Gradle in Action

BENJAMIN MUSCHKO

MANNING
SHELTER ISLAND
contents

foreword xiii
preface xv
acknowledgments xvii
about this book xix
about the cover illustration xxiii

PART 1 INTRODUCING GRADLE

1 Introduction to project automation 3

1.1 Life without project automation 4
1.2 Benefits of project automation 5

Prevents manual intervention 5 • Creates repeatable builds 5
Makes builds portable 6

1.3 Types of project automation 6

On-demand builds 6 • Triggered builds 7
Scheduled builds 7

1.4 Build tools 8

What's a build tool? 9 • Anatomy of a build tool 10

1.5 Java build tools 12

Apache Ant 12 • Apache Maven 16 • Requirements for
a next-generation build tool 19

1.6 Summary 20
CONTENTS

2 Next-generation builds with Gradle 22

2.1 Why Gradle? Why now? 23
   Evolution of Java build tools 24
   Why you should choose Gradle 26

2.2 Gradle’s compelling feature set 28
   Expressive build language and deep API 29
   Gradle is Groovy 30 • Flexible conventions 31
   Robust and powerful dependency management 32
   Scalable builds 33 • Effortless extendibility 34
   Integration with other build tools 34 • Community-driven
   and company-backed 35 • Icing on the cake:
   additional features 35

2.3 The bigger picture: continuous delivery 36
   Automating your project from build to deployment 36

2.4 Installing Gradle 38

2.5 Getting started with Gradle 40

2.6 Using the Command line 42
   Listing available tasks of a project 42 • Task execution 43
   Command-line options 45 • Gradle daemon 46

2.7 Summary 47

3 Building a Gradle project by example 48

3.1 Introducing the case study 49
   The To Do application 49 • Task management use cases 49
   Examining the component interaction 50 • Building the
   application’s functionality 51

3.2 Building a Java project 54
   Using the Java plugin 54 • Customizing your project 58
   Configuring and using external dependencies 59

3.3 Web development with Gradle 61
   Adding web components 61 • Using the War
   and Jetty plugins 63

3.4 Gradle wrapper 68
   Setting up the wrapper 69 • Using the wrapper 70
   Customizing the wrapper 71

3.5 Summary 72
PART 2 MASTERING THE FUNDAMENTALS .................73

4 Build script essentials 75

4.1 Building blocks 76

Projects 76 • Tasks 78 • Properties 79

4.2 Working with tasks 80

Managing the project version 80 • Declaring task actions 81
Accessing DefaultTask properties 82 • Defining task dependencies 83 • Finalizer tasks 84 • Adding arbitrary code 85 • Understanding task configuration 85
Declaring task inputs and outputs 88 • Writing and using a custom task 90 • Gradle’s built-in task types 92
Task rules 95 • Building code in buildSrc directory 98

4.3 Hooking into the build lifecycle 99

Hooking into the task execution graph 101 • Implementing a task execution graph listener 101 • Initializing the build environment 103

4.4 Summary 104

5 Dependency management 105

5.1 A quick overview of dependency management 106

Imperfect dependency management techniques 106
Importance of automated dependency management 107
Using automated dependency management 108
Challenges of automated dependency management 109

5.2 Learning dependency management by example 110

5.3 Dependency configurations 111

Understanding the configuration API representation 111
Defining a custom configuration 112
Accessing a configuration 113

5.4 Declaring dependencies 114

Understanding the dependency API representation 114
External module dependencies 115 • File dependencies 120

5.5 Using and configuring repositories 121

Understanding the repository API representation 121
Maven repositories 121 • Ivy repositories 124
Flat directory repositories 125
5.6 Understanding the local dependency cache 126
   Analyzing the cache structure 126
   Notable caching features 127

5.7 Troubleshooting dependency problems 128
   Responding to version conflicts 128
   Enforcing a specific version 129
   Using the dependency insight report 130
   Refreshing the cache 130

5.8 Summary 131

6 Multiproject builds 133

6.1 Modularizing a project 134
   Coupling and cohesion 134
   Identifying modules 135
   Refactoring to modules 136

6.2 Assembling a multiproject build 137
   Introducing the settings file 138
   Understanding the Settings API representation 139
   Settings execution 140
   Settings file resolution 140
   Hierarchical versus flat layout 141

6.3 Configuring subprojects 142
   Understanding the Project API representation 143
   Defining specific behavior 144
   Declaring project dependencies 145
   Partial multiproject builds 147
   Declaring cross-project task dependencies 149
   Defining common behavior 151

6.4 Individual project files 152
   Creating build files per project 153
   Defining the root project’s build code 153
   Defining the subprojects’ build code 153

6.5 Customizing projects 154

6.6 Summary 155

7 Testing with Gradle 157

7.1 Automated testing 158
   Types of automated testing 158
   Test automation pyramid 159

7.2 Testing Java applications 159
   Project layout 160
   Test configurations 161
   Test tasks 161
   Automatic test detection 162

