Cédric Champeau
@CedricChampeau
version 1.0,2015-05-05

Convention over configuration: the Gradle way

Who am I

```
speaker {
    name 'Cédric Champeau'
    company 'Gradle Inc'
    oss 'Apache Groovy committer',
    successes (['Static type checker',
                'Static compilation',
                'Traits',
                'Markup template engine',
                'DSLs'])
        failures Stream.of(bugs),
        twitter '@CedricChampeau',
        github 'melix',
        extraDescription '''Groovy in Action 2 co-author
Misc OSS contribs (Gradle plugins, deck2pdf, jlangdetect,
...)'''
}
```

Gradle

- A dependency management engine
- A dependency based execution system
- A plugin system
- A set of plugins

Gradle

- Open Source
- Language agnostic
    - builds Java, Groovy, Scala, C++, assembler, ...
- Designed for extensions
- Integrates well with Ivy, Ant, Maven, ...

About this slidedeck

- Slides written using Asciidoctor
- Converted to deck.js
- Exported to PDF thanks to deck2pdf
- Integrated with Gradle

```
repositories {
    jcenter()
}

apply plugin: 'com.github.jruby-gradle.base'
apply plugin: 'org.ysb33r.vfs'
apply plugin: 'java'
apply plugin: 'org.asciidoctor.convert'

dependencies {
    gems 'rubygems:haml:4.0.6'
}
```

Export to PDF

```
configurations {
    pdf
}
repositories {
    mavenLocal()
}
dependencies {
    pdf 'com.github.melix:deck2pdf:0.2'
}

task generatePdf(type:JavaExec) {
    dependsOn asciidoctor
    main = 'me.champeau.deck2pdf.Main'
    workingDir = file("$buildDir/asciidoc/deckjs")
    args = ['index.html','gradle-coc.pdf']
    classpath = configurations.pdf

    inputs.file("$workingDir/index.html")
    outputs.file("$workingDir/gradle-coc.pdf")
}
```

Gradle basics

- build scripts (often *build.gradle*)
- a task execution graph
- inline "plugins"
- binary plugins
- plugin portal
- Gradle wrapper

Gradle basics

> Make the impossible possible,
> make the possible easy
> and make the easy elegant
>
> **inspired by Moshe Feldenkrais**

First step : Gradle wrapper

- Enforces a specific version of Gradle
- Wrapper committed in VCS
- Makes sure that the version of Gradle being used is the one the project is supposed to be compiled with
- There’s no good reason not to use it

> Always use the wrapper. Always.
>
> **me**

Convention over configuration

A simple Java project

```
apply plugin: 'java'

repositories {
    jcenter()
}

dependencies {
   testCompile "junit:junit:4.12"
}
```

Convention over configuration

Before build

```
basic-java
├── build.gradle
└── src
    ├── main
    │   └── java
    │       └── com
    │           └── acme
    │               └── Greeter.java
    └── test
        └── java
            └── com
                └── acme
                    └── GreeterTest.java
```

Convention over configuration

After build

```
basic-java/
├── build
│   ├── classes
│   │   ├── main
│   │   │   └── com
│   │   │       └── acme
│   │   │           └── Greeter.class
│   │   └── test
│   │       └── com
│   │           └── acme
│   │               └── GreeterTest.class
│   ├── libs
│   │   └── basic-java.jar
│   ├── reports
│   ├── test-results
│   └── tmp
├── build.gradle
└── src
    ...
```

Integration tests

What conventions exist for defining integration tests?

Integration tests

What conventions exist for defining integration tests?

Defining a new source set

```
sourceSets {
    integTest {
        groovy.srcDir file('src/integTest/groovy')
        resources.srcDir file('src/integTest/resources')
    }
}
```

Integration tests

What conventions exist for defining integration tests?

