Next Generation Builds with **Gradle**

Benjamin Muschko

June 22, 2011

About Me

- Java/Groovy developer with 9 years of experience
- Gaelyk and Gradle contributor
- Author of various plugins for Grails, Gradle, Gaelyk
- GroovyMag author
- Follow me on Twitter: @bmuschko
- Fork me on GitHub: bmuschko

Gradle

What is it?

- Flexible, extensible Open Source build tool
- Core written in Java, scripts in Groovy DSL
- Dependency management & multi-project support
- Convention over Configuration
- Hosted on GitHub, first release in April 2008

Gradle

Build System History

1. Generation

maven 1

2. Generation

+

maven 2

3. Generation

Gradle

maven 3

Gradle

Best of all Worlds

Flexibility
Full Control
Chaining of Targets

Dependency Management

Convention over Configuration
Multi-module Projects
Extensibility via Mojos

Groovy DSL on top of Ant

Gradle

Gradle

Gradle Goodies

- Ease of migration
- Add custom logic using tasks and plugins
- Gradle wrapper (run Gradle without installation)
- Incremental builds (only build what changed)
- Gradle daemon (avoid startup cost)
- Rich CLI (e.g. GUI, dry-run, camel case)

Gradle Pain Points

- IDE support is not great
  - Eclipse STS has Gradle plugin
  - IntelliJ minimal support
  - NetBeans has no support
  - Eclipse, Idea plugins to the rescue
- Plugin ecosystem needs to catch up
  - no central repository
  - no plugin descriptor
  - your favorite plugin might not exist
- Gradle doesn’t support project archetypes

Gradle

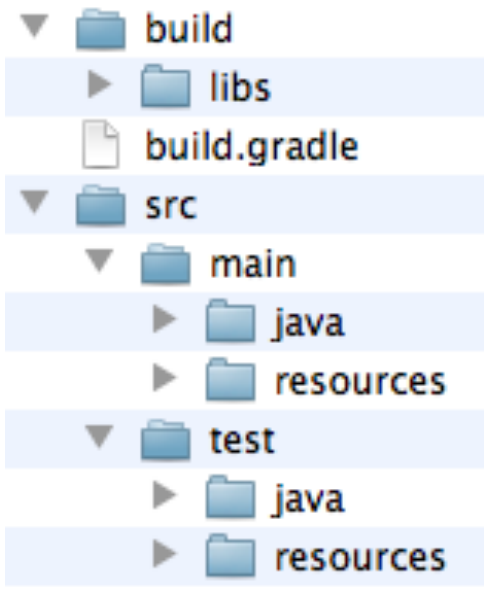
Convention over Configuration

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

Gradle

Convention over Configuration

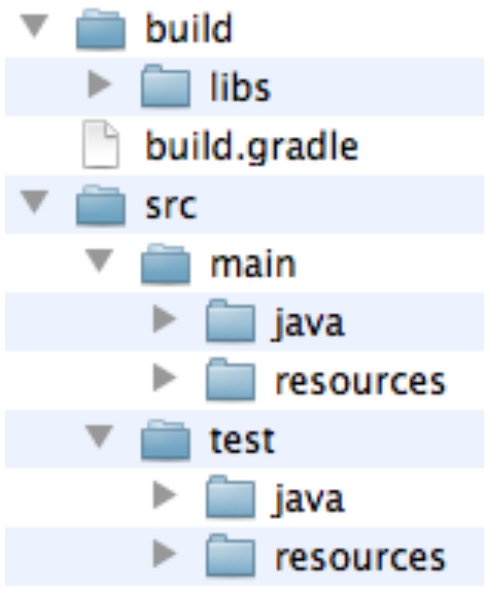
```
apply plugin: 'java'
```



Gradle

Convention over Configuration

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

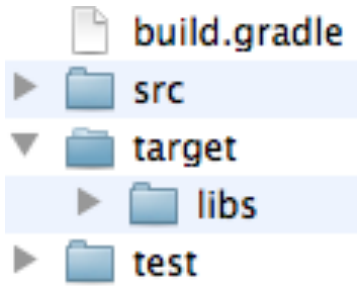



```
sourceSets {
    main {
        java {
            srcDir 'src/main/java'
        }
        resources {
            srcDir 'src/main/resources'
        }
    }
    test {
        java {
            srcDir 'src/test/java'
        }
        resources {
            srcDir 'src/test/resources'
        }
    }
}
```

Gradle

Customizing Default Layout

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




```
sourceSets {
    main {
        java {
            srcDir 'src'
        }
    }
    test {
        java {
            srcDir 'test'
        }
    }
}

buildDir = 'target'
archivesBaseName = 'groovydc'
version = 1.1
```

