Gradle
Build automation

ivan.macalak@posam.sk

What is Gradle

- Gradle is flexible general purpose build tool
- It combines Ant with Dependency management and Maven conventions
- It provides Groovy based DSL interface for writing build scripts
- Conventions with great flexibility
- Combination of declarative and imperative build tool

Tools put together

- Flexibility
- Full control
- Chaining of targets

- Dependency management

maven

- Convention over configuration
- Multimodule projects
- Extensibility via plugins

- Groovy DSL on top of Ant

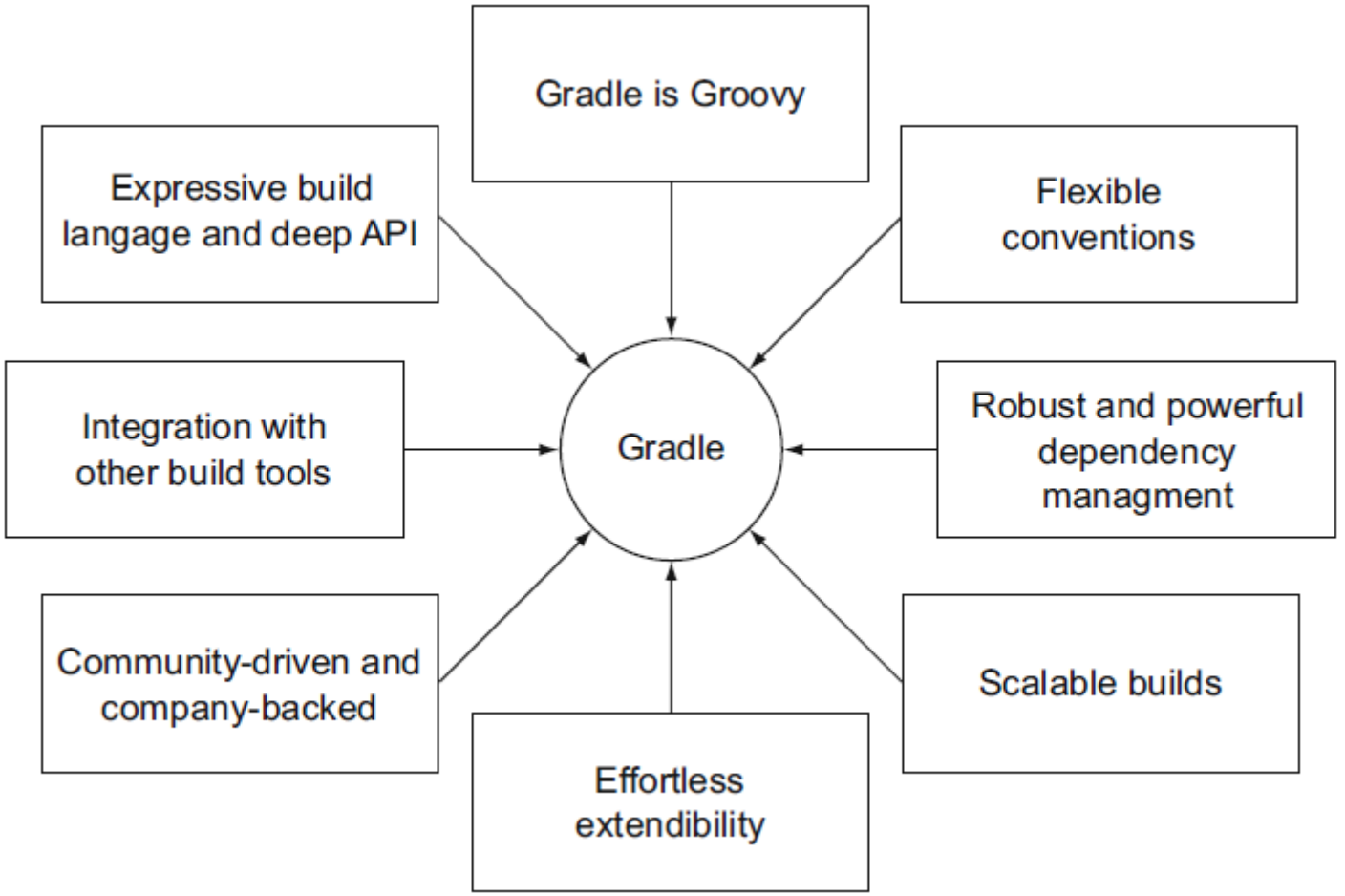
Kotlin

- Kotlin DSL

Gradle offers

- Build automation, checking, testing, publishing
- Dependency management based on Apache Ivy
- Support of multi-project builds
- Incremental builds
- Pluggable architecture
- IDE support and integration
- You can build Java, Android, C++ and more

Gradle is ...


Gradle is Groovy
Expressive build langage and deep API
Flexible conventions
Integration with other build tools
Gradle
Robust and powerful dependency managment
Community-driven and company-backed
Effortless extendibility
Scalable builds

Gradle Install or Wrapp

- Gradle requires JDK
- Groovy distribution is shipped with Gradle
- You can manually download and install Gradle
  - https://gradle.org/gradle-download
  - Unpack distribution and set `GRADLE_HOME`
  - Set JVM options to run gradle, `GRADLE_OPTS` or `JAVA_OPTS`
  - Use `gradle` command to build your project
- Or you can use Gradle Wrapper
  - Automatically downloads Gradle distribution
  - You should check it into version control
  - Use `gradlew` instead `gradle` command
  - Available on Windows and Linux

gradle/gradlew command

- Build is started with `gradle/gradlew` command
- Command accepts different options. Use `gradle -h` to list command options
- You typically provide task name and configuration properties. Gradle looks for `build.gradle` in current folder
- If there is a default task defined in build script, you just execute `gradle/gradlew` command with no additional parameters

Gradle daemon

- Background process
  - Speeds up the build
- Ideal when you build frequently
- Can be enabled via CLI, or for environment
  - `gradle -daemon`
  - `GRADLE_OPTS: -Dorg.gradle.daemon=true`
  - `org.gradle.daemon=true in the GRADLE_USER_HOME/gradle.properties`