Defining a new source set

```
sourceSets {
    integTest {
        groovy.srcDir file('src/integTest/groovy')
        resources.srcDir file('src/integTest/resources')
    }
}
```

Create an integTest task

```
task integTest(type:Test) {
    dependsOn jar
    mustRunAfter jar
    testClassesDir = sourceSets.integTest.output.classesDir
    classpath = sourceSets.integTest.runtimeClasspath
}
check.dependsOn(integTest)
```

Integration tests

What conventions exist for defining integration tests?

Defining a new source set

```
sourceSets {
    integTest {
        groovy.srcDir file('src/integTest/groovy')
        resources.srcDir file('src/integTest/resources')
    }
}
```

Create an integTest task

```
task integTest(type:Test) {
    dependsOn jar
    mustRunAfter jar
    testClassesDir = sourceSets.integTest.output.classesDir
    classpath = sourceSets.integTest.runtimeClasspath
}
check.dependsOn(integTest)
```

Define the integration test classpath

```
configurations {
    integTestCompile
    integTestRuntime
}
dependencies {
    integTestCompile 'org.codehaus.groovy:groovy:2.4.3'
    integTestCompile('org.spockframework:spock-core:1.0-groovy-2.4') {
        exclude group: 'org.codehaus.groovy'
    }
    integTestCompile files(jar.archivePath)
}
```

Write the integration test

```
package com.acme
import spock.lang.Specification

class GreetingSpec extends Specification {

   def "should greet people"() {
      given: "a greeter instance"
         def greeter = new Greeter()
      when: "we greet someone"
         def greeting = greeter.greet(someone)
      then: "greeting looks correct"
         greeting == "Hello, $someone!"
      where:
         someone << ['Bob','Alice']
   }

}
```

Write the integration test

```
package com.acme
import spock.lang.Specification

class GreetingSpec extends Specification {

   def "should greet people"() {
      given: "a greeter instance"
         def greeter = new Greeter()
      when: "we greet someone"
         def greeting = greeter.greet(someone)
      then: "greeting looks correct"
         greeting == "Hello, $someone!"
      where:
         someone << ['Bob','Alice']
   }

}
```

Run it

```
$ ./gradlew --daemon j-i:iT
:java-inttest:compileJava
:java-inttest:compileGroovy UP-TO-DATE
:java-inttest:processResources UP-TO-DATE
:java-inttest:classes
:java-inttest:jar
:java-inttest:compileIntegTestJava UP-TO-DATE
:java-inttest:compileIntegTestGroovy
:java-inttest:processIntegTestResources UP-TO-DATE
:java-inttest:integTestClasses
:java-inttest:integTest

BUILD SUCCESSFUL

Total time: 3.646 secs
```

Creating a source jar

```
task sourcesJar(type: Jar) {
    classifier = 'sources'
    from sourceSets.main.allSource
}
```

What about other source trees?

What about other source trees?

Building a custom convention

```
task sourcesJar()

sourceSets.each { sourceSet ->
    Task t = task "${sourceSet.name}SourcesJar"(type:Jar) {
        classifier = sourceSet.name=='main'?'sources':"${sourceSet.name}-
sources"
        from sourceSet.allSource
    }
    sourcesJar.dependsOn(t)
}
```

What about other source trees?

Building a custom convention

```
task sourcesJar()

sourceSets.each { sourceSet ->
    Task t = task "${sourceSet.name}SourcesJar"(type:Jar) {
        classifier = sourceSet.name=='main'?'sources':"${sourceSet.name}-
sources"
        from sourceSet.allSource
    }
    sourcesJar.dependsOn(t)
}
```

Output directory

```
examples/java-custom-artifacts/build/libs/
├── java-custom-artifacts-integTest-sources.jar
├── java-custom-artifacts-sources.jar
└── java-custom-artifacts-test-sources.jar
```

What about other source trees?

