Building with

Pourquoi un outil de build?

- Compiler, mais aussi…
- Copier des ressources
- Générer une archive
- Générer la documentation
- Exécuter les tests
- Générer des rapports de qualité
- Déployer
- etc.

Et mon IDE ?

- L'IDE peut faire quelques-unes de ces tâches seulement
- Configuration manuelle nécessaire
- Pas scriptable
- Tous les développeurs n'ont pas le même IDE

Pourquoi automatiser ?

- Gagner du temps
- Ne rien oublier, ne pas se tromper
- Intégration continue
- Déploiements automatiques

Ant

- Ant est un outil génial :
  - Plein de tâches disponibles
  - Facile d'écrire les siennes
  - Très souple
- Mais…
  - Chaque projet doit réécrire les mêmes tâches (compilation, copie, jar, tests, rapports de tests, etc.)
  - Chaque projet a sa propre manière d'être construit
  - Ant ne gère pas les dépendances aux librairies (Ivy peut être utilisé)

Ant: Exemple

```
<target name="clean">
    <delete dir="build/classes"/>
</target>

<target name="resources">
    <mkdir dir="build/classes"/>
    <copy todir="build/classes">
        <fileset dir="src">
            <exclude name="**/*.java"/>
        </fileset>
    </copy>
</target>

<target name="build" depends="resources">
    <javac destdir="build/classes" srcdir="src">
        <classpath refid="classpath"/>
    </javac>
</target>

<target name="war" depends="build">
    <zip destfile="build/AntDemo.war">
        <zipfileset dir="WebContent"/>
        <zipfileset dir="build/classes" prefix="WEB-INF/classes"/>
    </zip>
</target>
```

Maven

- Maven a apporté de bonnes idées :
  - plugins
  - convention over configuration
  - cycle standard
  - gestion des dépendances
- Mais…
  - l'implémentation est mauvaise
  - Verbeux
  - Rigide
  - Mal documenté

Maven : exemple

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.ninja-squad</groupId>
    <artifactId>maven-demo</artifactId>
    <version>0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <dependencies>
        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
            <version>13.0</version>
        </dependency>
    </dependencies>
</project>
```

Gradle

- Reprend les bonnes idées de Ant
- Reprend les bonnes idées de Maven
- Peut utiliser des tâches ant
- N'utilise pas XML, mais Groovy

Gradle : exemple

```
apply plugin : 'java'

group = 'com.ninja-squad'

repositories {
    mavenCentral()
}

dependencies {
    compile 'com.google.guava:guava:13.0'
    testCompile 'junit:junit:4.+'
}
```

Démo

Groovy

- Dynamic Java
- Properties
- Pas besoin de parenthèses
- Pas besoin de point-virgule
- Pas besoin de typage fort
- Closures
- Pratique pour définir des DSLs

build.gradle

- A la racine du projet
- Permet de configurer l'objet Project

Tâches

- Définit quelque chose à faire
- Plein de types de tâches prédéfinies, configurables
- Peut dépendre d'une ou plusieurs autres tâches

Phases d'exécution

- Deux phases :
  1. Configuration :
     - Quelles tâches existent
     - Quel est le graphe de dépendances
  2. Exécution :
     - En fonction des arguments, exécute les tâches adéquates

Exemple de tâche

```
task hello << {
    println 'Hello world'
}
```

Equivalent à

```
task hello
hello.doLast {
    println 'Hello world'
}
```


Equivalent à

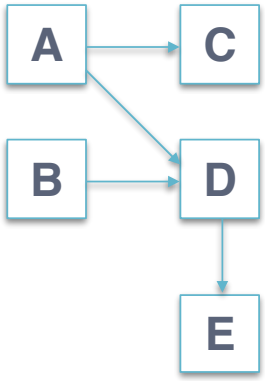
```
task hello
hello.doLast({
    println 'Hello world'
});
```

Equivalent à

```
project.task("hello4");
project.tasks.getByName("hello4").doLast({
    System.out.println("Hello world");
});
```


Dépendances entre tâches

- Les tâches constituent un DAG
  - Directed Acyclic Graph



Dépendances : exemple

