Gradle

Advanced Dependency Management with Gradle

Benjamin Muschko, Gradle Inc.

Custom requirements in complex builds

Dependency management requires conscious decisions and trade-offs…

- Transitive dependencies
- Broken module versions
- Caching strategies
- Accessing res. artifacts
- Enterprise requirements
- Modifying metadata

Gradle

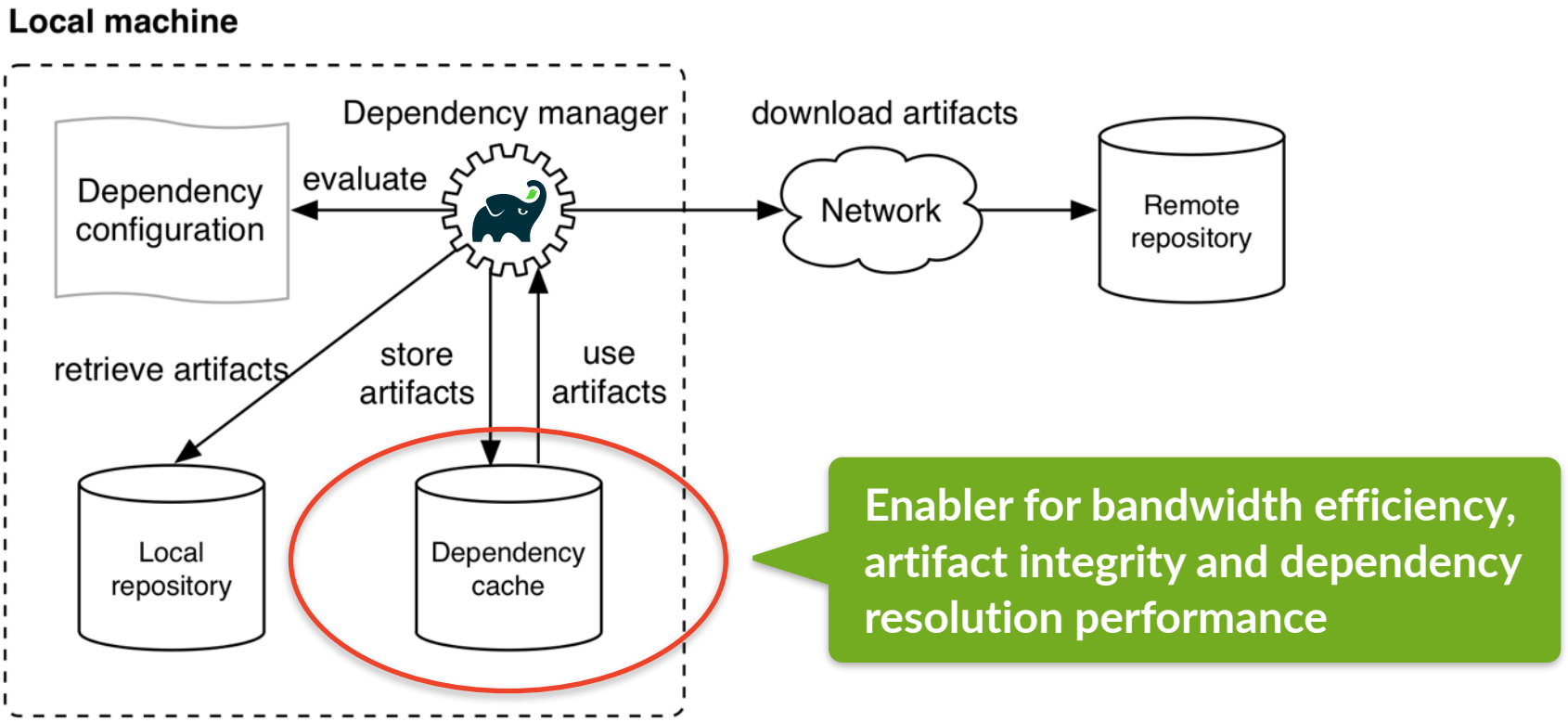
Deep model and API as enabler

Dependency management runtime behavior can be fine-tuned…

- Dependency resolve rules
- Component metadata rules
- Component selection rules
- Artifact Query API

Gradle

Gradle's dependency cache



Enabler for bandwidth efficiency, artifact integrity and dependency resolution performance

Gradle

Opaque cache

I updated the Gradle version and now all dependencies are downloaded again. What happened to the cache?