Building a custom convention

```
task sourcesJar()

sourceSets.each { sourceSet ->
    Task t = task "${sourceSet.name}SourcesJar"(type:Jar) {
        classifier = sourceSet.name=='main'?'sources':"${sourceSet.name}-
sources"
        from sourceSet.allSource
    }
    sourcesJar.dependsOn(t)
}
```

Output directory

```
examples/java-custom-artifacts/build/libs/
├── java-custom-artifacts-integTest-sources.jar
├── java-custom-artifacts-sources.jar
└── java-custom-artifacts-test-sources.jar
```

Can we make it cleaner?

- How about sharing it with other sub-projects?
- How about sharing it with external projects?

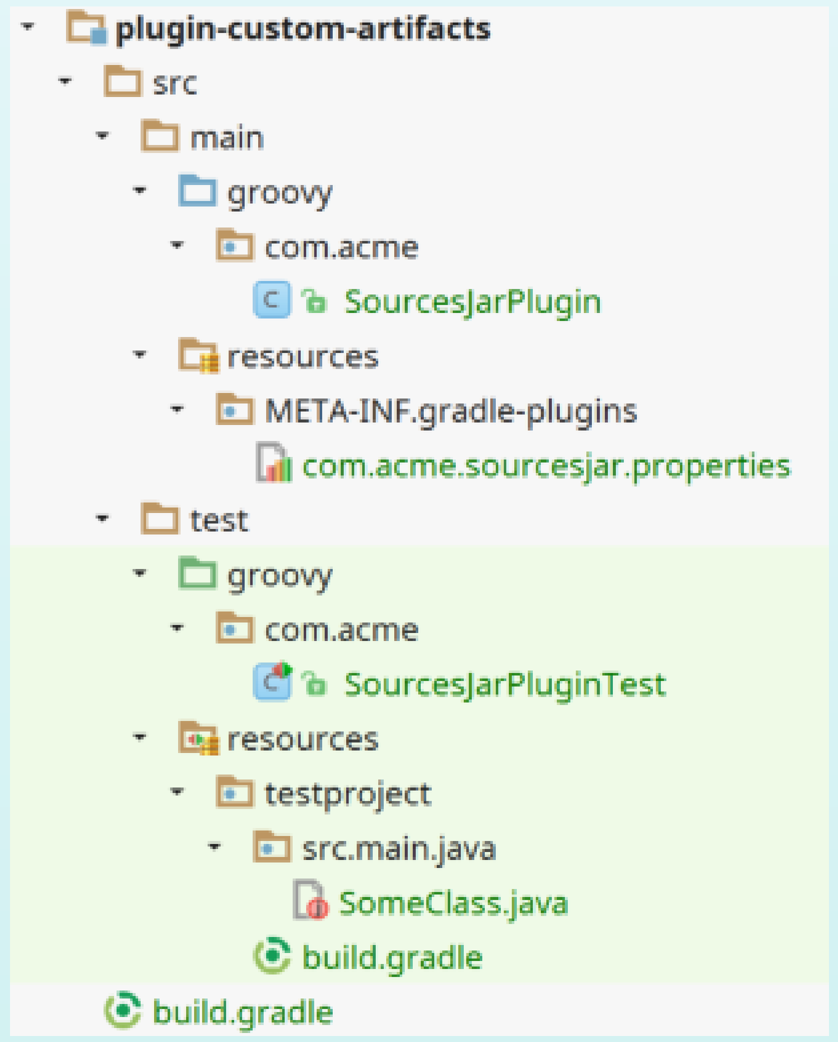
Writing a plugin

- Usually written in Groovy
- Can be written in Java
- Other languages not recommended (adds dependencies)

Writing a plugin

- Usually written in Groovy
- Can be written in Java
- Other languages not recommended (adds dependencies)

Plugin structure



```
plugin-custom-artifacts
  src
    main
      groovy
        com.acme
          SourcesJarPlugin
      resources
        META-INF.gradle-plugins
          com.acme.sourcesjar.properties
    test
      groovy
        com.acme
          SourcesJarPluginTest
      resources
        testproject
          src.main.java
            SomeClass.java
          build.gradle
  build.gradle
```

build.gradle

```
apply plugin: 'groovy'

group = 'com.acme.gradle'
version = '1.0-SNAPSHOT'

repositories {
    jcenter()
}

dependencies {
    compile gradleApi()
    compile localGroovy()
    testCompile 'org.spockframework:spock-core:1.0-groovy-2.3'
}
```