```
task hello << {
    println 'Hello'
}

task world(dependsOn: hello) << {
    println 'World'
}
```


Equivalent à

```
task world(dependsOn: 'hello') << {
    println 'World'
}

task hello << {
    println 'Hello'
}
```

Apostrophes !

Equivalent à

```
task world {
    dependsOn 'hello'
    doLast {
        println 'World'
    }
}

task hello << {
    println 'Hello'
}
```

Que se passe-t-il si je met le println directement dans la tâche, sans doLast ?

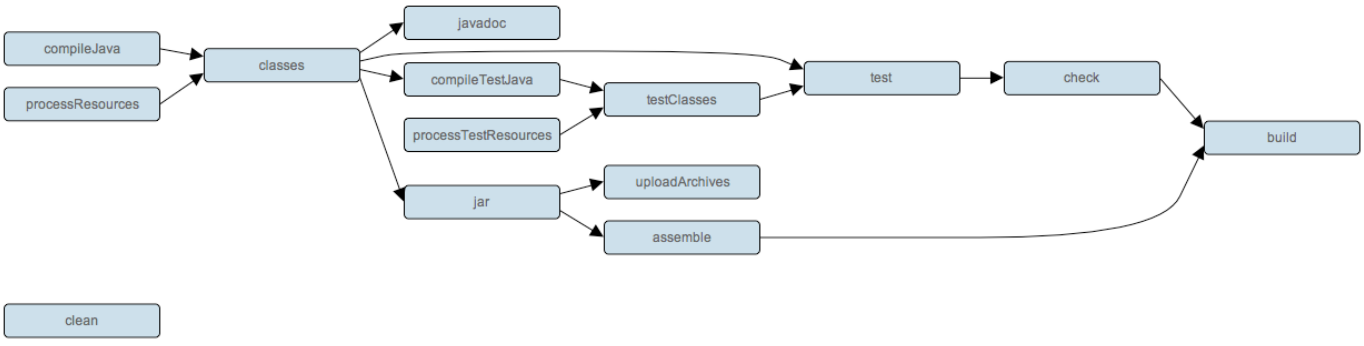
Plugins

- Ajoutent des tâches au projet
- Se basent sur des conventions
- Permettent de configurer les tâches ajoutées
- On peut ajouter des tâches additionnelles et des dépendances

Exemple: plugin Java

