Maven

Víctor Herrero Cazurro

Temario

1. Introducción.
2. Instalación.
3. Consola Maven.
4. Arquetipos
5. Plugin Eclipse.
6. Características.
7. Estructura de un proyecto.
8. Maven Project Object Model (Pom.xml).
9. Dependencias.
10. Ciclo de Vida (Fases).
11. Plugins.
12. Templates y Arquetipos.

Introducción

- Maven es un Project Management Framework, esto es, un framework de gestión de proyectos de software.
- Creada por Jason van Zyl en 2002.
- Estuvo integrado inicialmente dentro del proyecto Jakarta.
- Ahora ya es un proyecto de nivel superior de la Apache Software Foundation.

Introducción

- Maven esta Basado en POM (project object model). Cada proyecto tiene la información para su ciclo de vida en el descriptor xml (por defecto el fichero pom.xml)
- Con Maven vamos a poder:
  - compilar,
  - empaquetar,
  - generar documentación,
  - pasar los test,
  - preparar las builds, ...

Introducción

- Maven es un sistema de estándares, un repositorio, y un software usado para manejar y describir proyectos.
- Define un ciclo de vida estándar para la construcción, prueba, y despliegue de componentes del proyecto.
- Proporciona un marco que permita la reutilización fácil de la lógica común de la estructura para todos los proyectos que siguen los estándares Maven.

Introducción

- Maven es una herramienta similar a ANT, pero con diferencias:
  - Maven se basa en patrones y en estándares. No es necesario aprender nuevas tareas para cada proyecto. Esto mejora el mantenimiento.
  - Con Ant se escriben las tareas y sus dependencias con otras tareas, mientras que con Maven hacemos una descripción del proyecto y Maven realiza las tareas.
  - Maven hace la gestión de librerías, incluso teniendo en cuenta las dependencias transitivas.

Introducción

- Maven y ANT, no son herramientas enfrentadas, sino que pueden ser complementarias. Usando cada una según nos interese.

Introducción

- Los objetivos perseguidos con Maven son:
  - Facilita el proceso de construcción.
  - Proporciona un sistema de construcción uniforme. Todos los proyectos Maven funcionan de la misma forma, por lo que el esfuerzo de aprendizaje sólo se hace una vez.
  - Proporciona información útil sobre el proyecto.
    - Lista de cambios desde el control de versiones.
    - Dependencias transitivas.
    - Informes de la ejecución de pruebas unitarias.
    - Ayuda a utilizar las “mejores practicas” de desarrollo.
    - Permite introducir nuevos servicios de forma sencilla.

Instalación de Maven

- Existen dos opciones de instalación de Maven:
  - Instalación de Maven como herramienta independiente
  - Instalación del Maven embebido en el plugin del IDE.
- Lo recomendable es siempre la primera opción, para controlar la versión instalada.

Instalación de Maven

- Pagina de descarga de Maven.

  http://maven.apache.org/download.html

- Lo que se descarga es un comprimido que habrá que descomprimir en cualquier ubicación del disco.

Instalación de Maven

- Habrá que añadir a la variable de entorno **PATH**, el directorio bin de la distribución.

```
SET PATH = %PATH%;"c:\program files\maven-2.2.1\bin"
```

- Comprobar que se ha definido **JAVA_HOME** apuntando a la ubicación de la distribución de java.

```
SET JAVA_HOME=“c:\java\jdk1.6.0_23”
```

Instalación de Maven

- Para comprobar que esta correctamente instalada, abrir una consola y ejecutar

```
mvn --version
```

Consola Maven

- Existe un comando de Maven que ayuda en la creación de un proyecto, siguiendo una serie de pasos

```
mvn archetype:generate
```

- Los pasos por los que nos llevará son:
  - Elección del arquetipo.
  - Versión del arquetipo.
  - GroupId.
  - ArtifactId.
  - Version.
  - Package.

Consola Maven

- Con dicho comando, lo que estamos utilizando son arquetipos, plantillas de proyectos ya definidas.
- Se puede ver un listado de arquetipos en

  http://docs.codehaus.org/display/MAVENUSER/Archetypes+List

- El 197, que es el por defecto, crea un proyecto java con una clase de código y de test de ejemplo.
- El 200, es un proyecto web.