Gradle

Opaque cache

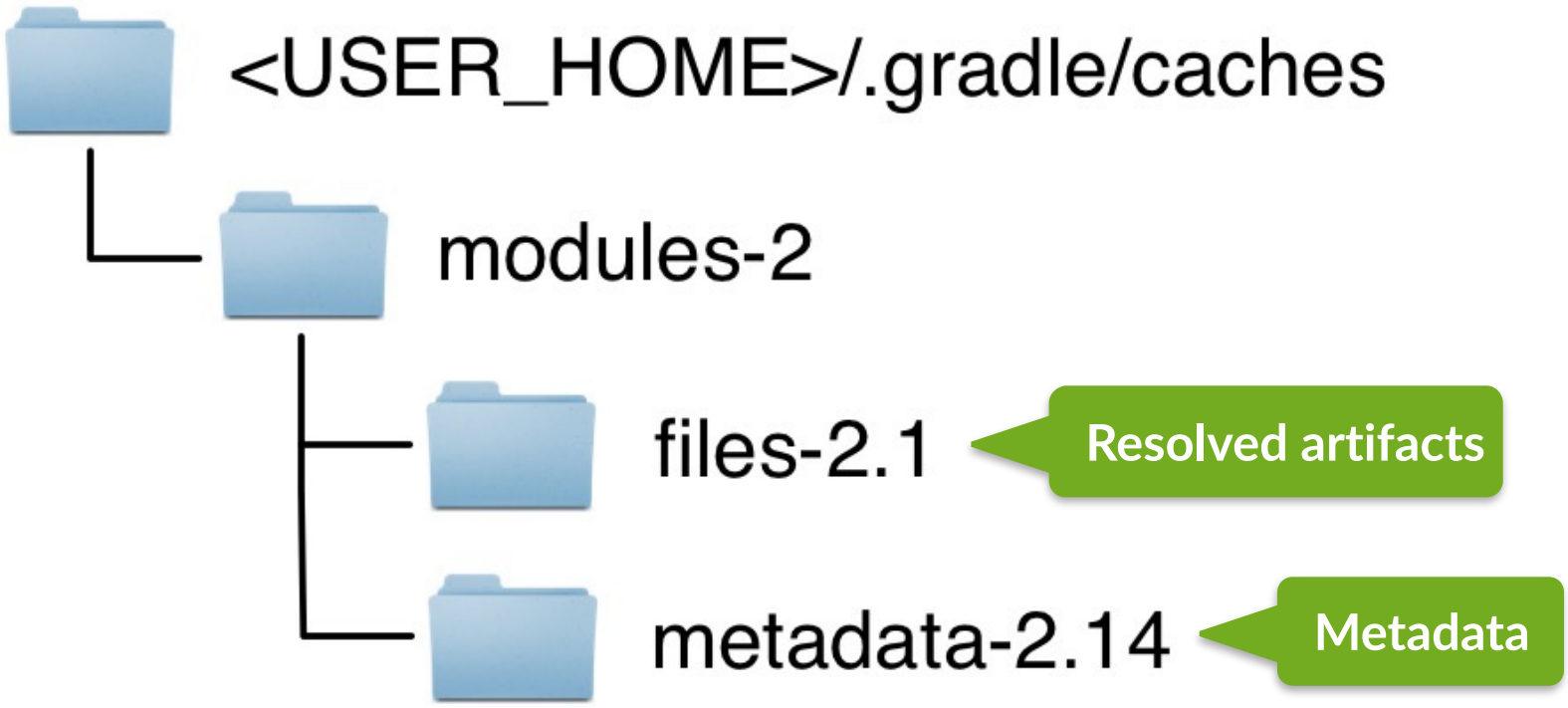
I updated the Gradle version and now all dependencies are downloaded again. What happened to the cache?