Gradle

Dependency Management

```
repositories {
    mavenCentral()
    mavenRepo name: 'InternalRepo' urls: 'http://repo.internal.it'
    add(new org.apache.ivy.plugins.resolver.URLResolver()) {
        name = 'Cloud Repo'
        addArtifactPattern """http://cloud.repo.com/downloads/libs/
                              [module]-[revision].[ext]"""
    }
    flatDir dirs: '/home/gradle/libs'
}

dependencies {
    compile 'log4j:log4j:1.2.15', transitive: true
    compile('org.gradle.test.excludes:api:1.0') {
        exclude module: 'shared'
    }
    testCompile group: 'junit', name: 'junit', version: '4.+'
    runtime 'taglibs:standard:1.1.2', 'javax.servlet:jstl:1.1.2'
}
```

Gradle

Java Gradle Build Example

```
apply plugin: 'java'

sourceCompatibility = 1.5
version = '1.0'

repositories {
    mavenCentral()
}

dependencies {
    compile 'commons-lang:commons-lang:2.3'
    testCompile group: 'junit', name: 'junit', version: '4.+'
}

jar {
    manifest {
        attributes 'Implementation-Title': 'GroovyDC example',
                   'Implementation-Version': version
    }
}
```

Building a Java project

```
> gradle build
:compileJava                Compiles Java sources
:processResources           Copies resources to classes dir
:classes                    Assembles the main classes
:jar                        Creates JAR artifact
:assemble                   Assembles all archives
:compileTestJava            Compiles Java test sources
:processTestResources       Copies test resources to classes dir
:testClasses                Assembles the test classes
:test                       Runs the unit tests
:check                      Runs all checks
:build                      Assembles and tests project

BUILD SUCCESSFUL

Total time: 1 secs
```

- Declare task in build script using Gradle DSL
  - written in Groovy
  - Ant tasks reusable out-of-the-box
  - hooks into specific phase of execution lifecycle
  - can apply additional task rule
  - can be chained and imported if defined in separate script
- Custom task
  - written as class that extends Gradle's `DefaultTask`
  - describes behavior, gets applied to build script
- Custom plugin
  - bundles more complex logic
  - wide range of existing plugins

Gradle

Task Example

```
defaultTasks 'clean', 'run'

task clean << {
    ant.delete(dir: 'output')
    println 'Deleted output directory'
}

task run(dependsOn: clean) << {
    println 'Default Running!'
}

task setConfig {
    description = 'Sets headless system property.'
    setHeadless()
}

def setHeadless() {
    System.setProperty('java.awt.headless', 'true')
}
```

- Gradle builds dependency graph (DAG)
- Your build script defines dependency graph
- Three distinct build phases
  - Initialization
  - Configuration
  - Execution
- Multi-module projects require `settings.gradle`

Custom Task Example

```
task depPersist(type: DependenciesTask) {
    println 'Writes dependencies to file.'
    output = file('build/dependencies.txt')
}

import org.gradle.api.DefaultTask

class DependenciesTask extends DefaultTask {
    File output

    @TaskAction
    void execute() {
        def text = new StringBuilder()
        text <<= 'gradle dependencies'.execute().text
        output << text
    }
}
```

Plugin Example

- Plugin implementation

```
package org.groovydc

import org.gradle.api.Plugin
import org.gradle.api.Project

class ExamplePlugin implements Plugin<Project> {
    @Override
    void apply(Project project) {
        // Your logic goes here
    }
}
```

- Optional manifest `META-INF/gradle-plugins/example.properties`

```
implementation-class=com.groovydc.ExamplePlugin
```

Existing Plugins

- Standard Plugins
  - Language Plugins: Java, Groovy, Scala, Antlr
  - Integration Plugins: WAR, Jetty, Maven, OSGI, Application
  - Development Plugins: Eclipse, IDEA, Code Quality, Sonar
- Third Party Plugins
  - Language Plugins: Clojuresque
  - Integration Plugins: Android, GWT, AspectJ, Tomcat, GAE
  - Development Plugins: Emma, FindBugs, CheckStyle, JSLint

  ...and many more

Gradle

Using Ant within Gradle

- build.xml

```
<project>
    <target name="run" description="Prints message">
        <echo>Hello GroovyDC!</echo>
    </target>
</project>
```

- build.gradle

```
ant.importBuild 'build.xml'

task echo << {
    ant.echo 'Super simple migration'
}
```

- Existing `build.xml` can be imported
- Ant targets get treated as Gradle tasks
- AntBuilder implicitly available in `build.gradle`
- All existing standard Ant tasks available
- `dependsOn` doesn't respect execution order