Consola Maven

- También se puede crear con la siguiente instrucción.
- Crear un proyecto con maven

```
mvn archetype:create -DgroupId=com.empresa.solucion -
DartifactId=mi-aplicacion
```

- Esta instrucción, crea un proyecto en el directorio desde le que se ejecuta, llamado mi-aplicación con un paquete para el código fuente com.empresa.solucion.

Consola Maven

- Una vez creado el proyecto se puede convertir a proyecto eclipse con

```
mvn eclipse:eclipse
```

- Esta instrucción, añade al proyecto los ficheros de eclipse.
  - .classpath
  - .project
- Una vez creado se puede importar con eclipse.

Consola Maven

- Aunque se tenga instalado el plugin de eclipse para Maven, eclipse no lo reconoce como proyecto Maven.
- Habrá que convertirlo a proyecto Maven, con el menú Configure, en el cual una vez se ha instalado el plugin aparece convert to maven project.

Templates y Arquetipos

- Se pueden crear proyectos Maven usando los arquetipos.
- Estos arquetipos crean un proyecto base, con la estructura de Maven, y con las configuraciones (dependencias, plugin, …) necesarias para trabajar con dicho proyecto.
- Existen arquetipos de la mayoría de los frameworks (struts, jsf, spring, etc..).

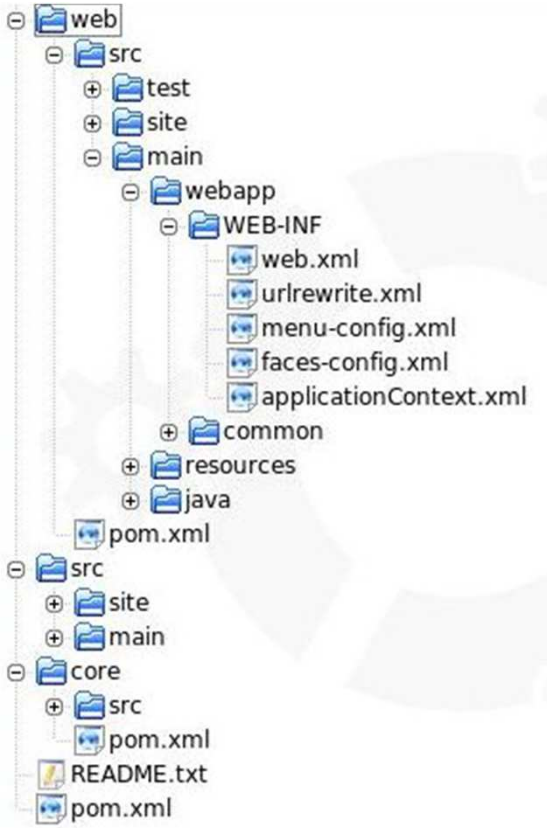
Templates y Arquetipos

- Ejemplo de empleo de un arquetipo appFuse para generar una aplicación con **jsf, spring e hibernate**:

```
mvn archetype:create
        -DarchetypeGroupId=org.appfuse.archetypes
        -DarchetypeArtifactId=appfuse-modular-jsf
        -DremoteRepositories=http://static.appfuse.org/releases
        -DarchetypeVersion=2.0.1
        -DgroupId=com.mycompany.app
        -DartifactId=myproject
```

Templates y Arquetipos

- El resultado de la creación de un proyecto con este arquetipo es



Templates y Arquetipos

- Se pueden generar arquetipos basados en proyectos ya existentes, ejecutando en el directorio del proyecto.

```
mvn archetype:create-from-project
```

- Que genera un arquetipo colocándolo en la carpeta

```
target\generated-sources\archetype
```

Templates y Arquetipos

- Dicho arquetipo deberá ser instalado en el repositorio para que sea accesible, con el mismo goal que se emplea con los proyectos.

```
mvn install
```

- Esta instrucción lo instala en el repositorio local.
- OJO, lo que se instala es el arquetipo generado no el proyecto por lo tanto habrá que situarse en

```
target\generated-sources\archetype
```

Plugin Eclipse para Maven

- Aunque existen varios plugin en eclipse para trabajar con Maven, disponibles desde el Market Place, emplearemos el Maven Integration for eclipse.
- Para instalarlo, buscarlo en el Market Place, instalar y seguir las instrucciones.
- Este plugin viene con una versión embebida de Maven (3.0.2), podemos emplear esa o podemos emplear la que hemos instalado.
- **Actualmente ya se incluye con Eclipse**