The cache structure is versioned and might change across Gradle versions.

Allows for cache optimizations and reorganizations

Gradle

Cache structure



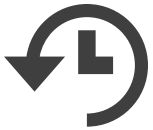
Gradle

Performance features

Lazy download of binary artifacts

Checksum-based download

Minimize number of HTTP calls



TTL for dynamic/changing modules

Gradle

Other features

Store metadata on artifact's origin

Concurrency-safe

Optimize disk usage

Gradle

Time To Live (TTL) for cached dependencies

My build consumes a SNAPSHOT dependency. I know that it changed 10 mins ago but Gradle doesn't resolve it properly.

Gradle

Time To Live (TTL) for cached dependencies

My build consumes a SNAPSHOT dependency. I know that it changed 10 mins ago but Gradle doesn't resolve it properly.

Gradle caching kicks in. Changing and dynamic versions are not checked on the remote repository for 24 hours.

Gradle

Time To Live (TTL) for cached dependencies

My build consumes a SNAPSHOT dependency. I know that it changed 10 mins ago but Gradle doesn't resolve it properly.

Gradle caching kicks in. Changing and dynamic versions are not checked on the remote repository for 24 hours.

How can I change the default behavior?

Gradle

Fine-tuning dependency caching

TTL for changing and dynamic modules can be customized

```
configurations.all {
    resolutionStrategy.cacheDynamicVersionsFor 10, 'minutes'
    resolutionStrategy.cacheChangingModulesFor 4, 'hours'
}
```

Gradle

Resolving a particular Maven SNAPSHOT version

All new SNAPSHOT versions of library Y from today are broken. Can I depend on a SNAPSHOT with a timestamp?

Gradle

Resolving a particular Maven SNAPSHOT version

All new SNAPSHOT versions of library Y from today are broken. Can I depend on a SNAPSHOT with a timestamp?

```
dependencies {
    compile 'org.gradle.training:mylib:1.0-20150423.152626-8'
}
```

Gradle 2.4

Gradle

Cache command line options

Working offline:

```
--offline
```

Fresh resolve of cache dependencies:

```
--refresh-dependencies
```

Debugging resolution failures

A failed resolution doesn't give you the cause...

```
$ gradle dependencies --configuration compile

compile - Compile classpath for source set 'main'.
+--- commons-lang:commons-lang:2.5
\--- javax.mail:mail:1.3.12 FAILED
```

Gradle

Debugging resolution failures

Logging levels are your friend...

```
$ gradle dependencies --configuration compile --info

compile - Compile classpath for source set 'main'.
Resource missing. [HTTP GET: https://repo1.maven.org/maven2/
javax/mail/mail/1.3.12/mail-1.3.12.pom]
Resource missing. [HTTP HEAD: https://repo1.maven.org/maven2/
javax/mail/mail/1.3.12/mail-1.3.12.jar]
+--- commons-lang:commons-lang:2.5
\--- javax.mail:mail:1.3.12 FAILED
```

Gradle

Resolution works, compilation fails

Dependency report looks good...

```
$ gradle dependencies --configuration compile

compile - Compile classpath for source set 'main'.
+--- commons-lang:commons-lang:2.5
\--- javax.mail:mail:1.3.1
     \--- javax.activation:activation:1.0.2
```

Gradle

Resolution works, compilation fails

Compilation requires compile configuration as input.

