Gradle - Build system evolved

By Bhagwat Kumar

Agenda

- What and Why?
- Key features of Gradle
- Project and tasks
- Using java/groovy/war/jetty plugins
- Manage dependency
- Working with multi-module project
- Create grails application using Gradle

What is Gradle?

- A general-purpose build automation tool. It can automate
  - building
  - testing
  - deployment
  - publishing
  - generate static websites
  - generate documentation
  - indeed anything else
- Designed to take advantage of convention over configuration.
- Combines the power and flexibility of Ant with the dependency management and conventions of Maven into a more effective way to build.

Why Gradle?

- Powerful and expressive Groovy DSL instead of XML.
- Adaptable to nonconventional project structures.
- Works well for java as well as non java projects.
- Extensible (plugins!).
- Multi project builds are hassle free.
- Powerful dependency management.
- Reduced start up cost using Gradle daemon.
- Caching of task input output.
- Introduced to Android with **Android Studio.**
- **Grails 3.0** build system will be updated to use Gradle.

Combines best features from other build tools.

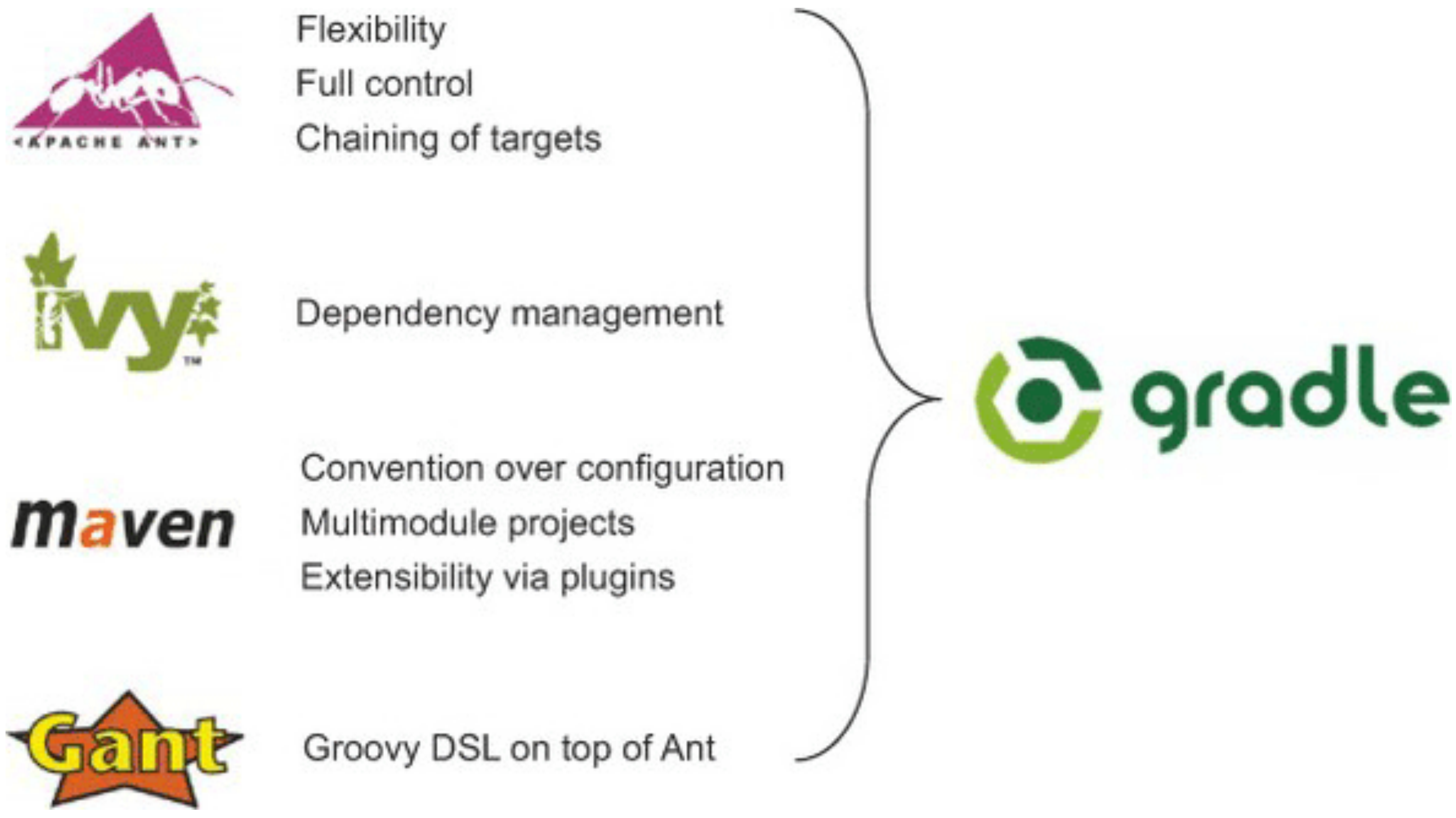

Image source : http://www.drdobbs.com/jvm/why-build-your-java-projects-with-gradle/240168608

Features

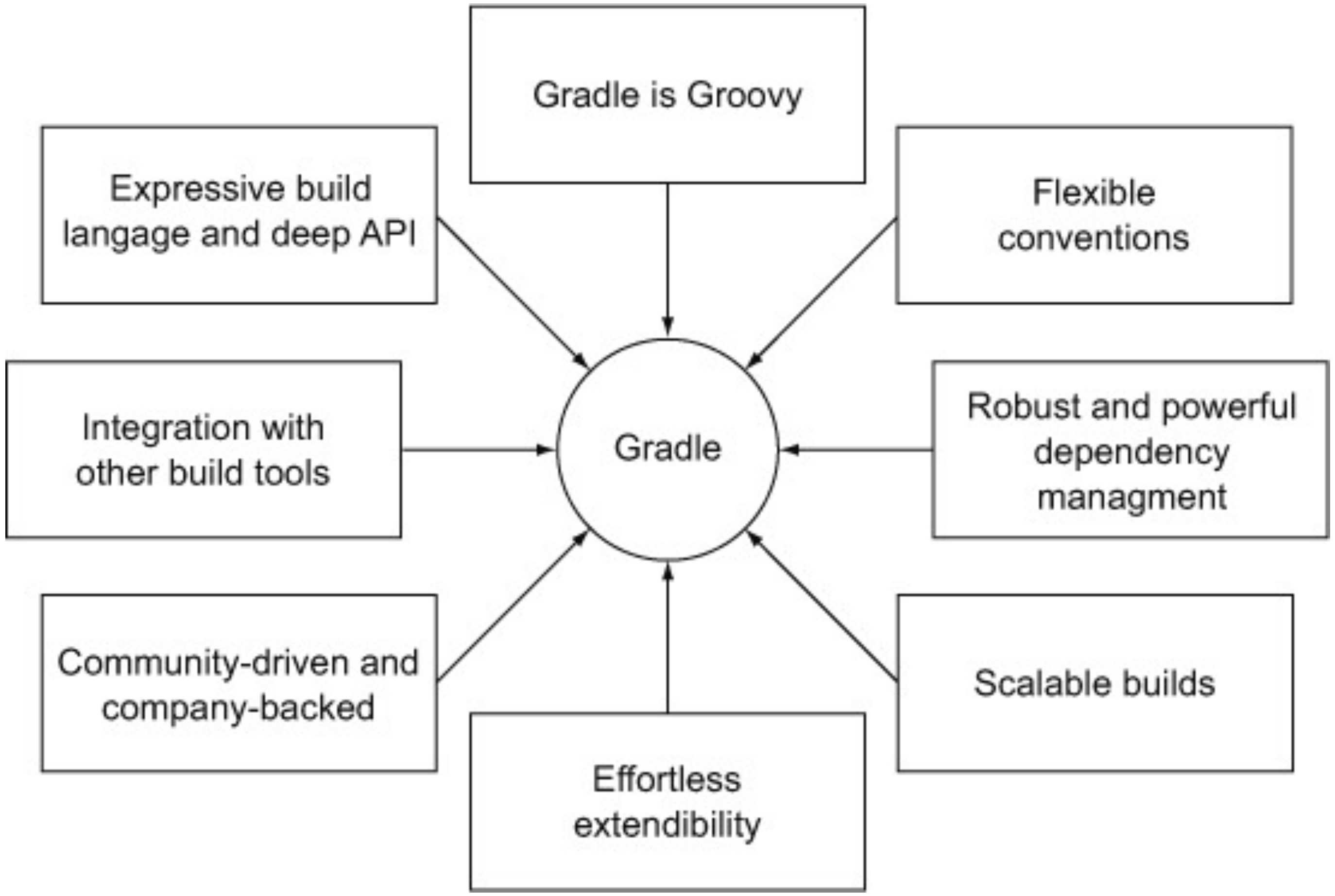
Gradle is Groovy
Expressive build langage and deep API
Flexible conventions
Integration with other build tools
Gradle
Robust and powerful dependency managment
Community-driven and company-backed
Scalable builds
Effortless extendibility

