One Build To Rule Them All

Building a Full Application Stack With Gradle

John Engelman

Object Partners Inc

@johnrengelman
github.com/johnrengelman

What is a full application stack?

For the purposes of this talk, a "full application stack" is a multi-project build that consists of multiple technologies, languages, frameworks, and/or external integrations that share components and participate in a single build to produce a set of linked output artifacts.

Origin

Single Repository Multi-Project Build

~20 Developers

77 Projects (lots of small libs)

4 Grails Applications w/ AngularJS

Grunt/Gulp & Bower to build JS code

9 Dropwizard Applications

2 Gradle Plugins

Avg. Build Time: ~ 40 mins

Topics

1. Multiproject Best Practices
2. Building Grails Applications & Plugins
3. Building Javascript projects
4. Building Application Distributions

Gradle Multi-Project Best Practices

Project Layout

Flat vs Nested

Flat Layout

```
myproject/
|
+-- foo-lib/
|
+-- bar-service/
|
+-- baz-service/
```

Nested Layout

```
myproject/
|
+-- libs/
|   |
|   +-- foo-lib/
|
+-- services/
    |
    +-- bar-service/
    |
    +-- baz-service/
```

Auto-Detecting Sub-Projects

- Don't need to explicitly include each project
- Automatically find subprojects

```
//settings.gradle
def skippedDirs = ['buildSrc', 'gradle']
def path = [] as LinkedList
rootDir.traverse(
  type: FileType.FILES,
  nameFilter: ~/.*\.gradle/, //find .gradle files
  maxDepth: 1, //limit search depth
  preDir: {
      path << it.name // build up the directory structure
      if (skippedDirs.contains(it.name)) { // ignore skipped directories
          return FileVisitResult.SKIP_SUBTREE
      }
  },
  postDir: { path.removeLast() } //drop the .gradle file off the path
) {
    if (path) {
        include path.join(':') //register the project path using : notation
    }
}
```

Name Gradle files after Project names

- Quicker access to project's build file

```
//settings.gradle
rootProject.children.each { project ->
    setSubprojectBuildFile(project)
}

void setSubprojectBuildFile(def project) {
    String fileBaseName = project.name.replaceAll("\\p{Upper}") { "-${it.toLo
    project.buildFileName = "${fileBaseName}.gradle"
    assert project.buildFile.isFile()
    project.children.each { subproject ->
        setSubprojectBuildFile(subproject)
    }
}
```

```
myproject/
|
+-- fooLib/
|   |
|   +-- foo-lib.gradle
|
+-- barService/
|   |
|   +-- bar-service.gradle
|
+-- build.gradle
|
+-- settings.gradle
```

Demo

Building Grails Projects

Quick Start

```
buildscript {
  repositories { jcenter() }
  dependencies { classpath 'org.grails:grails-gradle-plugin:2.1.0' }
}

apply plugin: 'grails'

grails {
  grailsVersion '2.3.8'
  groovyVersion '2.1.9'
}

repositories { grails.central() }

dependencies {
  bootstrap 'org.grails.plugins:tomcat:7.0.50.1'
}
```

Configuration Options

```
grails {
  grailsVersion = '2.3.8'
  groovyVersion = '2.1.9'
  springLoadedVersion = '1.1.3'
}
```

Resolving Grails Plugins

```
repositories {
  grails.central() // Add the Grails Central Maven repo
}

dependencies {
  bootstrap 'org.grails.plugins:tomcat:7.0.50.1'
  // bootstrap, compile, runtime, test, provided
  // MUST add 'org.grails.plugins'
}
```

Useful Tasks

Task	Action
init	creates a new Grails application project
init-plugin	creates a new Grails Plugin project
run	$ grails run-app
test	$ grails test-app
war	$ grails war
packagePlugin	$ grails package-plugin

Build Pipeline Participation

Task	Dependent Tasks
clean	delete buildPlugins/ dir
assemble	war, packagePlugin
check	no default
build	assemble, check

Run Any Grails Script

```
$ gradle grails-<script-name>
```

Configure Grails tasks

```
run {
  env = 'prod'
  args += '--stacktrace'
}
```

Question: How Does it work?