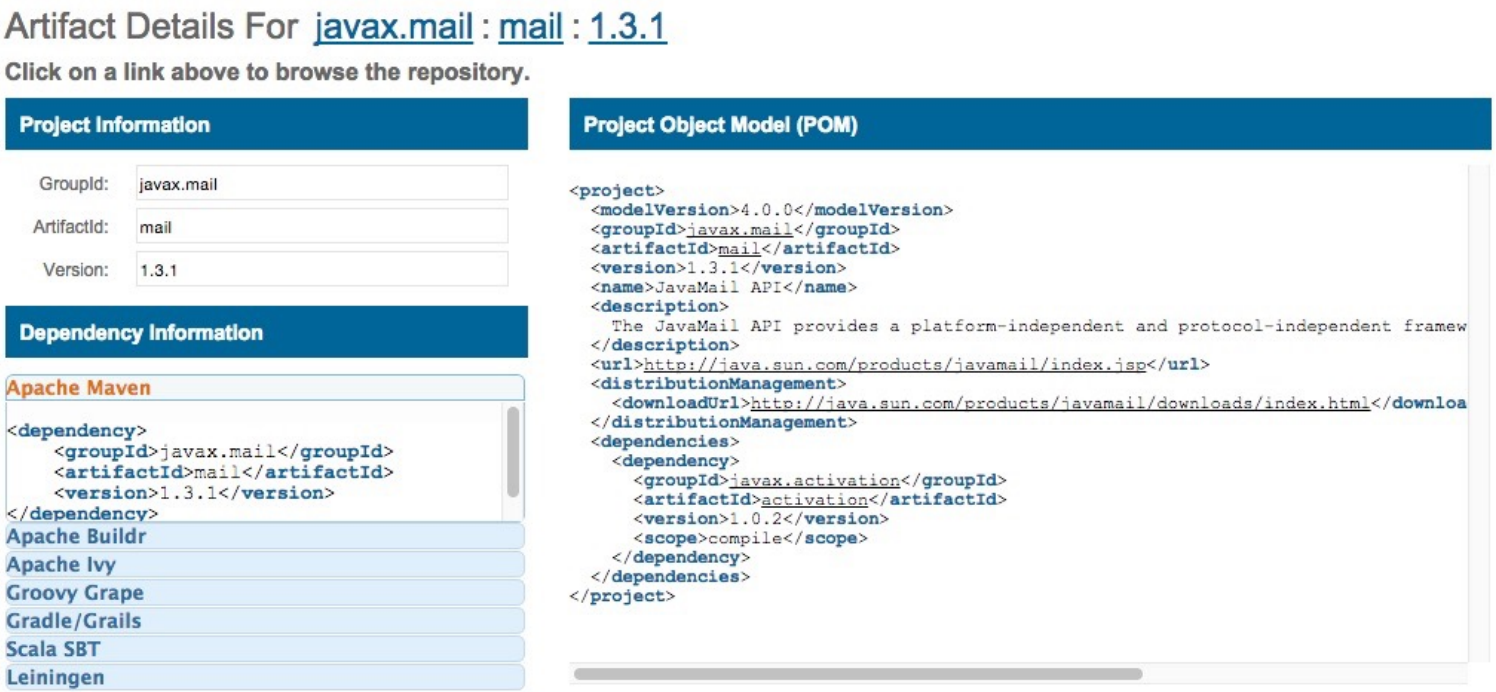
```
$ gradle compileJava

FAILURE: Build failed with an exception.

* What went wrong:
Could not resolve all dependencies for configuration
':compile'.
> Could not find mail.jar (javax.mail:mail:1.3.1).
  Searched in the following locations:
      https://repo1.maven.org/maven2/javax/mail/mail/1.3.1/
      mail-1.3.1.jar
```

Gradle

Resolution works, compilation fails



Artifact Details For javax.mail : mail : 1.3.1

Click on a link above to browse the repository.