Plugin Eclipse para Maven

- La configuración del plugin se hace desde

```
Window->Preferences->Maven
```

  - **Instalations**: Configurar instalación de Maven que emplea el plugin.
  - **Archetypes**: Catálogos de arquetipos.
  - **Discovery**: Instalación de extensiones del plugin.
  - **User Settings**: Ubicación del settings de usuario y del repositorio, así como la reindexación del índice de las dependencias

Plugin Eclipse para Maven

- El plugin nos proporciona una serie de funcionalidades integradas en eclipse:
  - **Creación** de proyectos Maven empleando arquetipos.
  - **Conversión** de proyectos a Maven (Ojo, ya que la estructura no coincidirá con la de Maven y por tanto puede haber resultados inesperados).
  - **Ejecución** de comandos Maven.
  - **Editor** de pom.xml.

Características

- Creación sencilla y ágil de un nuevo proyecto.
- Estandarización de la estructura de un proyecto. El proyecto se describe en su totalidad en el fichero **pom.xml**, y existe una localización estándar para el código fuente, recursos, ficheros web...
- Potente mecanismo de gestión de las dependencias, y la resolución de dependencias transitivas.

Características

- Gestión simultánea de varios proyectos.
- Repositorio de librerías Open Source actualizado.
- Extensible, dispone de multitud de plugins y de la posibilidad de creación de otros que necesitemos.
- Acceso inmediato a nuevas funcionalidades requiriendo un mínimo esfuerzo.
- Integración con tareas ANT.

Características

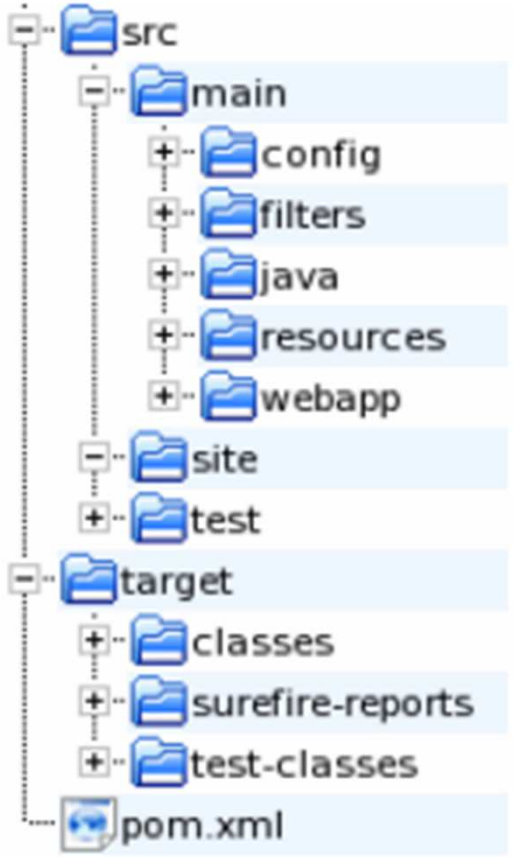
- Generación de diversos formatos de empaquetado de proyectos: WAR, EAR, JAR...
- Generación de un portal Web del proyecto. Incluyendo documentación del proyecto, informes del estado del proyecto, calidad del código.
- Gestión de releases y publicación. Integración con sistemas de gestión de versiones (como CVS o SVN).

Estructura de un proyecto

- Maven define una estructura de carpetas por defecto.
- La estructura puede ser modificada en el pom.xml.

Estructura de un proyecto

- Un proyecto estándar se vería así



Estructura de un proyecto

- **src/main/java**: Código Fuente
- **src/main/resources**: Otros recursos
- **src/main/webapp**: páginas, tags, etc.
- **src/main/webapp/WEB-INF**: Contiene el web.xml y otros ficheros de configuración.
- **src/test/java**: Código Fuente de pruebas.
- **src/test/resources**: Otros recursos para las pruebas.
- **site**: Carpeta con información para generar una página del proyecto.
- **target**: Carpeta donde se guardan los resultados.