  - In Gradle v3 the daemon is enabled by default
    - Use `-no-daemon` option to disable daemon for build

Gradle build

- Scripts - build source code is organized in one or more gradle scripts. Default script name is `build.gradle`
- Projects - at least one or more projects. The name of the projects is the build script's parent directory
- Tasks - build is collection of tasks with dependencies. You can use build-in tasks, or define your tasks. You ca exclude task from execution during build process
- Properties - comprises a build environment
- Plugins - gradle features extension. Bring new tasks and configuration into your project build. There are a lot of official, or unofficial Gradle plugins you can use in your projects

Gradle build lifecycle

- Gradle creates instance(s) of defined Project(s)

- Project objects are configured
- Project scripts are executed, without task execution
- Statements you have written outside of the task in the configuration block would be executed

- Gradle determines the subset of the tasks, created and configured during the configuration phase
- Gradle executes all tasks in defined order given in the command line

Gradle build Script

- Build script is code in Gradle
  - Set of actions that execute in some predefined order and perform certain operations
  - Represented by `org.gradle.api.Script`
- You can implement any build functionality directly in build script using Groovy
- For cleaner code organization, you can split your code into multiple source files and apply them, where you need
- Check the available plugins before, you implement required functionality

Gradle Project

- Root entity (`org.gradle.api.Project`)
- Contains one or more Tasks
- Execution of the build represents the execution of the Project object, which internally calls different tasks to perform the operations
- Build can contain more than one Project. Then we call it multiproject build

Gradle Task

- Atomic unit of execution (`org.gradle.api.Task`)
- Collection of actions and properties. It can depend on some other tasks
- Can accept input and return output
- Task defines two types of closures: `doFirst` and `doLast`. You can use these to add specific code before, or after task execution
- Gradle provides conditional Task execution