Project Information

GroupId:	javax.mail
ArtifactId:	mail
Version:	1.3.1

Dependency Information

Apache Maven

```
<dependency>
    <groupId>javax.mail</groupId>
    <artifactId>mail</artifactId>
    <version>1.3.1</version>
</dependency>
```

Apache Buildr
Apache Ivy
Groovy Grape
Gradle/Grails
Scala SBT
Leiningen

Project Object Model (POM)

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>javax.mail</groupId>
  <artifactId>mail</artifactId>
  <version>1.3.1</version>
  <name>JavaMail API</name>
  <description>
    The JavaMail API provides a platform-independent and protocol-independent framew
  </description>
  <url>http://java.sun.com/products/javamail/index.jsp</url>
  <distributionManagement>
    <downloadUrl>http://java.sun.com/products/javamail/downloads/index.html</downloa
  </distributionManagement>
  <dependencies>
    <dependency>
      <groupId>javax.activation</groupId>
      <artifactId>activation</artifactId>
      <version>1.0.2</version>
      <scope>compile</scope>
    </dependency>
  </dependencies>
</project>
```

No JAR file available

Name	Last Modified	Size	SHA1 Checksum
mail-1.3.1.pom	21-Dec-2005	759 B	59b316aef0ffe41f4d8eb676747c7bcf8342e812

Gradle

Using dependency resolve rules

We never want our projects to use the version X of library Y.
How do we automatically pick version Z for consumers?

Using dependency resolve rules

We never want our projects to use the version X of library Y.
How do we automatically pick version Z for consumers?

We want to standardize on “good” versions for library Y.
How can we recommend this version for consumers?

Gradle

Using dependency resolve rules

We never want our projects to use the version X of library Y.
How do we automatically pick version Z for consumers?

We want to standardize on “good” versions for library Y.
How can we recommend this version for consumers?

Library Y got new coordinates. Whenever it is requested with old and new coordinates, how do we force the new one?

Gradle

Enforcing an artifact version

```
// Apply rule to specific configuration
configurations.all {

    // Iterate over resolved dependencies
    resolutionStrategy.eachDependency { DependencyResolveDetails
                                        details ->

        // Filter for a specific dependency
        if (details.requested.group == 'org.springframework') {

            // Force a different version
            details.useVersion '3.2.13.RELEASE'
        }
    }
}
```

Gradle

DEMO

Enforcing an artifact version of Spring framework

Recommending artifact versions

- Registry for commonly-used artifact versions
- Flexible version storage data structure
- Allow for user to look up default version
- Similar concept as Maven POM's `dependencyManagement` section

Gradle

Repurposing the artifact version identifier

Instead of a concrete version use the self-defined keyword `default`

```
dependencies {
    compile 'org.springframework:spring-core:default'
    compile 'org.springframework:spring-web:default'
}
```

Gradle

DEMO

Recommending an artifact version stored in a simple data structure

Using other data structures

What happens if the number of default versions exceeds 10 entries?

```
{
    "defaultVersions": [
        {
            "group": "org.springframework",
            "name": "spring-core",
            "version": "3.2.13.RELEASE"
        },
        ...
    ]
}
```

Gradle

Recommending artifact versions

Want a more fluent DSL that ties into Gradle's extensibility capabilities?

```
dependencies {
    compile defaultVersion('org.springframework', 'spring-core')
    compile defaultVersion('org.springframework', 'spring-web')
}
```

Gradle

DEMO

Extending Gradle's DSL for providing a default artifact version

Open Source plugins

Nebula Recommender Plugin (https://github.com/nebula-plugins/nebula-dependency-recommender)

Spring Dependency Management Plugin (https://github.com/spring-gradle-plugins/dependency-management-plugin)

Gradle

Using component selection rules

Reject a module selection based on custom rules e.g. group, name, version

```
configurations.all.resolutionStrategy {
    componentSelection.all { ComponentSelection selection ->

        // Filter selection candidate
        if(selection.candidate.group == 'com.google.collections') {

            // Reject selection with message
            selection.reject("The dependency is deprecated.")
        }
    }
}
```

Gradle

DEMO

Rejecting legacy module Google Collections with Guava

Replacing legacy modules

Module coordinates have changed but consumers still have access to new and legacy artifacts

