
PUZZLE ITC
changing IT for the better

Stefan Rotman
rotman@puzzle.ch

1. Introducing Gradle
2. Groovy Basics
3. Gradle Build Scripts
4. Tasks
5. Plugins
6. Dependency Management
7. Publishing Artifacts

Agenda

$ whoami

Stefan Rotman <rotman@puzzle.ch>

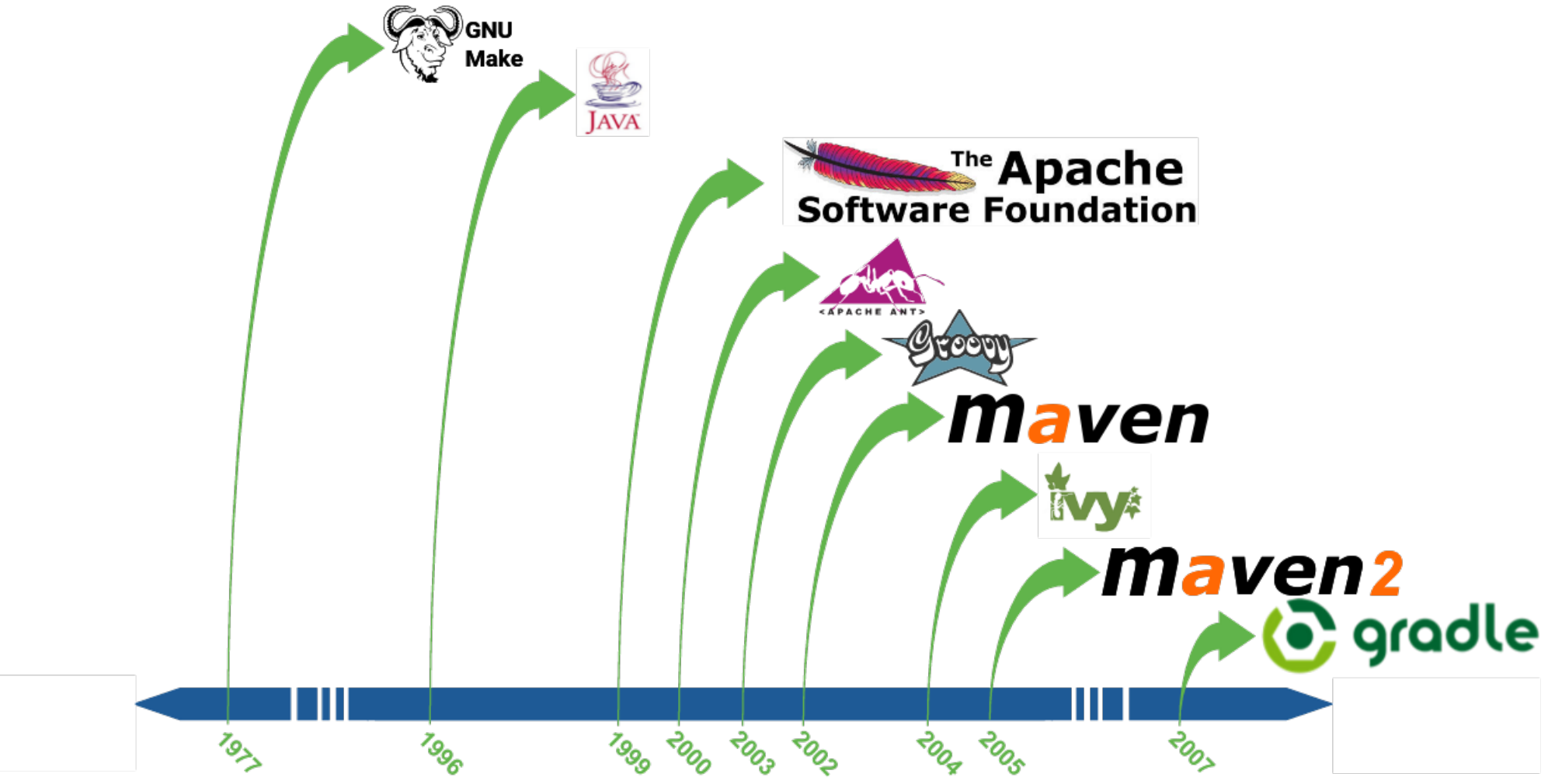
Puzzle ITC – since 2010 [Software Engineer, Software Architect]

Java Enthusiast – since 1997

Gradle Enthusiast – since 2014

1

Introducing Gradle


GNU
Make
JAVA
The Apache
Software Foundation
<APACHE ANT>
Groovy
maven
ivy
maven2
gradle
1977
1996
1999
2000
2003
2002
2004
2005
2007

Gradle is ...