Gradle and Ant Tasks

- There are many exiting useful Ant tasks you can easily reuse in the Gradle build script

```
task hello {
  doLast {
     String greeting = 'hello from Ant'
     ant.echo(message: greeting)
  }
}
```

Incremental Builds

- Incremental build speeds up build execution
- Only not up to date task are executed
- Gradle implements mechanism to execute build task, only if task input, or output has changed

Environment Variables

- Use `GRADLE_OPTS` or `JAVA_OPTS` to specify command-line arguments to use to start the JVM. You can use these variables to set for example the JVM memory of Java process which executes the build
- You can use environment properties like this
  - `ORG_GRADLE_PROJECT_propertyName=Value`
- System properties can be defined using -D command line option. It has the same effect as -D option for java command
- It is useful to store some variables in VCS. For such scenario, use Gradle property file

Gradle Properties

- Properties are defined in gradle.properties file and are injected into project
- Properties file can be placed
  - In project directory
  - In gradle user home directory
    - `$USER_HOME/.gradle` by default
    - Can be changed by `GRADLE_USER_HOME` env variable
  - The properties file in the user's home directory has precedence over property files in the project directories
- Project properties can be defined using -P command line option
  - it has precedence over property files

Repositories

- External Project dependencies are usually stored in repositories
- Gradle supports several repository types
  - Maven
  - Apache Ivy
  - Flat directories
- You must define one or more repositories where Gradle will look for dependencies
  - Use `repositories {…}` closure

Repositories Definition Example [1]

```
repositories {
  mavenCentral()
  jcenter()
  mavenLocal()
}
```

- Maven Central repo: http://search.maven.org/
- Jcenter repo: https://jcenter.bintray.com/
- Maven Local cache: `<USER_HOME>/.m2/repository`

Repositories Definition Example [2]

```
repositories {
  maven {
    url "http://private.repository/path"
    credentials {
      username 'user'
      password 'psswd'
   }
  }
  maven {
    url "http://repo.springsource.org/release"
  }
}
```

Repositories Definition Example [3]

```
repositories {
  flatDir {
    dirs '/localfile/dir1',
         '/localfile/dir2'
  }
}
```

Dependency Configurations

- Gradle groups dependencies to different configurations
- If you build Java project, the java plugin brings some configurations `(compile, runtine, testCompile, testRuntime, ...)`
- You can define your own dependency configuration
- You define dependencies in `dependencies` closure

```
dependencies { ...}
```

Dependency Artifact

- Typical Maven repository artifact consists of
  - JAR file group (or namespace)
  - JAR filename
  - JAR file version
  - classifier (in case JAR has classifier-like-specific JDK version)
- Example for log4j artifact
  - `group: 'log4j'`
  - `name: 'log4j'`
  - `version: '1.2.16'`

Dependency Resolution [1]

- Project's dependencies can be complex
- There are usually transitive dependencies
  - You specify only first level dependency
  - Gradle handles complex dependencies for you
  - You can exclude dependency from transitive tree
- Gradle handles also version conflicts using several strategies
  - Last version strategy
  - Fail on conflict
  - Force specific version

Dependency Resolution [2]

- Gradle supports dynamic dependency
- When you want to use always latest version

```
group:'some-group',name:'some-name',version:'latest.integration'
group:'some-group',name:'some-name',version:'1.+'
```

Dependency Artifact Examples

```
dependencies {
 compile group:'log4j', name:'log4j', version: '1.2.16'
}
dependencies {
  compile 'log4j:log4j:1.2.16'
  compile files('lib/vehicles/car-2.0.jar')
}
```

Project Dependencies

- Project dependencies ate typical for multi-project builds

```
dependencies {
    compile project(':data:access')
}
```

Build Script Dependencies

- Sometimes you need to add a dependency to the Gradle build script itself as you like to use 3rd party libraries in the build script code
- You need to define repositories and dependencies

```
buildscript {
  repositories {...}
  dependencies { ...}
}
```