```
apply plugin : 'java'
```


Effet de cette unique ligne



Conventions

```
MonProjet
|   build.gradle
+---src
|   +---main
|   |   +---java
|   |   \---resources
|   |
|   \---test
|       +---java
|       \---resources
\---build
```

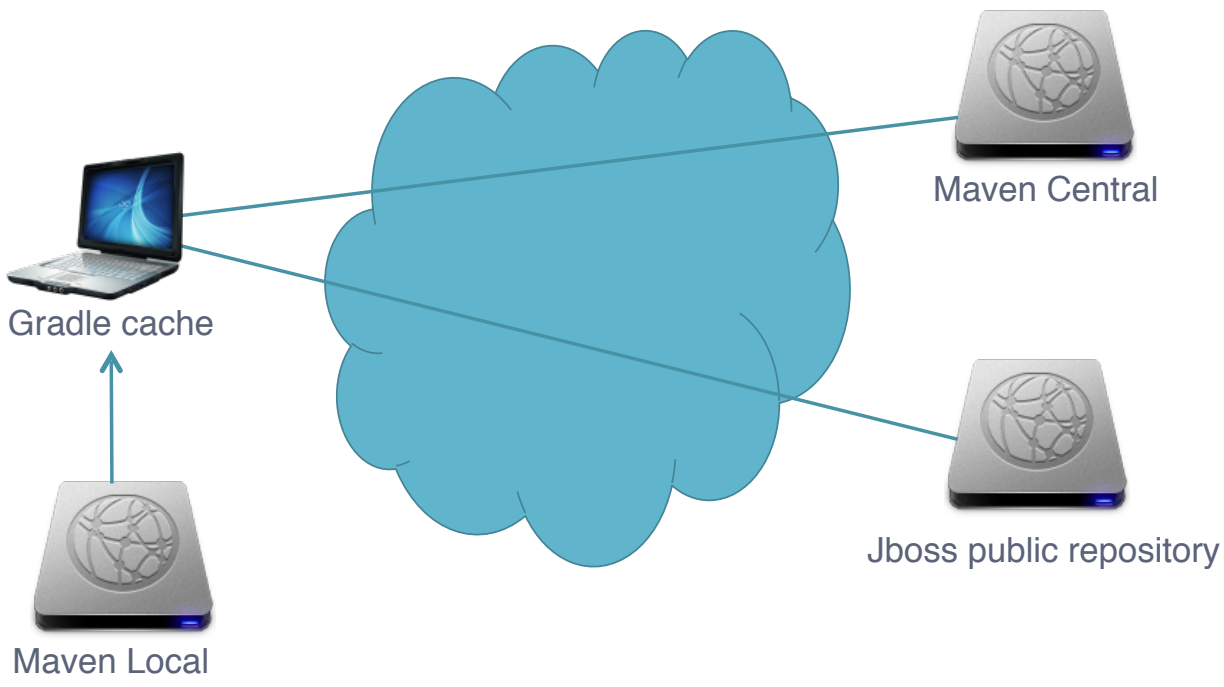

Intérêt de gradle

- Télécharger les dépendances directes
- Télécharger les dépendances transitives
- Gérer les conflits

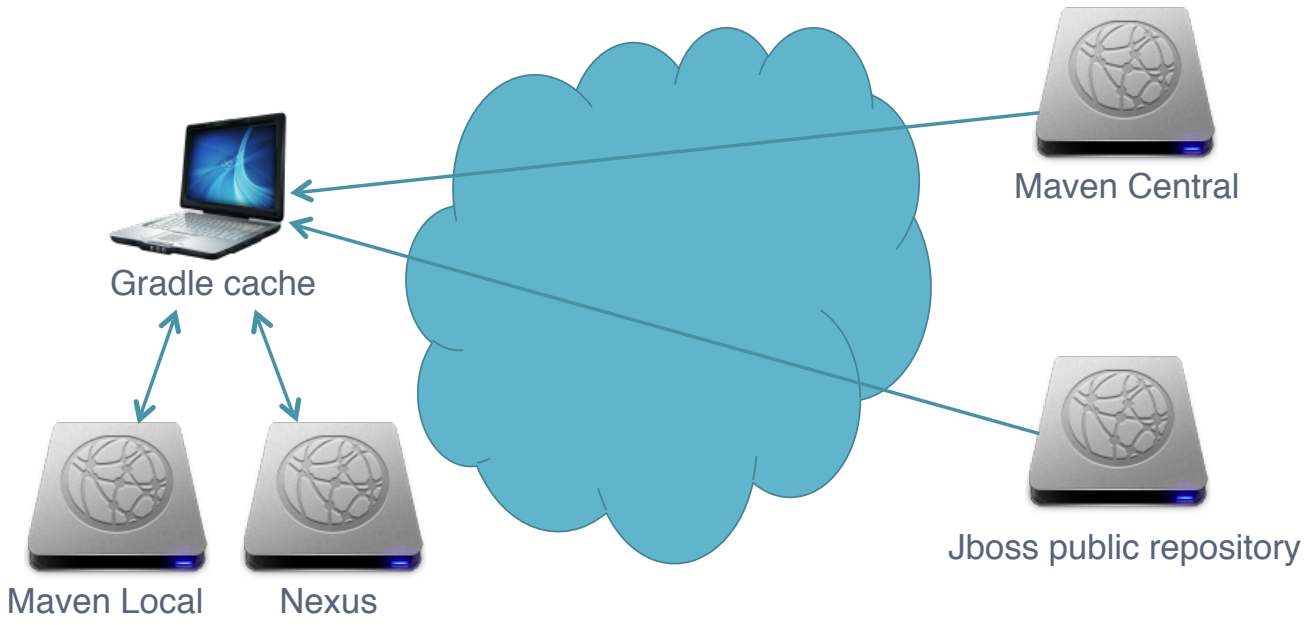
Mais aussi…

- Faire de votre projet une librairie
- La publier pour qu'elle puisse aussi être utilisée par d'autres

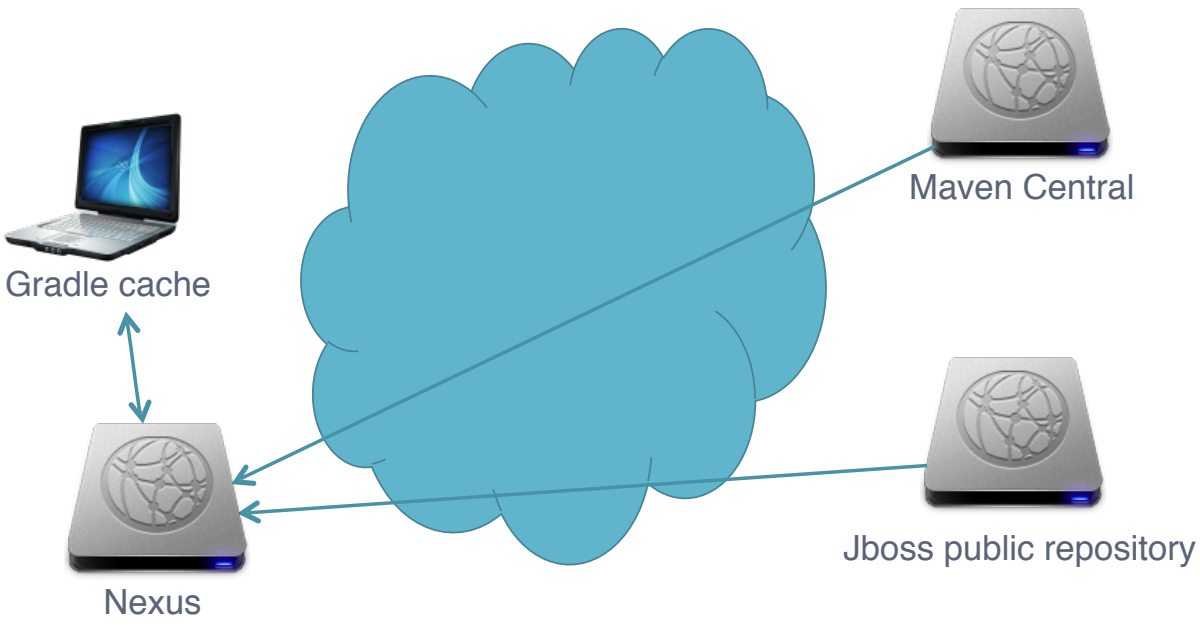
Repositories