Image source : http://www.drdobbs.com/jvm/why-build-your-java-projects-with-gradle/240168608

Who are using?

Carrier turn to the experts

EADS

fcc Farm Credit Canada

software AG

GRAILS

GRIFFON

Groovy

HIBERNATE

LinkedIn

ORBITZ

spring by Pivotal

NETFLIX

THALES Training & Consultancy

BOSCH

SIEMENS

Prezi

Installation

Using gvm

```
$ gvm install gradle

$ gradle -v

$ gradle init

$ gradle tasks
```

Access gradle documentation from local file system

```
{USER_HOME}/.gvm/gradle/current/docs/userguide/userguide.pdf
```

Gradle Build Script

- build.gradle file in project root directory.
- Its actually a groovy script.
- This is where you define gradle tasks.
- Every Gradle build is made up of one or more projects.

Gradle project

- It does not necessarily represent a thing to be built.
- It might represent a
  - library jar
  - web-app
  - distribution zip assembled from the JARs produced by other projects
  - thing to be done : deploy to staging server

Gradle Tasks

- An atomic piece of work that a build performs.
  - compile classes
  - create jar
  - generate java doc
  - publish some archive to a repository
- Each project is made up of one or more tasks

build.gradle

```
task hello << {
    println "Hello !!!!"
}
task greet <<{
    println "Welocome Mr. Kumar"
}
task intro(dependsOn: hello) << {
    println "I'm Gradle"
}
hello << { println "Hello extended!!!!" }

greet.dependsOn hello, intro

// gradle tasks  :list all the available tasks
// gradle intro  :executes intro task
// gradle -q greet :bare build output
// gradle --daemon hello  :subsequent execution will be fast
```

Defines task 'hello' with groovy closure.

Defines task 'intro' which depends on 'hello' task

appends additional groovy closure to be executed to existing 'hello' task

gradle tasks are groovy objects. You can change dependency of existing tasks

Using plugins

- Gradle core intentionally provides very little for automation.
- All of the useful features are added by plugins. e.g. java compile.
- Gradle plugins
  - add new tasks (e.g. JavaCompile)
  - add domain objects (e.g. SourceSet)
  - add conventions (e.g. Java source is located at src/main/java)
  - extends core objects and objects from other plugins
- Plugin portal : http://plugins.gradle.org/

Applying plugins

Script plugins

```
//from a script on the local filesystem or at a remote location.

apply from: 'other.gradle'
```

Binary plugins

```
// applying a binary plugin by type
apply plugin: 'java'
```

Using plugins DSL

```
//version is optional
plugins {
    id "com.jfrog.bintray" version "0.4.1"
}
```

Using java plugin

- New/modified tasks
  - clean, compileJava, classes, jar, uploadArchives, build etc.
- Default project layout
  - src/main/java
  - src/main/resources
  - src/test/java
- Dependency configurations
  - compile, runtime, testCompile, archives etc.
- Convention properties
  - sourceSets, sourceCompatibility, archivesBaseName etc.