Dependency Cache

- The dependency cache of Gradle tries to minimize the number of remote requests and downloads
- Gradle maintains
  - cache for dependency metadata
  - cache for downloaded artifacts
- Both the metadata cache and artifact cache are stored in the directory defined by the `GRADLE_USER_HOME`
  - by default, the `.gradle/caches` directory in the user home directory
- You can use command line options to
  - skip any network requests: `--offline`
  - refresh the metadata caches: `--refresh-dependencies`

Publishing Artifacts

- A Gradle project can contain artifacts you want to publish
- You can define one or more artifacts in one project
- You can publish to defined supported repositories
  - Maven
  - Ivy
  - Local directory
- Publishing to other repositories can be supported by plugins (Artifactory, Bintray)

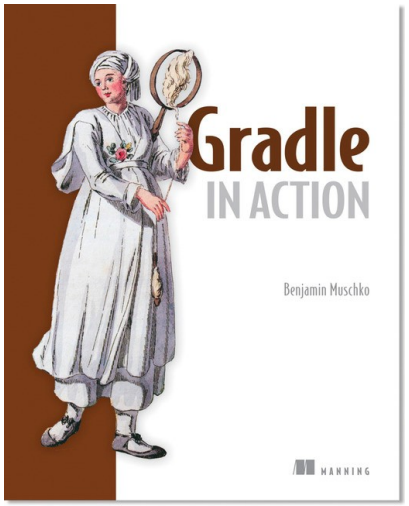
Android Plugin

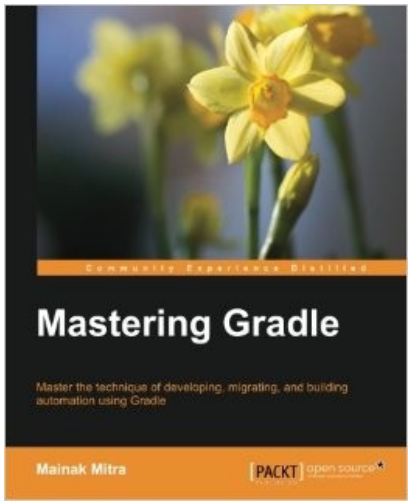
- Android releases its own Gradle plugin to support Android builds
- Specific Android plugin version requires concrete Gradle version
- Android studio uses Gradle as build tool
- Gradle build file can be manipulated by Android studio

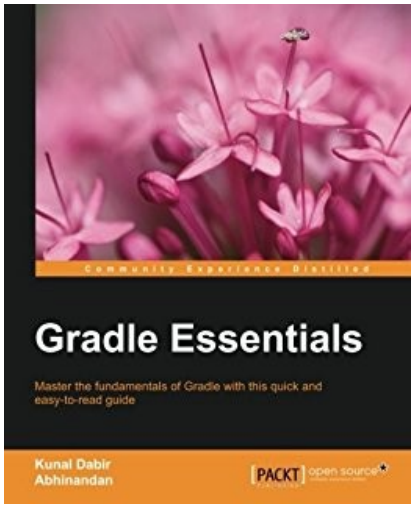
References

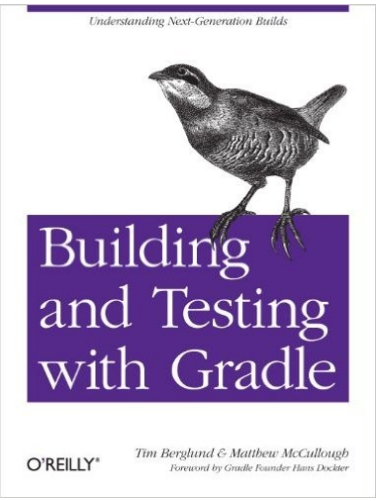
- Gradle official web pages: https://gradle.org
- https://docs.gradle.org/current/dsl/index.html
- https://plugins.gradle.org/
- http://gradle.org/maven_vs_gradle/
- http://gradle.org/case-study-continuous-delivery-netflix/
- http://gradle.org/case-study-gradle-continuous-delivery-linkedin/
- http://gradle.org/open-source-build-system-evaluation-in-the-age-of-continuous-delivery-part-1/

