VARIANT-AWARE DEPENDENCY MANAGEMENT WITH GRADLE

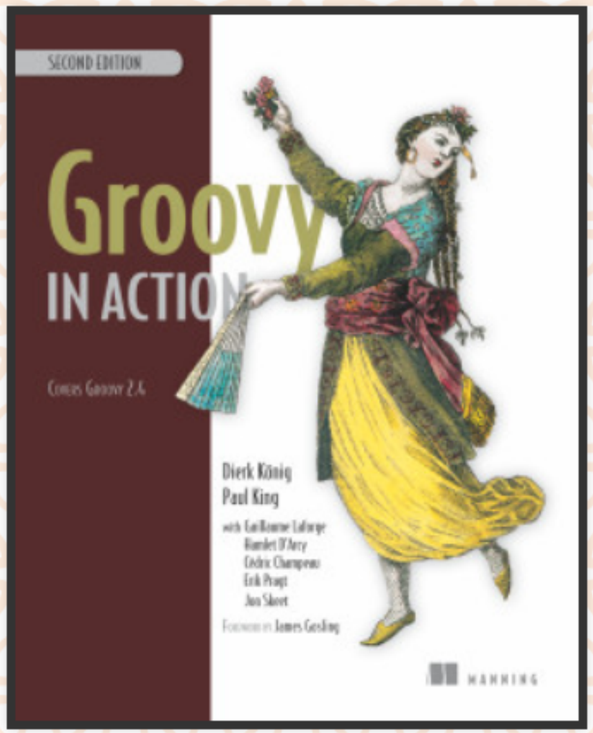
Cédric Champeau

June 1st - 3rd 2016
Copenhagen, Denmark

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, ...)'''
}
```

GROOVY IN ACTION 2



https://www.manning.com/books/groovy-in-action-second-edition

Coupon *ctwgr8conftw*

PRINCIPLES

- Apps target different platforms
- Apps have different flavors
- Each of them have different dependency set
- How can you create your own variants?

DECLARING A DEPENDENCY

DECLARE A DEPENDENCY ON A LIBRARY OF THE SAME PROJECT

```
model {
    components {
        main(JvmLibrarySpec) {
            sources {
                java {
                    dependencies {
                        library 'dep'
                    }
                }
            }
        }
        dep(JvmLibrarySpec)
    }
}
```

DECLARE A DEPENDENCY ON A LIBRARY FROM ANOTHER PROJECT

```
model {
    components {
        main(JvmLibrarySpec) {
            sources {
                java {
                    dependencies {
                        project ':other' library 'dep'
                    }
                }
            }
        }
    }
}
```

SHORTCUT NOTATION IF TARGET PROJECT ONLY HAS ONE LIBRARY

```
model {
    components {
        main(JvmLibrarySpec) {
            sources {
                java {
                    dependencies {
                        project ':other'
                    }
                }
            }
        }
    }
}
```

PLATFORM-AWARE DEPENDENCY MANAGEMENT

DECLARING TARGET PLATFORM OF A COMPONENT

```
model {
    components {
        main(JvmLibrarySpec) {
            targetPlatform 'java6'
            targetPlatform 'java7'
        }
        dep(JvmLibrarySpec) {
            targetPlatform 'java7'
        }
    }
}
```

DEPENDENCIES WITH PLATFORMS (1/3)

```
model {
    components {
        main(JvmLibrarySpec) {
            targetPlatform 'java6'
            targetPlatform 'java7'
            sources {
                java {
                    dependencies {
                        library 'dep'
                    }
                }
            }
        }
        dep(JvmLibrarySpec) {
            targetPlatform 'java7'
        }
    }
}
```

DEPENDENCIES WITH PLATFORMS (2/3)

```
$ ./gradlew platform-aware:java7MainJar
:platform-aware:compileDepJarDepJava
:platform-aware:createDepJar
:platform-aware:depJar
:platform-aware:compileJava7MainJarMainJava
:platform-aware:createJava7MainJar
:platform-aware:java7MainJar

BUILD SUCCESSFUL

Total time: 1.264 secs
```

DEPENDENCIES WITH PLATFORMS (3/3)

```
$ ./gradlew platform-aware:java6MainJar

FAILURE: Build failed with an exception.