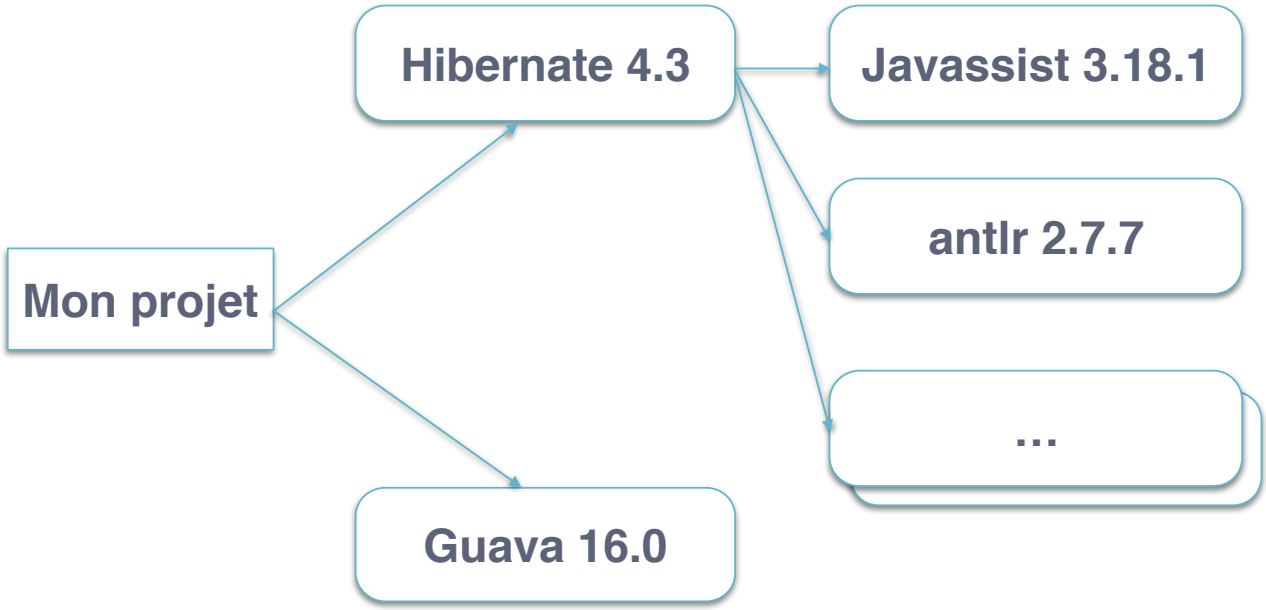
Déployer ses artefacts



Miroir / Proxy



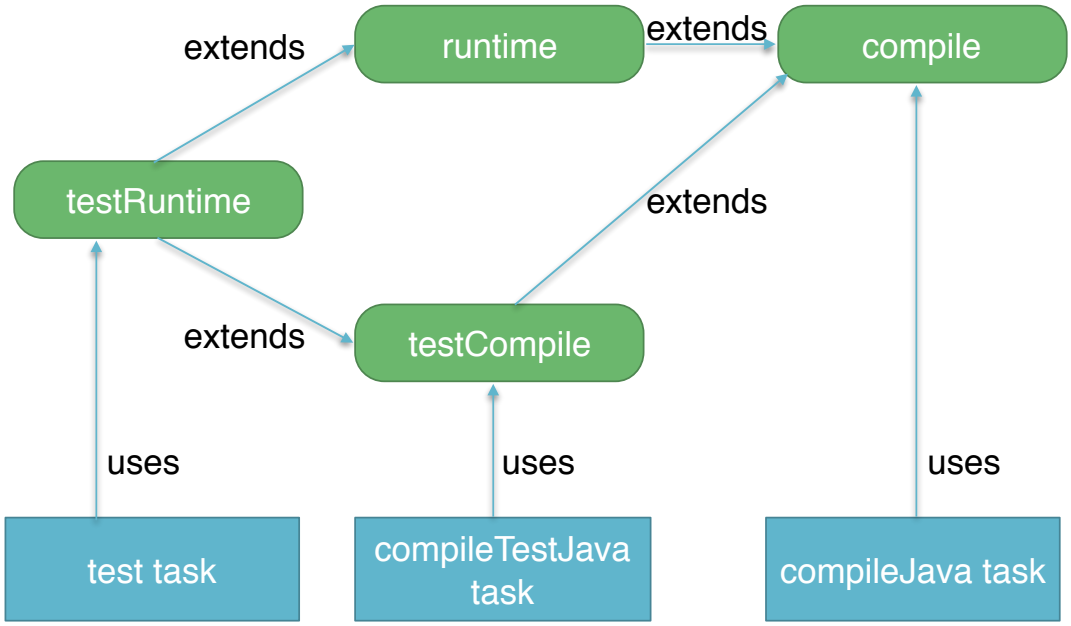
Dépendances



Configurations

- compile
- runtime
- testCompile
- testRuntime

Héritage des configurations



Déclarer des dépendances

```
apply plugin : 'java'

repositories {
    mavenCentral()
}

dependencies {
    compile 'com.google.guava:guava:13.0'
    testCompile 'junit:junit:4.+'
}
```

Le wrapper

- Problème : contribuer à un projet. Le projet utilise gradle. Mais quelle version ?
- Problème : vous devez corriger un bug dans une version qui date de 2 ans. Le projet utilisait gradle, mais quelle version ?
- Problème : vous contribuez à plusieurs projets utilisant une version différente de gradle

- Solution : le wrapper

Le wrapper

- Un jar : 50 KBs
- Un fichier properties
- un fichier gradlew.bat (pour Windows)
- un fichier gradlew (pour Unix, MacOS)

- Ces fichiers sont sauvegardés avec les sources, dans le système de gestion des sources

Le wrapper: fonctionnement

- Vérifie si la distribution de gradle en cache est identique à celle spécifiée dans le fichier properties
- Sinon, utilise le jar pour télécharger la distribution et l'installe dans le cache
- lance le gradle installé dans le cache