➔ **Migration is very easy, can be done gradually**

Gradle

Maven vs. Gradle

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>de.muschko</groupId>
   <artifactId>maven_gradle_comparison</artifactId>
   <packaging>jar</packaging>
   <name>GroovyDC example</name>
   <version>1.0</version>
   <build>
      <plugins>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.3.2</version>
            <configuration>
               <source>1.5</source>
            </configuration>
         </plugin>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jar-plugin</artifactId>
            <version>2.3.1</version>
            <configuration>
               <archive>
                  <manifest>
                     <addDefaultImplementationEntries>true</addDefaultImplementationEntries>
                  </manifest>
               </archive>
            </configuration>
         </plugin>
      </plugins>
   </build>
   <dependencies>
      <dependency>
         <groupId>commons-lang</groupId>
         <artifactId>commons-lang</artifactId>
         <version>2.3</version>
         <scope>compile</scope>
      </dependency>
      <dependency>
         <groupId>junit</groupId>
         <artifactId>junit</artifactId>
         <version>4.4</version>
         <scope>test</scope>
      </dependency>
   </dependencies>
</project>
```

```groovy
apply plugin: 'java'

sourceCompatibility = 1.5
version = 1.0

repositories {
    mavenCentral()
}

dependencies {
    compile 'commons-lang:commons-lang:2.3'
    testCompile 'junit:junit:4.4'
}

jar {
    manifest {
        attributes 'Implementation-Title': 'GroovyDC example',
                   'Implementation-Version': version
    }
}
```

- Existing `pom.xml` cannot be referenced/reused
- Support for multi-module projects
- Gradle provides Maven plugin
- `maven2gradle`* eases the pain

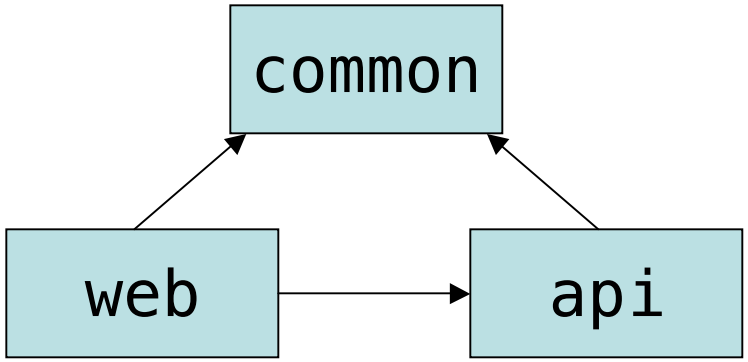
➥ **Tools available, full migration required**

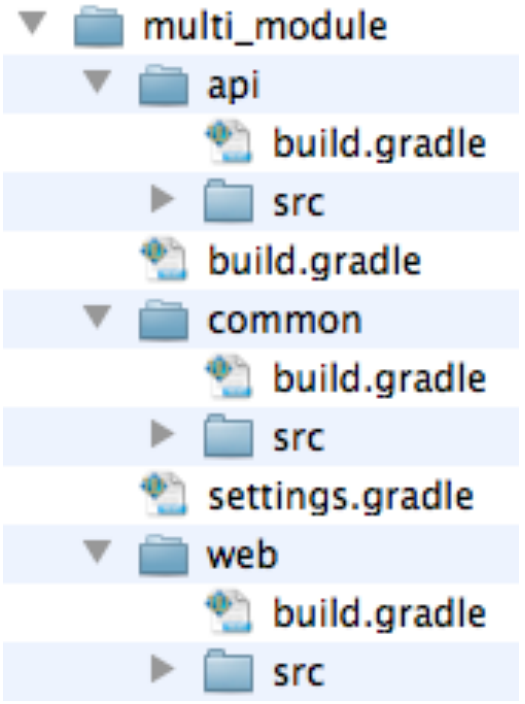
*https://github.com/jbaruch/maven2gradle

Multi-module Example

Three modules:

- common
- api
- web





Gradle

Multi-module Example

- Parent build.gradle

```
allprojects {
    apply plugin: 'java'
    version = 1.0
}

subprojects {
    sourceCompatibility = 1.6
    targetCompatibility = 1.6

    repositories {
        mavenCentral()
    }
}
```

- settings.gradle

```
include 'common', 'api', 'web'
```

Multi-module Example

- Web module build.gradle

```
project(':web') {
    apply plugin: 'war'
    apply plugin: 'jetty'

    dependencies {
        compile project(':common'),
                project(':api')

        runtime 'taglibs:standard:1.1.2',
                'javax.servlet:jstl:1.1.2'

        providedCompile 'javax.servlet:servlet-api:2.5',
                        'javax.servlet:jsp-api:2.0'
    }
}
```

- If you did it in Maven you can easily do it in Gradle!
- Layout is totally flexible
- Number of `build.gradle` files is >= 1
- `settings.gradle` defines included modules
- `allprojects` applies to project and subprojects
- `subprojects` just applies to subprojects

Links

- Gradle home
  - http://www.gradle.org/
- Gradle cookbook
  - http://gradle.codehaus.org/Cookbook
- Gradle non-standard plugins
  - http://docs.codehaus.org/display/GRADLE/Plugins
- Presentation & source code
  - http://github.com/bmuschko/presentations

```
> gradle qa
:askQuestions

BUILD SUCCESSFUL

Total time: 300 secs
```