pom.xml

- Fichero que define el proyecto Maven.
- Se encuentra siempre en la raiz del proyecto y contienen toda información necesaria para el ciclo de vida del proyecto:
  - dependencias,
  - plugins,
  - repositorios de donde obtener estos,
  - configuración de los informes…

pom.xml

- <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.ejemplo.maven</groupId>
- <artifactId>HolaMundo</artifactId>
- <version>0.0.1-SNAPSHOT</version>
- </project>

pom.xml

- Los elementos principales que aparecen en el POM son:
  - Project: Elemento de nivel superior en todos los pom.xml.
  - ModelVersion: Versión del modelo de objetos que emplea el POM.
  - groupId: Identificador único de la organización o del grupo que creó el proyecto.
  - artifactId: Identificador único del proyecto.

pom.xml

- packaging: Tipo de paquete del proyecto (por ejemplo, JAR, WAR, EAR, etc.). El valor predeterminado es JAR.
- versión: Indica la versión del proyecto.
- name: Indica el nombre del proyecto, se utiliza en la documentación generada de Maven.
- url: Indica donde encontrar el sitio del proyecto, se utiliza en la documentación generada de Maven.
- descripción: Muestra la descripción básica del proyecto. Se utiliza en la documentación generada de Maven.

Dependencias

- Son aquellos otros componentes (jar) que nuestro software necesita en algún momento del ciclo de vida.
- Con Maven las dependencias no acompañan al código fuente, se reflejan en el **pom.xml** y cuando se construye el proyecto se resuelven.
- Maven obtiene automáticamente las dependencias del proyecto, bien del repositorio local si ya esta o bien de un repositorio remoto.

Dependencias

- Las dependencias pueden necesitarse en distintos momentos del ciclo de vida del proyecto y Maven permite definir el ámbito o **scope** de las mismas, este puede ser:
  - **compile**: es el valor por defecto. Se utiliza en todos los casos (compilar, ejecutar, ...).
  - **test**: Sólo se utiliza para compilar o ejecutar los test.
  - **run-time**: no se utiliza para compilar, pero si es necesario para ejecutar.

Dependencias

- **provided**: también se utiliza en todos los casos, pero se espera que el jar sea suministrado por la JDK o el contenedor. Es decir, no se incluirá al empaquetar el proyecto, ni en el repositorio.
- **system**. es similar a provided, pero eres tu el que tiene que suministrar el jar. No se incluirá al empaquetar el proyecto, ni en el repositorio.

Dependencias

- La sintaxis para añadir dependencias será

```
<dependencies>
        <dependency>
                <groupId>junit</groupId>
                <artifactId>junit</artifactId>
                <version>3.8.1</version>
                <scope>test</scope>
        </dependency>
</dependencies>
```

Dependencias

- Las dependencias de resuelven de forma transitiva, por lo que las dependencias que tenga un jar del que depende nuestro proyecto, serán también añadidas como dependencias en el proyecto.
- Se pueden excluir dependencias, para controlar, que la versión empleada de un determinado jar, sea la que se desea, y no otra anterior o posterior.

Dependencias

- Para añadir una exclusión, habrá que incluir las etiquetas <exclusions><exclusion>, dentro de la dependencia que aporta el jar a excluir.

```
<dependency>
        …
        <exclusions>
                <exclusion>
                        <groupId>commons-logging</groupId>
                        <artifactId>
                                commons-logging
                        </artifactId>
                </exclusion>
        </exclusions>
</dependency>
```

Ciclo de vida

- Las partes del ciclo de vida de Maven, son conocidas como goals.
- Los principales son:
  - **validate**: Valida la estructura del proyecto y que todo lo necesario esta disponible.
  - **compile** : Genera los ficheros .class compilando los fuentes .java
  - **test**: Ejecuta los test automáticos de JUnit existentes, abortando el proceso si alguno de ellos falla.

Ciclo de vida

- **package**: Genera el fichero .jar con los .class compilados.
- **integration-test**: distribuye el paquete de la aplicación en un entorno de pruebas de integración, y lanza las pruebas.
- **verify**: pasa los criterios de calidad al proyecto.
- **install**: Copia el fichero .jar a un directorio de nuestro ordenador donde Maven deja todos los .jar. De esta forma esos .jar pueden utilizarse en otros proyectos Maven en el mismo ordenador.

Ciclo de vida

- **deploy**: Copia el fichero .jar a un servidor remoto, poniéndolo disponible para cualquier proyecto maven con acceso a ese servidor remoto.