Changing project layout for java plugin

```
apply plugin: 'java'

sourceSets {
    main {
        java {
            srcDirs = ['source/main/java']
        }
    }

    test {
        java {
            srcDirs = ['source/test/java']
        }
    }
}
```

```
|-- build.gradle
|-- settings.gradle
`-- source
    `-- main
        `-- java
            `-- com
                `-- intelligrape
                    `-- gradle
                        `-- Application.java
```

```
package com.intelligrape.gradle.demo;

public class Application {
    public static void main(String[] args) {
        System.out.println("Hello Gradle!!!");
    }
}
```

Application.java

A task to execute java class

build.gradle

```
apply plugin: 'java'

task runMe(type: JavaExec, dependsOn: 'classes') {
    main = "com.intelligrape.gradle.demo.Application"
    classpath = sourceSets.main.runtimeClasspath
}
```

Execute java class using 'runMe' task

```
$ gradle -q runMe
Hello Gradle!!!
```

Or execute using java command

```
$ gradle -q build

$java -cp build/classes/main com.intelligrape.gradle.demo.Application
Hello Gradle!!!
```

Configure jar manifest

```
apply plugin: 'java'

sourceCompatibility = 1.5

//controls jar file name
version = '1.0-snapshot'
archivesBaseName="sample-java"

//self executable jar
jar {
    manifest {
        attributes("Main-Class": "com.intelligrape.manifest.Application",
                "Implementation-Version": version)
    }
}
```

```
$ gradle -q build

$java -jar build/lib/sample-java-1.0-snapshot.jar
Hello Gradle!!!
```

Dependency management

```
apply plugin: 'java'

repositories {
    //mavenLocal()
    mavenCentral()
    //maven { url "http://repo.mycompany.com/maven2"}
   //flatDir { dirs 'lib1', 'lib2'}
}

dependencies {
    compile "org.quartz-scheduler:quartz:2.1.5"
    testCompile group: 'junit', name: 'junit', version: '4.11'
    compile project(':shared')  //project dependency
    runtime files('libs/a.jar', 'libs/b.jar') //file dependency
}
```

Working with groovy plugin

- Groovy plugin extends the Java plugin to add support for Groovy projects.
- It can deal with Groovy code, mixed Groovy and Java code, and even pure Java code.
- New/modified tasks
  - compileGroovy, groovyDoc, classes etc
- Project layout
  - src/main/groovy
  - src/test/groovy
- You must add dependency for groovy
  - compile localGroovy()
  - compile 'org.codehaus.groovy:groovy-all:2.3.6'

build.gradle with groovy plugin

```
apply plugin: 'groovy'

repositories {
    mavenCentral()
}
dependencies {
//    compile localGroovy()
    compile 'org.codehaus.groovy:groovy-all:2.3.6'
}
sourceSets {
    main {
        groovy {
            srcDirs = ['source/groovy']
        }
    }
    test {
        groovy {
            srcDirs = ['source/groovy']
        }
    }
}
```

Using application plugin

```
apply plugin: 'java'

version = '1.0-snapshot'

apply plugin: 'application'

applicationName="sample-java"
mainClassName="demo.gradle.groovy.SampleGroovy"
```

```
$ gradle -q run
Hello Gradle!!!
```

Using war and jetty plugin

```
apply plugin: "war"      //gradle assemble
//apply plugin: "jetty" //gradle jettyRun

version = "1.0.0"
archivesBaseName = "multi-web"

repositories {
    mavenCentral()
}

dependencies {
    compile project(":utils")
    compile "javax.servlet:servlet-api:2.5"
}

war {
    webInf { from 'webapp/WEB-INF' }
}
```

```
|-- build.gradle
`-- src
    `-- main
        |-- java
        |   `-- com
        |       `-- intelligrape
        |           `-- web
        |               `-- HomeServlet.java
        `-- webapp
            |-- WEB-INF
            |   `-- web.xml
            `-- index.jsp
```