* What went wrong:
Could not resolve all dependencies for 'Jar 'java6MainJar'' source set '
> Cannot find a compatible binary for library 'dep' (Java SE 6). Availab

* Try:
Run with --stacktrace option to get the stack trace. Run with --info or

BUILD FAILED

Total time: 1.169 secs
```

IN SHORT

- A component can target multiple versions of Java
- Produces as many jars as there are target platforms
- By default, will use the platform Gradle is executed with
- A dependency will resolve to the **highest compatible version**
- Error messages tell which platforms are available

CUSTOM LIBRARIES

- It is possible to define **custom library types**
- To be compatible with JVM dependency management, they have to
  - extend `LibrarySpec`
  - produce at least one binary of type `JarBinarySpec`

EXAMPLE

```
interface MyLib extends LibrarySpec {}
class DefaultMyLib extends BaseComponentSpec implements MyLib {}

class ComponentTypeRules extends RuleSource {

    @ComponentType
    void registerComponent(
            TypeBuilder<MyLib> builder) {
        builder.defaultImplementation(DefaultMyLib)
    }

    @ComponentBinaries
    void createBinaries(
            ModelMap<JarBinarySpec> binaries,
            MyLib library,
            PlatformResolvers platforms,
            @Path("buildDir") File buildDir,
```

USAGE

```
model {
    components {
        main(MyLib) {
            sources {
                java(JavaSourceSet)
            }
        }
    }
}
```

DEPENDENCIES ONTO CUSTOM COMPONENTS

```
model {
    components {
        main(MyLib) {
            sources {
                java(JavaSourceSet) {
                    dependencies {
                        library 'dep'
                    }
                }
            }
        }
        dep(MyLib) {
            sources {
                java(JavaSourceSet)
            }
        }
    }
}
```

WHAT DEPENDS ON WHAT?

- A `JvmLibrarySpec` can depend on a `CustomLib`
- A `CustomLib` can depend on a `JvmLibrarySpec`
- Combibations of the above

DEFINING CUSTOM BINARIES

- Custom binaries are the entry point for custom variants
- Can be unmanaged or `@Managed`

EXAMPLE

```
@Managed
interface MyJar extends JarBinarySpec {}

...

    @ComponentBinaries
    void createBinaries(
            ModelMap<MyJar> binaries,
            MyLib library,
            PlatformResolvers platforms,
            @Path("buildDir") File buildDir,
            JavaToolChainRegistry toolChains) {

        def platform = platforms.resolve(JavaPlatform, DefaultPlatformR
        def toolChain = toolChains.getForPlatform(platform)
        def baseName = "${library.name}"
        String binaryName = "${baseName}Jar"
```

CUSTOM VARIANT DIMENSIONS

MAIN STEPS

- Define a custom library type
- Define a custom binary type
- Annotate some properties with `@Variant`

@VARIANT

- Must be set on a getter
- The type can be either a `String`
- or a `? extend Named`

DEFINE A CUSTOM LIBRARY TYPE

- The library interface also defines the DSL

```
trait MyLib implements LibrarySpec {
    List<String> flavors = []
    void flavors(String... flavors) { this.flavors.addAll(flavors as Lis
}
class DefaultMyLib extends BaseComponentSpec implements MyLib {}
```

DEFINE THE BINARY TYPE

```
@Managed
interface MyJar extends org.gradle.jvm.internal.JarBinarySpecInternal {
    @Variant
    String getFlavor()
    void setFlavor(String flavor)
}
```

UPDATE THE @COMPONENTBINARIES METHOD TO CREATE BINARIES

```
        library.flavors.each { flavor ->
            String binaryName = "${baseName}${flavor.capitalize()}Jar"
            binaries.create(binaryName, MyJar) { jar ->
                jar.toolChain = toolChain
                jar.targetPlatform = platform
                jar.flavor = flavor
            }
        }
```

DESCRIBE THE COMPONENTS

