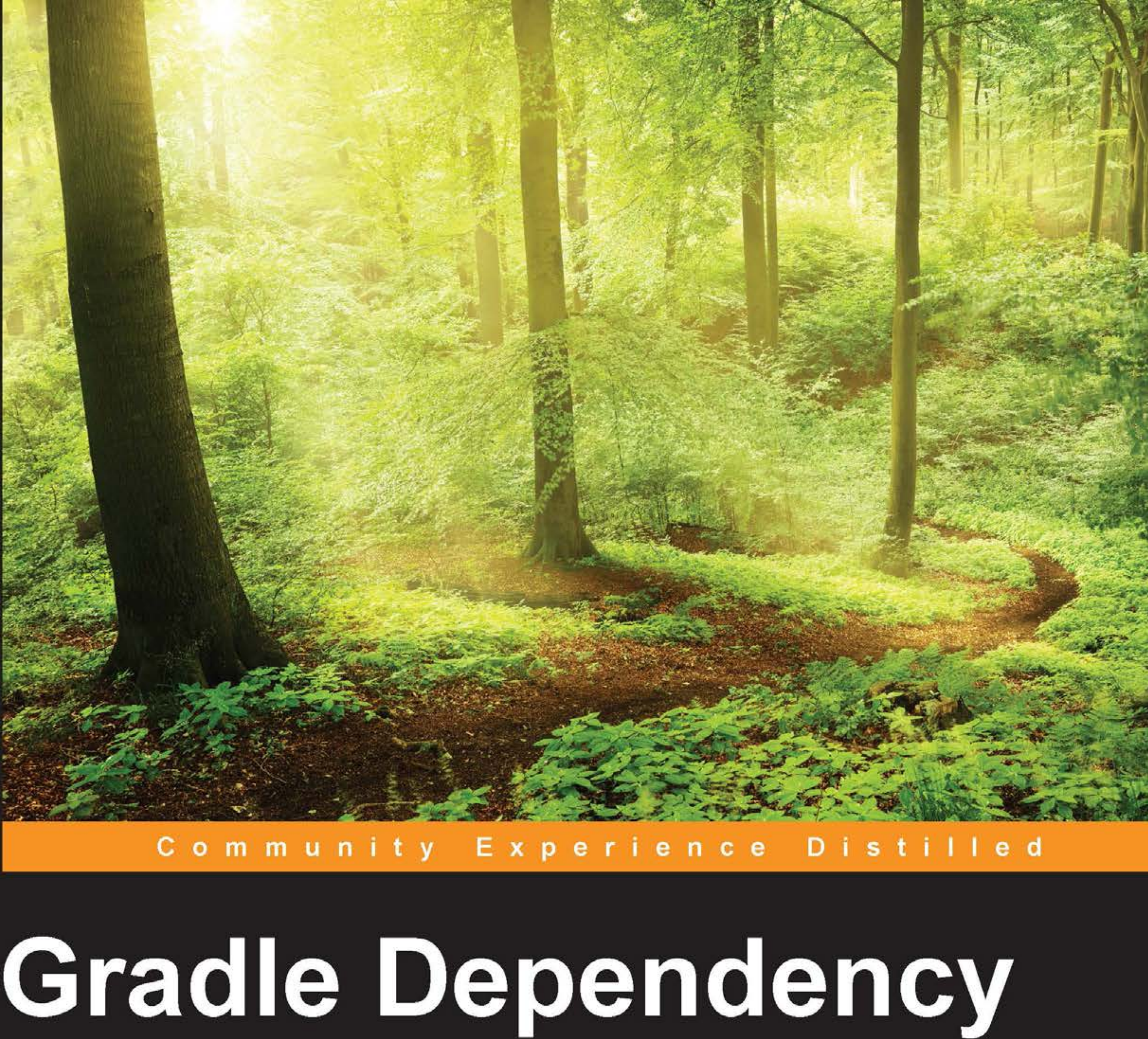

Community Experience Distilled

Gradle Dependency Management

Learn how to use Gradle's powerful dependency management through extensive code samples, and discover how to define, customize, and deploy dependencies

Hubert Klein Ikkink

[PACKT] open source*
PUBLISHING community experience distilled

Gradle Dependency Management

Learn how to use Gradle's powerful dependency management through extensive code samples, and discover how to define, customize, and deploy dependencies