Answer:

Grails Launcher

Gradle is ***NOT*** building Grails project

What Gradle Does:

1. Resolve dependencies
2. Create GrailsLaunchContext
3. Serialize to file
4. Fork GrailsLauncher.Main in new JVM

What GrailsLauncher Does:

1. Deserialize GrailsLaunchContext from file
2. Create ClassLoader with bootstrap classpath from context
3. Load GrailsScriptRunner class
4. Call GrailsScriptRunner.executeCommand(scriptName, [args], env)

Plugin Gotchyas

- Do **NOT** apply Java/Groovy plugin
- Do **NOT** apply plugins that apply the Java/Groovy plugin
- Support for plugin publishing lacks
  - Grails Release Plugin generates POM from BuildConfig
  - Issues with generating plugin.xml, pom.xml, and packagePlugin UP-TO-DATE

Demo

Building Java Script Projects

Lots of Tools Available

- Grunt
- Gulp
- Bower
- Tomorrow's next cool thing

Integrating with Bower & Grunt

Step 1 - Get Node

```
buildscript {
  repositories { jcenter() }
  dependencies { classpath 'com.moowork.gradle:gradle-node-plugin:0.5' }
}

apply plugin: 'node'
```

Step 2 - Configure Node

```
allprojects {
  plugins.withType(NodePlugin) {
    node {
      version = '0.10.26'
      download = true
      workDir = rootProject.file("${rootProject.buildDir}/nodejs")
    }
  }
}

project.afterEvaluate {
  nodeSetup {
    inputs.property 'version', node.version
    onlyIf { !(it.variant.nodeDir).exists() }
  }
}
```

Step 3 - Configure Project Node tasks

```
task npmClean(type: Delete) {
  delete 'node_modules'
}

project.afterEvaluate {
  //use the 1 Node install from root
  nodeSetup.enabled = false
  npmInstall.dependsOn rootProject.tasks.nodeSetup
}
```

Step 4 - Get Grunt

```
buildscript {
  repositories { jcenter() }
  dependencies { classpath 'com.moowork.gradle:gradle-grunt-plugin:0.5' }
}

apply plugin: 'grunt'


//Make sure NPM is installed before running Grunt
project.afterEvaluate {
    project.tasks.withType(GruntTask) { task ->
        task.dependsOn npmInstall
    }
}
```

Step 5 - Configure Grunt Tasks

```
task compileJs(type: GruntTask) {
  args = ['compile']
}

assemble.dependsOn compileJs
```

Step 6 - Get Bower

```
class BowerTask extends NodeTask {
  private String bowerScriptPath

  BowerTask() {
    group = 'Bower'
  }

  String getBowerScript() {
    bowerScriptPath ?: 'node_modules/bower/bin/bower'
  }

  @Override
  void exec() {
    def localBower = project.file(bowerScript)
    if (!localBower.file) {
      throw new GradleException('Bower not installed')
    }
    setScript(localBower)
    super.exec()
  }
```

Step 7 - Configure Bower Tasks

```
task bowerInstall(type: BowerTask) {
  args = ['install']
  inputs.file 'bower.json'
  inputs.file '.bowerrc'
}

assemble.dependsOn bowerInstall
```

Step 8 - Task Ordering

Example: Grails WAR contains Grunt/Bower output

```
project.tasks.'grails-war'.mustRunAfter compileJs, bowerInstall
```

Demo

Building Application Distributions

Many options

1. Zip/Jar tasks
2. War task
3. Applications plugin
4. Deploy with Gradle
5. Fat (Uber) Jars

Zip/Jar tasks

- Pros
  - Compiles project classes/resources into a single file
  - Part of normal build cycle
  - Built in to Gradle
- Cons
  - No dependency mgmt after build

War task

- Pros
  - Bundles application code with dependencies into a single file
  - Easy to deploy to container
- Cons
  - Only works for things that understand WAR (Tomcat)

Application Plugin

- Pros
  - Creates single Zip file for application
  - Bundles all dependencies
  - Creates shell scripts for executing program
- Cons
  - Zip is just a package for the app. Requires unzipping for deploy

Deploying with Gradle

- Pros
  - Relies on Jar files and POMs
  - Benefit from Gradle's dependency resolution
  - Gradle dependency caching
  - Gradle Wrapper (or provisioning Gradle instance)
  - Light-weight artifact (just need a build.gradle file)
- Cons
  - Complex to initially setup
  - Resolution time
  - ...

Fat (Uber) Jars

- Pros
  - Single jar file with all dependencies
  - Launch as "java -jar file.jar"
  - Runs anywhere a JRE is available (of the right version)
- Cons
  - Longer build time
  - Bigger artifacts (= longer downloads)
  - Still need to manager JAVA_OPTS externally

Ways to FatJar

Like Node integration, lots of ways

- Gradle FatJar Plugin
- Gradle OneJar Plugin
- Gradle Shadow Plugin
- Use Gradle's zipTree and Copy

Using zipTree & Copy

- I/O expensive.
- Has to write out all the jars to disk before creating a new jar
- But, has all the goodies that Gradle provides for copying (filtering, mapping, etc.)

```
task fatJat(type: Jar) {
  from sourceSets.main.output
  from {
    mainSourceSet.runtimeClasspath.collect {
      it (it.name.endsWith('.zip') || it.name.endsWith('.jar')) {
        project.zipTree(it)
      } else {
        project.files(it)
      }
    }
  }
}
```

Gradle Shadow Plugin

- Based on Maven Shade
- Uber-jarring
- Resource transformation
- File filtering
- Class relocating
- Based on Gradle's Jar task (and all it's inherent abilities)
- Uses JarInputStream & JarOutputStream to write file (fast!)