```
model {
    components {
        main(MyLib) {
            flavors 'free','paid'
            sources {
                java(JavaSourceSet) {
                    dependencies {
                        library 'dep'
                    }
                }
            }
        }
    }
    dep(MyLib) {
        flavors 'free'
        sources {
            java(JavaSourceSet)
        }
    }
```

BUILD THE "FREE" VARIANT

```
$ ./gradlew customvariants:mainFreeJar
...
:customvariants:compileDepFreeJarDepJava
:customvariants:createDepFreeJar
:customvariants:depFreeJar
:customvariants:compileMainFreeJarMainJava
:customvariants:createMainFreeJar
:customvariants:mainFreeJar

BUILD SUCCESSFUL

Total time: 2.08 secs
```

BUILD THE "PAID" VARIANT

```
$ gw customvariants:mainPaidJar
...

FAILURE: Build failed with an exception.

* What went wrong:
Could not resolve all dependencies for 'Jar 'mainPaidJar'' source set 'J
> Cannot find a compatible binary for library 'dep' (Java SE 8).
      Required platform 'java8', available: 'java8'
      Required flavor 'paid', available: 'free'


* Try:
Run with --stacktrace option to get the stack trace. Run with --info or
```

DETAILED ERRORS (1/2)

```
$ ./gradlew customvariants:fourthFreeJar
...

FAILURE: Build failed with an exception.

* What went wrong:
Could not resolve all dependencies for 'Jar 'fourthFreeJar'' source set
> Cannot find a compatible binary for library 'third' (Java SE 8).
      Required platform 'java8', available: 'java8' on Jar 'thirdDemoJar
      Required flavor 'free', available: 'demo' on Jar 'thirdDemoJar','s
```

DETAILED ERRORS (2/2)

```
$ gw customvariants:fifthJar
...

FAILURE: Build failed with an exception.

* What went wrong:
Could not resolve all dependencies for 'Jar 'fifthJar'' source set 'Java
> Multiple binaries available for library 'third' (Java SE 8) :
     - Jar 'thirdDemoJar':
         * flavor 'demo'
         * targetPlatform 'java8'
     - Jar 'thirdSharewhareJar':
         * flavor 'sharewhare'
         * targetPlatform 'java8'
```

HOW MATCHING WORKS

- `JavaPlatform` matches the **highest compatible platform**
- If a variant is `null` in either the consumer or the candidate library: dimension is **ignored**
- If a variant is not null:
  - if the type is `String`, matches **stricly**
  - if the type is `Named`, matches **strictly** on `getName()`, unless a custom `VariantDimensionSelector` is provided
  - `JavaPlatform` is handled through a specific `VariantDimensionSelector` (`DefaultJavaPlatformVariantDimensionSelector`)

BONUS: BINARY-LEVEL DEPENDENCIES

VARIANT-SPECIFIC DEPENDENCIE

- A **source set** can define its own dependencies

```
model {
    components {
        main(AndroidLikeLib) {
            flavors 'free', 'paid'
            sources {
                paid {
                    // some extra dependencies for paid flavor
                }
            }
        }
    }
}
```

SHARED SOURCE SET DEPENDENCIES

- But for a shared source set?

```
model {
    components {
        main(JvmLibrarySpec) {
            targetPlatform 'java6'
            targetPlatform 'java7'
            binaries.named('java6MainJar') {
                sources.named('java') {
                    source.srcDir file("$projectDir/src/main/java7compa
                }
            }
            binaries.named('java7MainJar') {
                sources.named('java') {
                    dependencies {
                        library 'dep'
                    }
                }
            }
```

WHAT WE CAN DO?

- Create new binary level source sets
- But no variation of dependencies for a specific source set

```
model {
    components {
        main(JvmLibrarySpec) {
            targetPlatform 'java6'
            targetPlatform 'java7'
            binaries.named('java6MainJar') {
                sources {
                    java6(JavaSourceSet) // creates a Java 6 specific so
                    dependencies {
                        project ':other' library 'lib'
                    }
                }
            }
        }
    }
}
```

CONCLUSION : GRADLE...

- Natively supports Java variant aware dependency management
- Supports defining custom variant dimensions

BUT...

- New (experimental) Java software model only
- No support for external dependencies yet
- No support for publishing either

QUESTIONS?