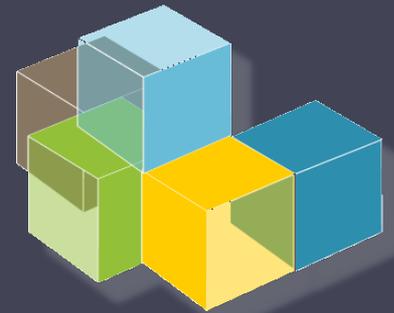


Universidad de Oviedo



Escuela de
Ingeniería
Informática



ARQUITECTURA
DEL SOFTWARE

Arquitectura del Software

Lab. 05

Automatización de la construcción:

Maven, Gradle, npm,...

2021 - 22

José Emilio Labra Gayo
Pablo González
Irene Cid
Hugo Lebrede

Construcción de Software

- Tareas
 - Compilación
 - De código fuente a código binario
 - Empaquetado
 - Gestión de dependencias e integración
 - También llamado enlace (linking)
 - Ejecución de pruebas
 - Despliegue
 - Crear documentación/*release notes*

Automatización de la construcción

- Automatizar tareas de construcción
- Objetivos:
 - Evitar errores (minimizar "*malas construcciones*")
 - Eliminar tareas redundantes y repetitivas
 - Gestionar complejidad
 - Mejorar calidad de producto
 - Tener un histórico de construcciones y *releases*
 - Integración continua
 - Ahorro de tiempo y dinero

Herramientas de automatización

- Makefile (mundo C)
- Ant (Java)
- Maven (Java)
- SBT (Scala, lenguajes JVM)
- Gradle (Groovy, lenguajes JVM)
- rake (Ruby)
- npm (Node.js)
- etc.

Maven - Introducción

Herramienta de automatización de construcción **en Java**

- Predefine cómo construir el software
- Describe dependencias del software

Inicialmente considera las siguientes fases de construcción cíclicas:

`validate, compile, test, package, integration-test, verify, install, deploy`

Dos fases independientes:

`clean, site`

Se pueden añadir más fases configuradas (goal)

Maven - Configuración

Configuración: fichero XML (Project Object Model)

`pom.xml`

Identificación de package :

- 3 coordenadas: groupId, artefactId, versión
- Tanto para identificar un package de dependencia como para generación de nuestro proyecto

Maven -Configuración

Sintaxis XML. Super POM. Herencia y Agregación Estructura básica

- Información del proyecto
 - groupId, artifactId, versión, packaging, name, description
- Parent
- Properties
- Dependencias
- Build
- Profiles

```

<parent>
  <groupId>org.uniovi.maven</groupId>
  <artifactId>maven-parent-example</artifactId>
  <version>2020</version>
  <relativePath>../pom/maven/pom.xml</relativePath>
</parent>
<artifactId>maven-child</artifactId>
<version>2020</version>
<packaging>pom</packaging>
<name>Apache Maven</name>
<description>Maven is a software build management and </description>
<url>https://maven.apache.org/ref/${project.version}/</url>
<properties>
  <maven.version>3.0.5</maven.version>
  <maven.compiler.source>1.8</maven.compiler.source>
</properties>
<modules>
  <module>maven-model</module>
  <module>maven-core</module>
</modules>
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.apache.maven</groupId>
      <artifactId>maven-model</artifactId>
      <version>${project.version}</version>
    </dependency>
  </dependencies>
</dependencyManagement>
<build>
  <pluginManagement>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-source-plugin</artifactId>
        <version>3.2.1</version>
      </plugin>
    </plugins>
  </pluginManagement>
</build>
<profiles>
<profile>
  <id>maven-repo-local</id>
  <activation>
    <property>
      <name>maven.repo.local</name>
    </property>
  </activation>
</profile>
</profiles>
</build>
</plugins>

```

Maven - Dependencias

Almacenes de artefactos - repositorios

Guardan diferentes tipo de artefactos

Ficheros JAR, EAR, WAR, ZIP, plugins, etc.