Writing a plugin

Writing a plugin

Plugin class

```
class SourcesJarPlugin implements Plugin<Project> {
    @Override
    void apply(Project project) {
        // do something
    }
}
```

Writing a plugin

Plugin class

```
class SourcesJarPlugin implements Plugin<Project> {
    @Override
    void apply(Project project) {
        // do something
    }
}
```

Descriptor

src/main/resources/META-INF/gradle-plugins/com.acme.sourcesjar.properties

```
implementation-class=com.acme.SourcesJarPlugin
```

Plugin code

SourcesJarPlugin.groovy

```
class SourcesJarPlugin implements Plugin<Project> {
    @Override
    void apply(Project project) {
        project.afterEvaluate {
            def sourcesJarTask = project.tasks.create('sourcesJar')
            project.sourceSets.each { sourceSet ->
                Task t = project.tasks.create(name:
"${sourceSet.name}SourcesJar", type: Jar) {
                    classifier = sourceSet.name == 'main' ? 'sources' :
"${sourceSet.name}-sources"
                    from sourceSet.allSource
                }
                sourcesJarTask.dependsOn(t)
                project.artifacts.add("archives", t)
            }
        }
    }
}
```

Test the plugin

SourcesJarPluginTest.groovy

```groovy
def "tasks are created when plugin is applied"() {
     given: "A sample project"
     Project project = ProjectBuilder.builder()
             .withProjectDir(projectDir).build()

     when: "We apply the sourcesjar plugin"
     project.apply(plugin:'com.acme.sourcesjar')

     then: "the sourcesJar task is created"
     project.getTasksByName('sourcesJar', false).size() == 1
 }
```

Publishing the plugin

Publishing the plugin

The maven-publish plugin

```
apply plugin: 'maven-publish'

publishing {
    publications {
        mavenJava(MavenPublication) {
            from components.java
        }
    }
}
```

Publishing the plugin

The maven-publish plugin

```
apply plugin: 'maven-publish'

publishing {
    publications {
        mavenJava(MavenPublication) {
            from components.java
        }
    }
}
```

Publish to local repo

```
$ ./gradlew --daemon publishToMavenLocal
:plugin-custom-artifacts:generatePomFileForMavenJavaPublication
:plugin-custom-artifacts:compileJava UP-TO-DATE
:plugin-custom-artifacts:compileGroovy
:plugin-custom-artifacts:processResources
:plugin-custom-artifacts:classes
:plugin-custom-artifacts:jar
:plugin-custom-artifacts:publishMavenJavaPublicationToMavenLocal
:plugin-custom-artifacts:publishToMavenLocal

BUILD SUCCESSFUL

Total time: 2.011 secs
```

Use the plugin

```
buildscript {
    repositories {
        mavenLocal()
    }
    dependencies {
        classpath 'com.acme.gradle:plugin-custom-artifacts:1.0-SNAPSHOT'
    }
}

apply plugin: 'com.acme.sourcesjar'
```

Use the plugin

```
buildscript {
    repositories {
        mavenLocal()
    }
    dependencies {
        classpath 'com.acme.gradle:plugin-custom-artifacts:1.0-SNAPSHOT'
    }
}

apply plugin: 'com.acme.sourcesjar'
```

Output

```
$ ./gradlew sourcesJar
:java-custom-artifacts:integTestSourcesJar UP-TO-DATE
:java-custom-artifacts:mainSourcesJar UP-TO-DATE
:java-custom-artifacts:testSourcesJar UP-TO-DATE
:java-custom-artifacts:sourcesJar UP-TO-DATE
:java-custom-artifacts-using-plugin:integTestSourcesJar
:java-custom-artifacts-using-plugin:mainSourcesJar
:java-custom-artifacts-using-plugin:testSourcesJar
:java-custom-artifacts-using-plugin:sourcesJar
:java-inttest-sourcesjar:sourcesJar UP-TO-DATE

BUILD SUCCESSFUL

Total time: 0.704 secs
```