7.3 Unit testing 162
   Using JUnit 162
   Using alternative unit testing frameworks 166
   Multiple unit testing frameworks in harmony 168
9.2 Maven and Gradle  236
   Commonalities and differences  236 • Migration strategies  240
9.3 Comparing builds  243
9.4 Summary  245

PART 3 FROM BUILD TO DEPLOYMENT..................247

10 IDE support and tooling  249
10.1 Using IDE plugins to generate project files  250
   Using the Eclipse plugins  251 • Using the IDEA plugin  258
   Using the Sublime Text plugin  264
10.2 Managing Gradle projects in popular IDEs  267
   Gradle support in SpringSource STS  268 • Gradle support in
   IntelliJ IDEA  271 • Gradle support in NetBeans IDE  274
10.3 Embedding Gradle with the tooling API  277
10.4 Summary  281

11 Building polyglot projects  282
11.1 Managing JavaScript with Gradle  283
   Typical tasks when dealing with JavaScript  283
   Using JavaScript in the To Do application  284
   Dependency management for JavaScript libraries  285
   Merging and minifying JavaScript using a third-party
   Ant task  287 • JavaScript optimization as part of the
   development workflow  289 • JavaScript code analysis using
   an external Java library  290 • Using a third-party Gradle
   JavaScript plugin  292 • Executing Grunt from Gradle  293
11.2 Building polyglot, JVM-based projects  296
   Base capabilities of JVM language plugins  296
   Building Groovy projects  298 • Building Scala projects  303
11.3 Other languages  307
11.4 Summary  308

12 Code quality management and monitoring  310
12.1 Integrating code analysis into your build  311
12.2 Measuring code coverage  312
   Exploring code coverage tools  313 • Using the
   JaCoCo plugin  315 • Using the Cobertura plugin  317
CONTENTS

12.3 Performing static code analysis 319
   Exploring static code analysis tools 320 • Using the
   Checkstyle plugin 322 • Using the PMD plugin 324
   Using the FindBugs plugin 326 • Using the
   JDepend plugin 327

12.4 Integrating with Sonar 328
   Installing and running Sonar 330 • Analyzing a project
   with Sonar Runner 331 • Publishing code coverage
   metrics to Sonar 333

12.5 Summary 336

13 Continuous integration 337

13.1 Benefits of continuous integration 338

13.2 Setting up Git 340
   Creating a GitHub account 340 • Forking the
   GitHub repository 341 • Installing and configuring Git 341

13.3 Building a project with Jenkins 342
   Starting Jenkins 342 • Installing the Git and
   Gradle plugins 342 • Defining the build job 344
   Executing the build job 347 • Adding test reporting 348

13.4 Exploring cloud-based solutions 351

13.5 Modeling a build pipeline with Jenkins 352
   Challenges of building a pipeline 352 • Exploring essential
   Jenkins plugins 353 • Configuring the pipeline jobs 356

13.6 Summary 358

14 Artifact assembly and publishing 359

14.1 Building artifacts and distributions 360
   Declaring additional artifacts 361 • Creating distributions 363

14.2 Publishing artifacts to a binary repository 366
   Publishing to a Maven repository 366 • Old versus new
   publishing mechanisms 368 • Declaring a software component as
   a Maven publication 368 • Publishing a software component to
   the local Maven cache 369 • Declaring custom artifacts
   for publication 372 • Modifying the generated POM 373
   Publishing to a local Maven repository 375 • Publishing to
   a remote Maven repository 377

14.3 Publishing to a public binary repository 379
   Publishing to JFrog Bintray 380 • Publishing to
   Maven Central 383
14.4 Artifact assembly and publishing as part of the build pipeline 385

*Build binaries once 385*  *Publish once, reuse later 385*
*Picking an appropriate versioning scheme 387*
*Including build information in a deployable artifact 390*
*Publishing your To Do application WAR file 391*
*Extending the build pipeline 392*

14.5 Summary 393

15 **Infrastructure provisioning and deployment 395**

15.1 Infrastructure provisioning 396

*Infrastructure as code 396*  *Creating a virtual machine with Vagrant and Puppet 397*  *Executing Vagrant from Gradle 399*

15.2 Targeting a deployment environment 401

*Defining configuration in a Groovy script 401*  *Reading the configuration with Groovy's ConfigSlurper 403*  *Using the configuration throughout the build 404*

15.3 Automated deployments 405

*Retrieving the artifact from the binary repository 405*  *Identifying necessary deployment steps 407*  *Deployment through SSH commands 407*

15.4 Deployment tests 412

*Verifying a successful deployment with smoke tests 412*  *Verifying application functionality with acceptance tests 415*

15.5 Deployment as part of the build pipeline 416

*Automatic deployment to test environment 417*  *Deployment tests 417*  *On-demand deployment to UAT and production environment 418*

15.6 Summary 419

*appendix A  Driving the command line 421*  *appendix B  Groovy for Gradle users 429*  *index 440*