Todas las interacciones a través del repositorio

Sin caminos relativos

Compartir módulos entre equipos de desarrollo

Configuración repositorios **local**: settings.xml

Configuración repositorios **global**: pom.xml

Maven - Dependencias

Gestión automática de dependencias

Las dependencias son descargadas

Alojadas en repositorio local

Pueden crearse repositorios intermedios (proxies)

Ejemplo: artefactos comunes de una empresa

Transitividad

B depende de C

A depende de B \Rightarrow C también se descarga

Maven -Versionado

Identificación de proyecto

G A V (Grupo, artefacto, versión)

GroupId: Identificador de agrupamiento

ArtefactoId: Nombre del proyecto

Versión: Formato {Mayor}.{Menor}.{Mantenimiento}

Se puede añadir "-SNAPSHOT" (en desarrollo)

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>es.uniovi.asw</groupId>
  <artifactId>Entrecine8</artifactId>
  <version>1.0</version>
</project>
```

Maven - Archetipos

- Plantilla de Estructura de directorios
 - Por defecto Maven utiliza una estructura convencional
 - `src/main`
 - `src/main/java`
 - `src/main/webapp`
 - `src/main/resources`
 - `src/test/`
 - `src/test/java`
 - `src/test/resources`
 - Otros archetypes:
 - `maven-archetype-webapp`,
 - `maven-archetype-j2ee-simple`

Maven - Ejecución

Diferentes fases del ciclo de desarrollo

validate

compile

test

package

integration-test

install

deploy

clean

...

Invocación:

```
mvn clean
```

```
mvn compile
```

```
mvn clean compile
```

```
mvn compile install
```

```
...
```

Maven - Ejecución

- En cada una de estas fases tenemos objetivos (goals).
- Estos objetivos modifican como se ejecutan las fases para que satisfagan nuestros intereses.

```
<plugin>
  ...
  <executions>
    <execution>
      <phase>process-test-resources</phase>
      <goals>
        <goal>time</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Maven - Herencia

Múltiples módulos

Proyectos grandes pueden descomponerse

Cada proyecto crea un artefacto

Tiene su propio fichero pom.xml

El proyecto padre agrupa los módulos

```
<project>
  ...
  <packaging>pom</packaging>
  <modules>
    <module>extract</module>
    <module>game</module>
  </modules>
</project>
```

Maven-Ejecuciones conocidas

Otros arquetipos y ejecuciones

`archetype:generate` - Genera esqueleto de un proyecto

`eclipse:eclipse` - Genera proyecto eclipse

`site` - Genera sitio web del proyecto

`site:run` - Genera sitio web y arranca servidor

`javadoc:javadoc` - Generar documentación

`cobertura:cobertura` - Informe del código ejecutado en pruebas

`checkstyle:checkstyle` - Chequear el estilo de codificación

Gradle - Introducción

- Diseñado inicialmente para proyectos **basados en Java**, está basado en Groovy (también ahora en Kotlin) + DSL (Domain Specific Language) y resulta más fácil que Maven para crear tareas necesarias en la construcción
- Puede utilizarse para otros lenguajes: Kotlin, C++



Gradle -Conceptos básicos

- **Proyecto:** Es algo que construimos (por ejemplo ficheros jar) o que hacemos (desplegar nuestra aplicación en producción)
 - Ej creación proyecto
 - \$ gradle init
 - Task :wrapper
 - Select type of project to generate:
 - 1: basic
 - 2: application
 - 3: library
 - 4: Gradle plugin Enter selection (default: basic) [1..4]
 - Select build script DSL:
 - 1: Groovy
 - 2: Kotlin
- **Tarea:** Es una unidad atómica que se realiza durante la construcción (por ejemplo compilar nuestro proyecto o lanzar tests)



Gradle - Tareas

- Ventaja frente a Maven: Un sistema sencillo para codificar tareas y reutilizarlas
- Scripts se salvan en el build.gradle.
- El siguiente ejemplo define una tarea llamada “hello” el cual se usa para imprimir por pantalla “ASW”

```
task hello {  
    doLast {  
        println 'ASW'  
    }  
}
```

- Ejecución:

```
C:\> gradle -q hello
```



Gradle - Tareas

- Añadir dependencias a las tareas: Una tarea solo se ejecutará cuando se acabe de ejecutar de la que dependa

```
task taskX << {  
    println 'taskX' }  
task taskY(dependsOn: 'taskX') << {  
    println "taskY" }
```

```
task taskY << {  
    println 'taskY' }  
task taskX << {  
    println 'taskX' }  
taskY.dependsOn taskX
```