Quick Start

```
buildscript {
  respositories { jcenter() }
  dependencies {
    classpath 'com.github.jengelman.gradle.plugins:shadow:1.0.2'
  }
}

apply plugin: 'com.github.johnrengelman.shadow'

jar {
  manifest {
    attributes 'Main-Class': 'myapp.Main'
  }
}

shadowJar {
  mergeServiceFiles()
}
```

```
$ gradle shadowJar
```

Plugin Defaults

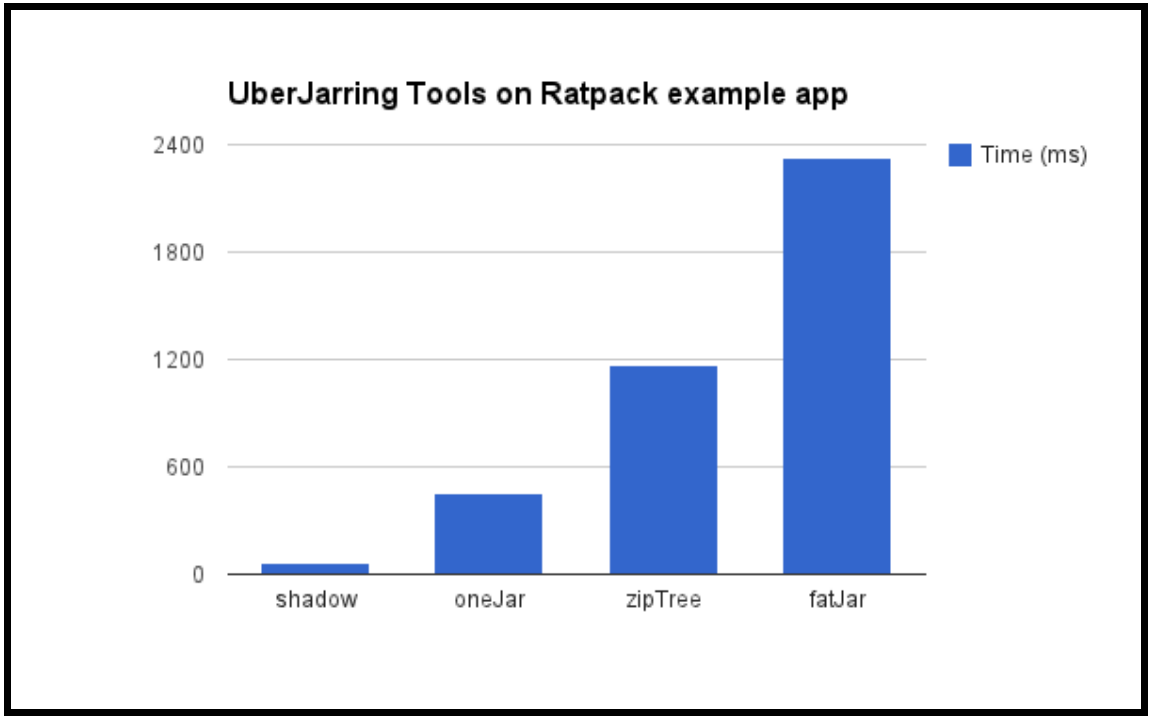
- Includes all dependencies in 'runtime'
- Excludes any 'META-INF/INDEX.LIST', 'META-INF/*.DSA', and 'META-INF/*.RSA' files
- Uses same 'MANIFEST.MF' as 'jar' task
- Classifier is 'all'
- Creates 'shadow' configuration & component and assigns output as an artifact

Speed Comparison

Example Ratpack Gradle App

Total files in resulting Jar: ~4074

Plugin	Time
zipTree (RatPack plugin)	1167 ms
oneJar	452 ms
fatJar	2325 ms
shadow	62.25 ms


UberJarring Tools on Ratpack example app
2400
1800
1200
600
0
shadow
oneJar
zipTree
fatJar
Time (ms)

Integration with Application plugin

```
apply plugin: 'application'
apply plugin: 'com.github.johnrengelman.shadow'

mainClassName = 'myapp.Main'
```

```
$ gradle installShadow
$ gradle runShadow
$ gradle distShadowZip
$ gradle distShadowTar
```

Demo

OPI

- Java, Groovy, Javascript, Mobile, Open Source
- ~ 100 Senior Consultants
  - Minneapolis, MN & Omaha, NE
  - Chicago, IL & Denver, CO
  - Average tenure > 5 years
- Founded in 1996

objectpartners.com
@objectpartners
objectpartners.com/blog