Hubert Klein Ikkink

[PACKT] PUBLISHING open source community experience distilled

BIRMINGHAM - MUMBAI

Gradle Dependency Management

First published: June 2015

Production reference: 1120615

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

ISBN 978-1-78439-278-9

www.packtpub.com

Credits

Author
Hubert Klein Ikkink

Reviewers
Tony Dieppa
Izzet Mustafaiev
Konstantin Zgirovskiy

Commissioning Editor
Pramila Balan

Acquisition Editor
Sonali Vernekar

Content Development Editor
Athira Laji

Technical Editor
Siddhesh Ghadi

Copy Editor
Sarang Chari

Project Coordinator
Harshal Ved

Proofreader
Safis Editing

Indexer
Monica Mehta

Production Coordinator
Arvindkumar Gupta

Cover Work
Arvindkumar Gupta

About the Author

Hubert Klein Ikkink, born in 1973, lives in Tilburg, the Netherlands, with his beautiful wife and three gorgeous children. He is also known as mrhaki, which is simply the initials of his name prepended by "mr". He studied information systems and management at Tilburg University. After finishing his studies in 1996, he started to develop Java software. Over the years, his focus switched from applets to servlets, and from Java Enterprise Edition applications to Spring-based software and Groovy-related technologies. He likes the expressiveness of the Groovy language and how it is used in other tools, such as Gradle. He also wrote *Gradle Effective Implementation Guide*, *Packt Publishing*.

In the Netherlands, Hubert works for a company called JDriven. JDriven focuses on technologies that simplify and improve the development of enterprise applications. Employees of JDriven have years of experience with Java and related technologies and are all eager to learn about new technologies. Hubert works on projects using Grails and Java combined with Groovy and Gradle.

About the Reviewers

Izzet Mustafaiev is a family guy who likes to throw BBQ parties and travel.

Professionally, he is a software engineer working at EPAM Systems with primary skills in Java and hands-on experience in Groovy/Ruby, and is exploring FP with Erlang/Elixir. Izzet has participated in different projects as a developer and as an architect. He advocates XP, clean code, and DevOps practices when he speaks at engineering conferences.

Konstantin Zgirovskiy grew up alongside Android, in a manner of speaking. In 2008, he started programming web services and Chrome extensions for an online browser game, which was later made official. In 2011, Konstantin continued to explore Android through writing a game, which brought him victory in a local programming contest. Nowadays, he works at Looksery, Inc., where he is involved in developing an app with face-tracking and transformation technology for video chats, video selfies, and images on mobile devices.

I would like to thank my cats, friends, colleagues, and family for their support. My thanks also go to Packt Publishing for this opportunity. Additionally, I would like to thank Dasha Tsareva-Lenskaya for motivating and encouraging me whenever I got distracted.

Table of Contents

Preface

When we write code in our Java or Groovy project, we mostly have dependencies on other projects or libraries. For example, we could use the Spring framework in our project, so we are dependent on classes found in the Spring framework. We want to be able to manage such dependencies from Gradle, our build automation tool.

We will see how we can define and customize the dependencies we need.
We learn not only how to define the dependencies, but also how to work with repositories that store the dependencies. Next, we will see how to customize the way Gradle resolves dependencies.

Besides being dependent on other libraries, our project can also be a dependency for other projects. This means that we need to know how to deploy our project artifacts so that other developers can use it. We learn how to define artifacts and how to deploy them to, for example, a Maven or Ivy repository.

What this book covers

Chapter 1, *Defining Dependencies,* introduces dependency configurations as a way to organize dependencies. You will learn about the different types of dependencies in Gradle.

Chapter 2, *Working with Repositories*, covers how we can define repositories that store our dependencies. We will see not only how to set the location, but also the layout of a repository.

Chapter 3, *Resolving Dependencies,* is about how Gradle resolves our dependencies. You will learn how to customize the dependency resolution and resolve conflicts between dependencies.

Chapter 4, *Publishing Artifacts*, covers how to define artifacts for our project to be published as dependencies for others. We will see how to use configurations to define artifacts. We also use a local directory as a repository to publish the artifacts.

Chapter 5, *Publishing to a Maven Repository*, looks at how to publish our artifacts to a Maven repository. You will learn how to define a publication for a Maven-like repository, such as Artifactory or Nexus, and how to use the new and incubating publishing feature of Gradle.