- Resultado Ejecución:

```
C:\> gradle -q taskY  
taskX  
taskY
```



Gradle - Dependencias

- Al igual que para Maven las librerías que se usarán se deberán descargar de algún repositorio (puede ser incluso un repositorio para Maven)

```
apply plugin: 'java'
repositories {
    mavenCentral()
}
dependencies {
    compile group: 'org.hibernate', name: 'hibernate-core', version: '3.6.7.Final'
    testCompile group: 'junit', name: 'junit', version: '4.+'
```

Gradle - Ciclo de vida

- Un proyecto -> un fichero *build.gradle*
- *BasePlugin* define pocas tareas que luego son implementadas por otro plugin específico (ej: Java Plugin)
 - clean
 - check
 - assemble
 - build



Gradle - Plugins

- **Plugin: Conjunto de tareas:**
 - Extienden el modelo básico Gradle
 - Configura el proyecto
 - Aplican configuraciones específicas.
- **Dos tipos:**
 - **Scripts:** Pueden ser aplicados de forma local o remota. `apply from: 'other.gradle'`
 - **Binarios:** Identificados por un plugin id.

```
apply plugin: JavaPlugin
```

```
plugins {  
    id "com.jfrog.bintray" version  
    "0.4.1"}  
}
```



npm

- **Node.js Package Manager**
 - Creado inicialmente por Isaac Schlueter
 - Posteriormente empresa: Npm Inc.
- **3 cosas**
 1. Sitio web (<https://www.npmjs.com/>)
Gestión de usuarios y organizaciones
 2. Almacén de software
Paquetes públicos/privados
 3. Aplicación en línea de comandos
Gestión tareas y dependencias
Fichero configuración: package.json



Configuración npm: package.json

- Fichero configuración: package.json
 - npm init crea un esqueleto simple
 - Campos:

```
{
  "name":           "...obligatorio...",
  "version":        "...obligatorio...",
  "description":    "...opcional...",
  "keywords":       "...",
  "repository":     {... },
  "author":         "...",
  "license":        "...",
  "bugs":           {...},
  "homepage":       "http://. . .",
  "main":           "index.js",
  "devDependencies": { ... },
  "dependencies":   { ... }
  "scripts":        { "test": " ... " },
  "bin":            {...},
}
```

Nota: Yeoman proporciona esqueletos completos



Paquetes npm

Almacén: <http://npmjs.org>

Instalación de paquetes:

2 opciones:

Local

```
npm install <packageName> --save (--save-dev)
```

Descarga los contenidos de <packageName> en node_modules

Global

```
npm install -g <packageName>
```

Guarda dependencia en the package.json

Solo para desarrollo



Dependencias npm

Gestión dependencias

Paquetes locales son guardados en `node_modules`

Acceso a través de: `require('...')`

Paquetes Global (instalados con opción `--global`)

Guardados en `/usr/local/npm` (en Linux)

Paquetes *Scoped* se marcan con `@`

Para usar un módulo dentro del proyecto u otro módulo

```
var uc = require('upper-case');
```



Dependencias npm

- Ejemplo de uso:

```
const http = require('http');
const uc = require('upper-case');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(uc("Hello World!"));
  res.end();
}).listen(8080);
```

- Un módulo para npm:
 - Un fichero `package.json`
 - Código asociado



NPM - Package.json

- Campos de definición del proyecto:
 - name
 - version
 - Author
 - homepage
 - description
 - private
 - Keywords
 - Main
 - Bin
- Dependencias
 - Dependencias
 - devDependencies
 - peerDependencies
 - bundleDependencies



NPM - Package.json

- Elementos del proyecto :
 - Links
 - Licencia
 - Archivos:
 - package.json
 - README
 - CHANGES / CHANGELOG / HISTORY
 - LICENSE / LICENCE
 - NOTICE
 - Scripts
 - Ejecutar tareas: `npm run <<namespace:>> script_name`