```
dependencies {
    modules {
        module('com.google.collections:google-collections') {
            replacedBy('com.google.guava:guava')
        }
    }
}
```

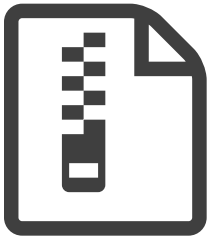
Gradle

DEMO

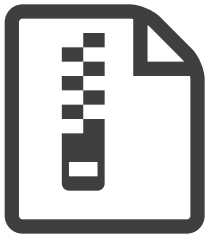
Replacing legacy module Google Collections with Guava

Using the artifact query API

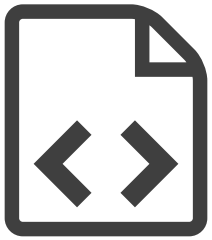
Query for raw artifacts resolved by a `Configuration`



Sources artifact



Javadoc artifact



Metadata artifact (ivy.xml, pom.xml)

Gradle

Resolving selected component IDs

```
// Use specific configuration
Configuration configuration = project.configurations.compile

// Determine resolution result
ResolutionResult result = configuration.incoming.resolutionResult

// Get all dependencies (resolved and unresolved)
Set<? extends DependencyResult> allDeps = result.allDependencies

// Filter resolved dependencies and collect component IDs
def componentIds = allDeps.collect { depResult ->
    if(dependencyResult instanceof ResolvedDependencyResult) {
        return it.selected.id
    }
}
```

Creating and executing query

```
// Get dependency handler
DependencyHandler handler = project.dependencies

// Create artifact resolution query
ArtifactResolutionQuery query =
                        handler.createArtifactResolutionQuery()
                        .forComponents(componentIds)
                        .withArtifacts(JvmLibrary,
                        SourcesArtifact, JavadocArtifact)

// Execute artifact resolution query
ArtifactResolutionResult result = query.execute()
```

Using resolved artifacts

```
// Iterate over resolved components
result.resolvedComponents.each { component ->
    // Get sources artifacts
    Set<ComponentArtifactsResult> sourceArtifacts =
                        component.getArtifacts(SourcesArtifact)

    sourceArtifacts.each {
        println "Source artifact for ${component.id}: ${it.file}"
    }

    // Get Javadoc artifacts
    Set<ComponentArtifactsResult> javadocArtifacts =
                        component.getArtifacts(JavadocArtifact)

    ...
}
```

Gradle

Metadata artifact query types

ArtifactResolutionQuery API

```
ArtifactResolutionQuery withArtifacts(Class<? extends Component>
componentType, Class<? extends Artifact>... artifactTypes)
```

Component type: `IvyModule`
Artifact type: `IvyDescriptorArtifact`

Component type: `MavenModule`
Artifact type: `MavenPomArtifact`

DEMO

Inspecting metadata of a resolved POM artifact

Solving other real-world Enterprise use cases

Common scenarios do not have a one-stop solution yet…

Locking dependency versions

My project depends on published modules of other projects. I always want to use the their latest version during development. What about reproducibility after releasing my project?

Gradle

Locking dependency versions

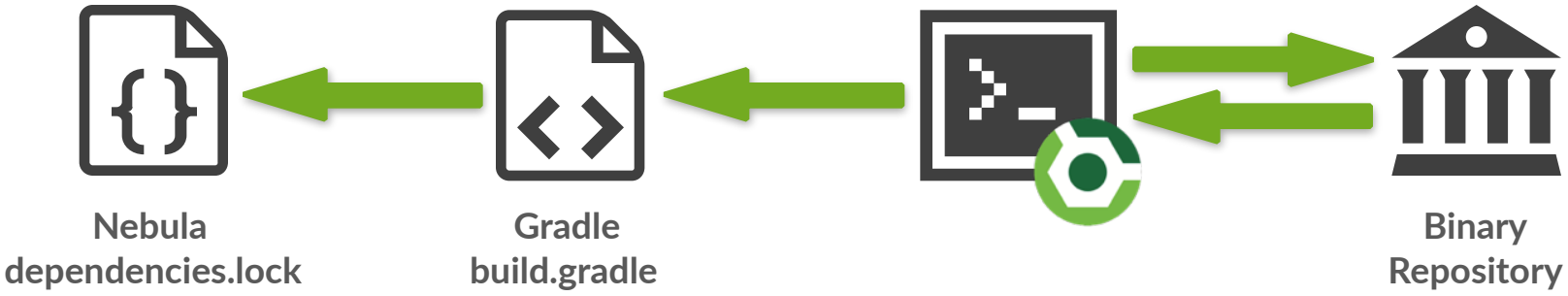
My project depends on published modules of other projects. I always want to use the their latest version during development. What about reproducibility after releasing my project?

Use version identifier latest.release during development. After releasing the project, pin to resolved versions.

Scenario: Enforce strong integration pressure

Gradle

Nebula dependency lock plugin



https://github.com/nebula-plugins/gradle-dependency-lock-plugin

Nebula dependency lock plugin

build.gradle