« Gradle is an open source build automation system that builds upon the concepts of Apache Ant and Apache Maven and introduces a Groovy-based domain-specific langage (DSL) instead of the XML form used by Apache Maven of declaring the project configuration. »

Wikipedia

Gradle is ...

- Not just a build-tool but a build-environment
- Implemented in Java and Groovy
- On GitHub (https://github.com/gradle/gradle)
- Very well documented (https://docs.gradle.org/)
- Developed by a dedicated company (Gradleware), as well as by a strong community

Gradle provides ...

- very flexible general purpose build tool (like Ant)
- build-by-convention frameworks (like Maven)
- strong dependency management (based on Ivy)
- Great multi-project support
- Groovy build scripts (no XML!)
- Gradle Wrapper
- ...

The Gradle Wrapper

A lightweight distribution of the Gradle executeable

Ensures the proper Gradle version amongst team and buildserver

- without the need for manual installation

Should be checked into version control!

```
$ gradle wrapper --gradle-version
3.1
:wrapper

BUILD SUCCESFUL

Total time : 1 secs

$ ./gradlew -version

------------------------------------
Gradle 3.1
------------------------------------

Build time:   2016-09-19 10:53:53 UTC
```

2

Groovy Basics

Groovy Basics

- Modern scripting langage for the JVM
- Designed to be easily picked up by Java developers
- Interchangeable with Java semantics/API

Groovy Language Features

- Dynamically and optionaly static typing
- Removes a lot of « syntax noise »
- Strings
  - Single quoted – no interpolation
  - Double quoted – interpolated
  - Triple quoted – multiline, interpolation depends on single or double
- Many JDK library enhancements

Groovy Closures

- « Compareable » to Java 8 lambdas, JavaScript functions, …
  - But they are Objects, not Interfaces
- Can have parameters

```
{ name -> "Hello, $name !" }                  // Explicit parameter
{ greeting, str -> "$greeting, $name!" }      // Multiple parameters
{ "Hello, $it!" }                             // Implicit parameter
{ -> "Hello, world!" }                        // No parameters
```

Groovy Methods

- Implicit return of last statement
- Parameters *can* have a default value
- Braces are optional in method calls
- If the last method parameter is a closure, it can be placed *outside* the method call

3

Gradle Build Scripts

Gradle Build scripts

Gradle scripts are valid and fully powered Groovy scripts

Gradle scripts follow the Gradle DSL

Gradle build scripts are actually build *configuration* scripts

Gradle builts upon 2 basic concepts :

- Projects
- Tasks

build.gradle

The « build script » is in a file named build.gradle

This build script is backed by the Project object

```
1  apply plugin: 'java'
2
3  repositories {
4      mavenCentral()
5  }
6
7  dependencies {
8      testCompile 'junit:junit:4.12'
9  }
10
11 task run(type: JavaExec) {
12     classpath = sourceSets.main.runtimeClasspath
13     main = 'ch.puzzle.gradle.HelloWorld'
14 }
```

Project properties

Project properties can be set directly in the build script, or in a separate gradle.properties file.

Some useful project properties include

```
version = <project version>
group = <com.project.group>
name = <project name> // Will default to the build directory
description = <project description>
```

Extra properties

All « extra » properties must be defined through the ext namespace

```
ext.isSnapshot = version.endsWith("-SNAPSHOT")

if (isSnapshot) {
  // do something snapshotty
}
```

Proxy Settings

Proxy settings can be specified in the gradle.properties file

This file can be in 2 places: the build root dir or Gradle home dir

Proxy Settings are configured via standard JVM system properties

```
systemProp.http.proxyHost=<host>
systemProp.http.proxyPort=<port>
systemProp.http.proxyUser=<username>
systemProp.http.proxyPassword=<password>
systemProp.http.nonProxyHosts=*.nonproxyrepos.com |localhost
systemProp.https.proxyHost=<host>
...
```

Build Lifecycle

A Gradle build has three distinct phases

- Initialization : Create Project instances for the projects to be build
- Configuration : Configuration of the project objects
  build scripts for *all* initialized projects are executed
- Execution : requested – and required – tasks are executed

4

Tasks

Defining a task

Tasks are the fundamental units of build activity

At a minimum, a task needs a name

To ensure « simple » task actions are executed during *execution phase*, they should be executed inside the **doFirst** or **doLast** methods.