Working with multi-project

build.gradle

'hello' task is added to root as well as all the sub projects

Applied to only sub projects

java plugin applied to all subproj

Applied to only root project

```
allprojects {
    task hello << { task ->
        println "I'm $task.project.name"
    }
}
subprojects {
    task onlySubProjectTask << { task ->
        println "I'm $task.project.name"
    }
    apply plugin: "java"
    repositories {
        mavenCentral()
    }
}
task onlyMainProjectTask << { task ->
    println "Only Main Project Task : ${task.project.name} "
}
```

```
multi-project/
|-- build.gradle
|-- settings.gradle
|-- top
|    `-- build.gradle
`-- util
     `-- build.gradle
```

settings.gradle

```
rootProject.name = 'multi-project'

include 'util', 'top'
```

Working with multi-project cont...

Executing 'hello' task : executed for all projects

```
$ gradle hello
:hello
I'm multi-project
:top:hello
I'm top
:util:hello
I'm util
```

Executing 'onlySubProjectTask' task : executed for only sub-projects

```
$ gradle onlySubProjectTask
:top:onlySubProjectTask
I'm top
:util:onlySubProjectTask
I'm util
```

Execute hello task of 'top' sub-project only

```
$ gradle :top:hello
:top:hello
I'm top
```

Using grails-gradle plugin

```
buildscript {
  repositories { jcenter() }
  dependencies {classpath "org.grails:grails-gradle-plugin:2.0.0"}
}
version "0.1"
group "example"

apply plugin: "grails"
grails {
  grailsVersion = '2.3.6'
  springLoadedVersion '1.1.3'
  groovyVersion = '2.3.6'
}
repositories {
  //maven repo for the Grails Central repository (Core libraries and plugins)
  grails.central()
}

dependencies {
  // No container is deployed by default, so add this
  bootstrap "org.grails.plugins:tomcat:7.0.50"
  // Just an example of adding a Grails plugin
  compile 'org.grails.plugins:resources:1.2'
}
```

Using grails-gradle plugin continued

- Create build.gradle file and copy content from previous slide.
- Execute 'gradle init' which will create file structure
- To run app execute:
  - gradle gradle-run-app
- The plugin configures a Gradle Task Rule to allow execution of any available Grails scripts as Gradle tasks. Simply prefix the Grails script name with grails- and execute from Gradle.
  - e.g. 'gradle grails-refresh-dependencies'
- Passing arguments to Grails Tasks :
  - gradle -PgrailsArgs="FooController" grails-create-controller
- Setting the Grails Environment :
  - gradle -PgrailsEnv=prod grails-run-app

Q/A

Thank you.

Blog: http://www.intelligrape.com/blog/author/bhagwat
LinkedIn: http://www.linkedin.com/in/bhagwatkumar
Twitter: http://twitter.com/bhagwatkumar
Mail : bhagwat@intelligrape.com

References

http://www.gradle.org/docs/current/userguide/userguide_single.html
http://www.gradle.org/docs/current/userguide/java_plugin.html
http://www.gradle.org/docs/current/userguide/groovy_plugin.html
http://plugins.gradle.org/
http://mrhaki.blogspot.in/2010/09/gradle-goodness-run-java-application.html
http://mrhaki.blogspot.in/2009/11/using-gradle-for-mixed-java-and-groovy.html
http://rominirani.com/2014/07/28/gradle-tutorial-part-1-installation-setup/
http://rominirani.com/2014/07/28/gradle-tutorial-part-2-java-projects/
http://rominirani.com/2014/07/29/gradle-tutorial-part-3-multiple-java-projects/
http://rominirani.com/2014/08/12/gradle-tutorial-part-4-java-web-applications/
http://grails.github.io/grails-gradle-plugin/docs/manual/guide/introduction.html
http://www.drdobbs.com/jvm/why-build-your-java-projects-with-gradle/240168608
http://en.wikipedia.org/wiki/List_of_build_automation_software
http://pygments.org/

Demo : https://github.com/bhagwat/gradledemo