```
dependencies {
    compile 'com.google.guava:guava:14.+'
    compile 'commons-lang:commons-lang:latest.release'
}
```

Nebula dependency lock plugin

build.gradle

```
dependencies {
    compile 'com.google.guava:guava:14.+'
    compile 'commons-lang:commons-lang:latest.release'
}
```

dependencies.lock

```
{
  "com.google.guava:guava": {
     "locked": "14.0.1", "requested": "14.+"
  },
  "commons-lang:commons-lang": {
     "locked": "2.5", "requested": "latest.release"
  }
}
```

Nebula dependency lock plugin

build.gradle

```
dependencies {
    compile 'com.google.guava:guava:14.+'
    compile 'commons-lang:commons-lang:latest.release'
}
```

dependencies.lock

```
{
  "com.google.guava:guava": {
     "locked": "14.0.1", "requested": "14.+"
  },
  "commons-lang:commons-lang": {
     "locked": "2.5", "requested": "latest.release"
  }
}
```

Custom conflict resolution strategy

My team is transitioning from build tool X to Gradle. Developers are used to a different version conflict resolution strategy. How can we emulate the “known” strategy?

Gradle

Custom conflict resolution strategy

My team is transitioning from build tool X to Gradle. Developers are used to a different version conflict resolution strategy. How can we emulate the “known” strategy?

Gradle doesn’t support configuring a custom strategy. It’s always latest version but you can you force module versions.

Scenario: Smoother migration between tools

Gradle

Example: Simplified breadth-first

Forcing top-level dependency versions (the ones declared in the build script)

```
configurations.all { config ->
    resolutionStrategy {
        config.allDependencies.each { dep ->
            force "$dep.group:$dep.name:$dep.version"
        }
    }
}
```

Possible, but better to train the team “the Gradle way”!

Switching between binary & project dependencies

We work on multiple detached projects at the same time.
Every change to one of the projects needs to be published.
To simplify my work I want to use project dependencies.

Switching between binary & project dependencies

We work on multiple detached projects at the same time.
Every change to one of the projects needs to be published.
To simplify my work I want to use project dependencies.

Three options here: Home-grown Gradle code,
Prezi Pride or dependency substitution rules in Grade 2.5.

Scenario: Flexible dependency definitions

Open Source approaches

Elastic Deps

(https://github.com/pniederw/elastic-deps)

```
dependencies {
    compile elastic('com.example:example:1.0', 'example')
}
```

Prezi Pride (https://github.com/prezi/pride)


```
dynamicDependencies {
    compile group: 'com.example', name: 'example', version: '1.0'
}
```

Dependency substitution rules

Replace a project dependency with an external dependency and vice versa

```
configurations.all {
    resolutionStrategy {
        dependencySubstitution {
            substitute module('com.example:my-module')
                       with project(':project1')
        }
    }
}
```

Gradle 2.5

Gradle

Thank You!

Please ask questions…

https://www.github.com/bmuschko

@bmuschko