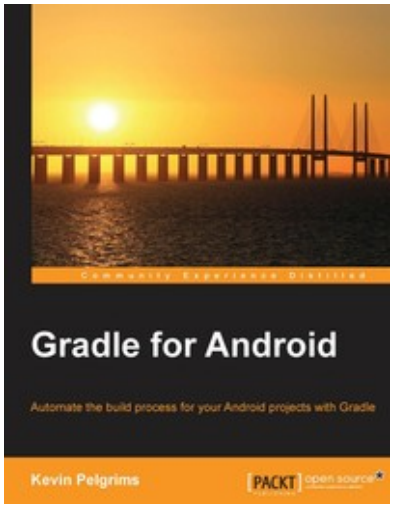
Books

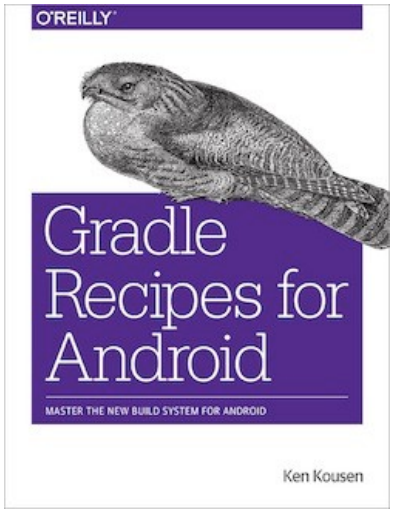

Gradle
IN ACTION
Benjamin Muschko
MANNING

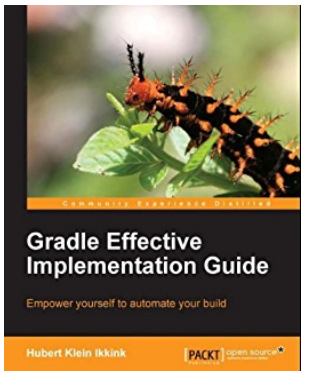

Community Experience Distilled
Mastering Gradle
Master the technique of developing, migrating, and building automation using Gradle
Mainak Mitra
PACKT open source

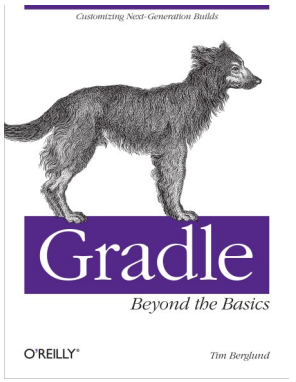

Community Experience Distilled
Gradle Essentials
Master the fundamentals of Gradle with this quick and easy-to-read guide
Kunal Dabir
Abhinandan
PACKT open source

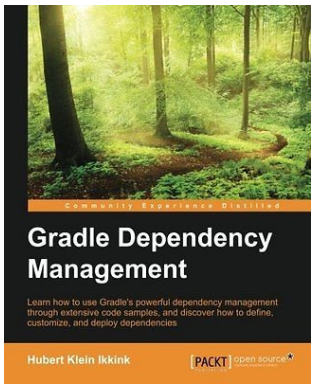

Understanding Next-Generation Builds
Building and Testing with Gradle
O'REILLY
Tim Berglund & Matthew McCullough
Foreword by Gradle Founder Hans Dockter


Community Experience Distilled
Gradle for Android
Automate the build process for your Android projects with Gradle
Kevin Pelgrims
PACKT open source


O'REILLY
Gradle Recipes for Android
MASTER THE NEW BUILD SYSTEM FOR ANDROID
Ken Kousen


Community Experience Distilled
Gradle Effective Implementation Guide
Empower yourself to automate your build
Hubert Klein Ikkink
PACKT open source


Customizing Next-Generation Builds
Gradle
Beyond the Basics
O'REILLY
Tim Berglund


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