Chapter 6, *Publishing to Bintray*, covers how to deploy our artifacts to Bintray. Bintray calls itself a Distribution as a Service and provides a low-level way to publish our artifacts to the world. In this chapter, we will look at how to use the Bintray Gradle plugin to publish our artifacts.

Chapter 7, *Publishing to an Ivy Repository*, is about publishing our artifacts to an Ivy repository. We will look into the different options to publish our artifacts to an Ivy repository, which is actually quite similar to publishing to a Maven repository.

What you need for this book

In order to work with Gradle and the code samples in this book, we need at least Java Development Kit (version 1.6 or higher), Gradle (samples are written with Gradle 2.3), and a good text editor.

Who this book is for

This book is for you if you are working on Java or Groovy projects and are using, or are going to use, Gradle to build your code. If your code depends on other projects or libraries, you will learn how to define and customize those dependencies. Your code can also be used by other projects, so you want to publish your project as a dependency for others whom you want to read this book.

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: "We can include other contexts through the use of the `include` directive."

A block of code is set as follows:

```
// Define new configurations for build.
configurations {

    // Define configuration vehicles.
    vehicles {
        description = 'Contains vehicle dependencies'
    }

    traffic {
        extendsFrom vehicles
        description = 'Contains traffic dependencies'
    }

}
```

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

```
$ gradle bintrayUpload
:generatePomFileForSamplePublication
:compileJava
:processResources UP-TO-DATE
:classes
:jar
:publishSamplePublicationToMavenLocal
:bintrayUpload

BUILD SUCCESSFUL

Total time: 9.125 secs
```

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: "From this screen, we click on the **New package** button."

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 from your account at `http://www.packtpub.com` for all the Packt Publishing books you have purchased. 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.

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/B03462_Coloredimages.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
Defining Dependencies

When we develop software, we need to write code. Our code consists of packages with classes, and those can be dependent on the other classes and packages in our project. This is fine for one project, but we sometimes depend on classes in other projects we didn't develop ourselves, for example, we might want to use classes from an Apache Commons library or we might be working on a project that is part of a bigger, multi-project application and we are dependent on classes in these other projects.

Most of the time, when we write software, we want to use classes outside of our project. Actually, we have a dependency on those classes. Those dependent classes are mostly stored in archive files, such as **Java Archive** (**JAR**) files. Such archive files are identified by a unique version number, so we can have a dependency on the library with a specific version.

In this chapter, you are going to learn how to define dependencies in your Gradle project. We will see how we can define the configurations of dependencies. You will learn about the different dependency types in Gradle and how to use them when you configure your build.

Declaring dependency configurations

In Gradle, we define dependency configurations to group dependencies together. A dependency configuration has a name and several properties, such as a description and is actually a special type of `FileCollection`. Configurations can extend from each other, so we can build a hierarchy of configurations in our build files. Gradle plugins can also add new configurations to our project, for example, the Java plugin adds several new configurations, such as `compile` and `testRuntime`, to our project. The `compile` configuration is then used to define the dependencies that are needed to compile our source tree. The dependency configurations are defined with a `configurations` configuration block. Inside the block, we can define new configurations for our build. All configurations are added to the project's `ConfigurationContainer` object.

In the following example build file, we define two new configurations, where the `traffic` configuration extends from the `vehicles` configuration. This means that any dependency added to the `vehicles` configuration is also available in the `traffic` configuration. We can also assign a `description` property to our configuration to provide some more information about the configuration for documentation purposes. The following code shows this:

```
// Define new configurations for build.
configurations {

  // Define configuration vehicles.
  vehicles {
    description = 'Contains vehicle dependencies'
  }

  traffic {
    extendsFrom vehicles
    description = 'Contains traffic dependencies'
  }

}
```

To see which configurations are available in a project, we can execute the `dependencies` task. This task is available for each Gradle project. The task outputs all the configurations and dependencies of a project. Let's run this task for our current project and check the output:

```
$ gradle -q dependencies

------------------------------------------------------------
Root project
------------------------------------------------------------

traffic - Contains traffic dependencies
No dependencies

vehicles - Contains vehicle dependencies
No dependencies
```

Note that we can see our two configurations, `traffic` and `vehicles`, in the output. We have not defined any dependencies to these configurations, as shown in the output.