NPM - Package.json

- Ejecución:
 - **Browser:** Si el modulo va a ser ejecutado en el lado del cliente en un navegador, se debe usar este campo en vez de main
 - “browser”: ”src/App.js,
 - **Main:** punto de entrada al programa cuando va a ser ejecutado por un interprete javascript
 - “main”: ”src/App.js,
 - Engines, cpu, os
 - Configuración target browser.
Elemento extremo [browserlist](#)



NPM - Package.json

- **Repository:** Especifica donde está el código almacenado

```
"repository": {  
  "type" : "git",  
  "url" : "https://github.com/npm/cli.git"  
}
```

```
"repository": {  
  "type" : "svn",  
  "url" : "https://v8.googlecode.com/svn/trunk/"  
}
```

NPM

- **config:** Usado para configurar los parámetros usados en los paquetes de scripts que persisten ante las actualizaciones.

```
{  
  "name" : "foo"  
  , "config" : { "port" : "8080" }  
}
```

- **Más elementos:** <https://docs.npmjs.com/files/package.json>

NPM - Reglas versiones

- Regla para los nombres:
 - 214 caracteres o menos.
 - No puede empezar por punto o guión bajo
 - Los nuevos paquetes no pueden tener letras mayúsculas en los nombres
 - El nombre formará parte de la URL, un argumento de la línea de commando y el nombre de un fichero. Por lo tanto el nombre no puede contener los caracteres no validos en URLs



NPM - Reglas versiones

- Versión del paquete: Debe ser parseable por [node-semver](#), que está empaqueta con npm a través de una dependencia
- Rangos: Conjunto de comparados que especifican versiones que satisfacen el rango.
 - Por ejemplo el comparados $\geq 1.2.7$ permitiría 1.2.7, 1.2.8, 2.5.3, y 1.3.9, pero no 1.2.6 o 1.1.0.
 - Más en <https://docs.npmjs.com/misc/semver>



NPM -Package-lock.json

- Es generado automáticamente cuando se actualizan los módulos del proyecto.
- Describe el árbol exacto de versiones usado, evitando que un usuario instale una versión más actual que cuando se probó. **Recuerda: el formato de versiones nos permite definir la más actual, versiones mayores a una determinada...**
- Permite ver claramente la resolución de dependencias final, optimizando el proceso.



Task Execution : Grup y Gulp

Ejecutar tareas propias de JavaScript:

- Comprimir imágenes
- Empaquetar los módulos que van a ser usado en un proyecto (webpack)
- Minimizar ficheros js y css
- Ejecutar test
- Transcompilar – babel.js

Estas tareas pueden ejecutarse directamente con npm o pueden usarse dos herramientas muy famosas: Gulp y/o Grunt

Task Execution : Grup y Gulp

- Grup:

- Escrito sobre NodeJS.
Módulo fs
- Instalar:

```
npm install -g grunt  
npm install -g grunt-cli
```

- Configuración package.json

```
{  
  "name": "ASW",  
  "version": "0.1.0",  
  "devDependencies": {  
    "grunt-contrib-jshint": "~0.10.0",  
    "grunt-contrib-nodeunit": "~0.4.1",  
    "grunt-contrib-uglify": "~0.5.0"  
  }  
}
```

- Gulp:

- Escrito sobre NodeJS:
módulo stream
- Instalar:

```
npm install --save-dev gulp  
npm install -g gulp-cli
```

- Crea un gulpfile.js

```
function defaultTask(cb) {  
  // tareas  
  cb();  
}  
exports.default = defaultTask
```

Ejemplos

Wrapper

```
module.exports = function(grunt) {  
  // CONFIGURE GRUNT  
  grunt.initConfig({  
    (pkg.name)  
    pkg: grunt.file.readJSON('package.json'),  
  });  
  grunt.loadNpmTasks('grunt-contrib-uglify');  
  grunt.registerTask('default', ['uglify']);  
};
```

Wrapper

```
gulp.task("jpgs, function()  
{ return gulp.src('src/images/*.jpg')  
  .pipe(imagemin({ progressive: true })))  
  .pipe(gulp.dest('optimized_images')); });
```

Otro Ejemplo

<https://github.com/pglez82/npm-tutorial>

Fin