The **<<** operator can be used as short-hand notation for doLast

Defining a task

```
task helloWorld << {
    println 'Hello, World!'
}
```

```
task helloWorld {
    doLast {
        println 'Hello
    }
}
```

```
task helloWorld {
    doFirst { print 'Hello, ' }
    doLast { print 'World! ' }
}
```

```
task helloWorld
helloWorld << {
    print 'Hello, '
}
helloWorld << {
    print 'World!'
}
```

Task Configuration

Tasks are commonly configured with configuration Closure:

```
task copy(type: Copy) {
    from 'resources'
    into 'build'
}
```

Note that *any* statement in the configuration Closure is considered part of the task configuration!

Adding a human-readable **description** to a task helps to identify it in the task list

Task Actions

The build steps of a task are defined in task actions.

The execution phase executes the task actions :

1 defined as **doFirst** actions

2 defined as task actions by the Task type (@TaskAction)

3 Defined as **doLast** actions

Task Dependencies

Gradle tasks can define dependencies on other tasks with the **dependsOn** method

```
task y
task x (dependsOn: y)

task x {
    dependsOn y
}

x.dependsOn y
```

4

Plugins

Gradle Plugins

Plugins are used to extends the build functionality

Plugins add tasks, domain objects, conventions and more.

Gradle comes with various core plugins

https://docs.gradle.org/current/userguide/standard_plugings.html

Add a core plugin

Gradle core plugins can be enabled using the apply statement :

```
apply plugin : 'java'
```

or defined inside the **plugins** block:

```
plugins {
    id 'java'
}
```

Add 3rd party plugins

3rd pary plugins can be found at the Gradle Plugin portal

https://plugins.gradle.org

Defined in the buildfile inside the **plugins** block:

```
plugins {
    id "org.gradle.hello-world" version "0.2"
}
```

Add 3rd party plugins (legacy style)

```
buildscript {
    repositories {
        maven {
            url "https://plugins.gradle.org/m2/"
        }
    }
    dependencies {
        classpath "org.gradle:gradle-hello-world-plugin:0.2"
    }
}

apply plugin: "org.gradle.hello-world"
```

Java Plugin – SourceSet conventions

Source set conventions:

src/main/java	- Production Java source
src/main/resources	- Production resources
src/test/java	- Test Java source
src/test/resources	- Test resources
src/<sourceSet>/java	- Java source for the given source set
src/<sourceSet>/resources	- Resources for the given source set

Java Plugin – Tasks

Some of the tasks added by the Java Plugin

- compileJava and processResources tasks for *each* source set
- clean and clean<Taskname> tasks for specific cleanups
- jar and uploadArchives
- test

Java Plugin – Test

Requires test-library dependency for test compilation

Works 'out of the box' for the test source set

6

Dependency Management

Standard Repositories

Dependency repositories are defined in the **repositories** block.

Gradle offers out-of-the-box support for various standard repositories :

```
repositories {
    mavenCentral()      // http://repo1.maven.org/maven2
    jcenter()           // https://jcenter.bintray.com
    mavenLocal()        // local maven repository (USER_HOME/.m2)
}
```

Custom Repositories

It is also possible to specify custom Maven, Ivy or Flat directory repositories :

```
repositories {
    maven {
        url "https://repo.mycompany.com/m2"
    }
}
```

To add basic auth, a **credentials** block can be added to the **maven** block

Credentials should not be in the project build file, but in a personal gradle.properties!

Dependency configurations

Dependencies in Gradle are grouped by configurations.

Some of those configurations as defined by the Java Plugin are

compile	compile-time dependencies for the production source
runtime	runtime dependencies for the production classes
testCompile	compile-time dependencies for the test source
testRuntime	runtime dependencies for the test classes

External dependencies

External dependencies are defined in de **dependencies** block

```
dependencies {
    compile group: 'com.example', name: 'dependency', version: '1.0'
    compile 'com.example:dependency:1.0'
}
```

Fine-tuning dependencies

https://docs.gradle.org/current/userguide/dependency_management.html#sec:finetuning_the_dependency_resolution_process

- Forcing a particular module version
  - Resolve version conflicts
- Excluding transitive dependencies
  - Either specific or alltogether
- Dependency substitution

7

Publishing Artifacts

Artifacts

The **archives** method call in the **artifacts** block adds artifacts that should be created when calling the **assemble** task

```
task fooJar(type: Jar)

artifacts {
    //archives foo --> implicit
    archives fooJar
}
```

Thank you!