Conventions evolved

- the **model** is important
- plugins can reason on the model rather than outputs

Conventions evolved

- the **model** is important
- plugins can reason on the model rather than outputs

Generating a checksum for all artifacts

```
import org.gradle.api.DefaultTask
import org.gradle.api.tasks.TaskAction

class Checksum extends DefaultTask {
    Checksum() {
        project.afterEvaluate {
            project.configurations.each {
                it.artifacts.each { artifact ->
                    this.dependsOn(artifact.buildDependencies)
                }
            }
        }
    }

    @TaskAction
    void exec() {
        project.configurations.each {
            it.artifacts.files.each { f ->
                project.ant.checksum(file:f)
            }
        }
    }
}
```

Conventions evolved

- the **model** is important
- plugins can reason on the model rather than outputs

Generating a checksum for all artifacts

```
import org.gradle.api.DefaultTask
import org.gradle.api.tasks.TaskAction

class Checksum extends DefaultTask {
    Checksum() {
        project.afterEvaluate {
            project.configurations.each {
                it.artifacts.each { artifact ->
                    this.dependsOn(artifact.buildDependencies)
                }
            }
        }
    }

    @TaskAction
    void exec() {
        project.configurations.each {
            it.artifacts.files.each { f ->
                project.ant.checksum(file:f)
            }
        }
    }
}
```

Using the task

```
allprojects {
    task checksum(type: Checksum)
}
```

Improving it

- Each task can define its inputs/outputs
- You should **always** do it

Improving it

- Each task can define its inputs/outputs
- You should **always** do it

Improved checksum

```
project.afterEvaluate {
    project.configurations.each {
        it.artifacts.each { artifact ->
            this.dependsOn(artifact.buildDependencies)
        }
        it.artifacts.files.each { f ->
            inputs.file(f)
            outputs.file("${f}.MD5")
        }
    }
}
```

Convention over configuraton : versioning

```
configurations.all {
    resolutionStrategy.eachDependency { DependencyResolveDetails details ->
        if (details.requested.version == 'default') {
            def version = findDefaultVersionInCatalog(details.requested.group,
details.requested.name)
            details.useVersion version
        }
    }
}
```

A glance at the future

A glance at the future

Android build plugin

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 20
    buildToolsVersion "20.0.0"

    defaultConfig {
        applicationId "com.mycompany.myapplication"
        minSdkVersion 13
        targetSdkVersion 20
        versionCode 1
        versionName "1.0"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
         debug {
            debuggable true
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:20.0.0'
    compile project(path: ':app2, configuration: 'android-endpoints')
}
```

Defining your own model

- A new model API is in the works
- Allows definining custom models
- Models can be shared between plugins
- Allows faster build execution as well as parallel task execution

Defining your own model

- A new model API is in the works
- Allows definining custom models
- Models can be shared between plugins
- Allows faster build execution as well as parallel task execution

Example

```
@Managed
interface Person {
  void setFirstName(String n); String getFirstName()
  void setLastName(String n); String getLastName()
}

class PersonRules extends RuleSource {
  @Model void person(Person p) {}

  @Mutate void setFirstName(Person p) {
    p.firstName = "John"
  }

 @Mutate void createHelloTask(CollectionBuilder<Task> tasks, Person p) {
    tasks.create("hello") {
      doLast {
        println "Hello $p.firstName $p.lastName!"
      }
    }
  }
}

apply plugin: PersonRules
```

Questions

Thank you!

- Slides and code : https://github.com/melix/bdxjug2015
- Gradle documentation : http://gradle.org/docs/current/userguide/userguide
- Follow me: @CedricChampeau