Ciclo de vida

- Su ejecución será secuencial, cuando se ejecuta cualquiera de los comandos, maven irá verificando todas las fases del ciclo de vida desde la primera hasta la ejecutada, ejecutando sólo aquellas que no se hayan ejecutado previamente.
- La sintaxis para su ejecución será

```
mvn <comando>
```

- Por ejemplo para compilar

```
mvn compile
```

Ciclo de vida

- Existen algunos goals que están fuera del ciclo de vida:
  - **clean**: Elimina todo el contenido del directorio target. Después de este comando se puede comenzar un compilado desde cero.
  - **assembly:assembly**: Genera un fichero .zip con todo lo necesario para instalar nuestro programa java. Se debe configurar previamente en un fichero xml qué se debe incluir en ese zip.
  - **test-compile**: compila los test, sin ejecutarlos.

Ciclo de vida

- **site**: Genera un sitio web con la información de nuestro proyecto. Dicha información debe escribirse en el fichero pom.xml y ficheros .apt separados.
- **site-deploy**: Sube el sitio web al servidor que hayamos configurado.
- **eclipse:eclipse**: Añade los ficheros necesarios para que el proyecto sea considerado un proyecto por eclipse.

Herencia de pom

- Cuando hay una relación entre proyectos Maven, es decir cuando unos dependen de otros, típico escenario de proyecto dividido en módulos, es necesario automatizar la compilación de todos los módulos que forman parte del proyecto, para ello, se tiene la herencia de pom.
- Podemos crear un pom que relacione todos los módulos, e indicar en los módulos que tienen un pom padre.

Herencia de pom

- En el pom.xml del proyecto padre (que puede solo tener dicho pom.xml), aparecerá:

```
<packaging>pom</packaging>
<modules>
        <module>proyecto_hijo</module>
</modules>
```

- Y en los módulos hijos

```
<parent>
        <groupId>paquete_inicial</groupId>
        <artifactId>proyecto_padre</groupId>
        <version>1.0-SNAPSHOT</version>
</parent>
```

Herencia de pom

- Con esta configuración, cualquier configuración global que se realice en el padre, será compartida por todos los hijos, excepto que se sobrescriba.
- Además el ciclo de vida de todos los proyectos queda centralizado en el proyecto padre, por tanto cualquier goal ejecutado sobre el padre, se ejecutará sobre todos los hijos.

Herencia de pom

- El arquetipo para generar un proyecto padre, será.

```
ArchetypeGroupId = org.codehaus.mojo.archetypes
ArchetypeArtifactId = pom-root
```

- Este arquetipo únicamente genera un proyecto con un **pom.xml**.
- Donde se encuentra el principal elemento de configuración, es en la etiqueta <package> de dicho fichero, que tiene como valor pom.

Herencia de pom

- A priori, Maven supone, que los proyectos hijos (módulos), estarán dentro de la carpeta del proyecto padre, es decir en el mismo path que el **pom.xml** del padre, pero normalmente no será así, para establecer dicha consideración, se debe configurar el parámetro **<parent><relativePath>** haciendo referencia a la ubicación del proyecto padre.

Plugins

- Permiten extender la funcionalidad de Maven.
- Se añaden al pom.xml.
- Se puede indicar una configuración personalizada a través de la etiqueta

```
<configuration>
```

- Se puede restringir la ejecución a alguna fase del ciclo de vida.

```
<executions>
        <execution>
                <phase>integration-test</phase>
```

Plugins

- Se pueden crear pluggins personalizados mediante MOJO.
- Existen Plugins para compilar, testear, documentar, desplegar, ...

Plugin Compiler

- Permite configurar como realizar la Compilación de un proyecto.

```
<build><plugins><plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.3.2</version>
        <configuration>
                <source>1.6</source>
                <target>1.6</target>
        </configuration>
</plugin><plugins></build>
```

Plugin Javadoc

- Para añadir un plugin que crea los javadoc como parte del ciclo de vida del proyecto

```
<reporting><plugins><plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-javadoc-plugin</artifactId>
</plugin></plugins></reporting>
```

Plugin SCM

- Se puede encontrar documentación en

```
http://maven.apache.org/scm/plugins/
```

- Permite la conexión con sistemas de control de versiones

```
<scm>
        <connection>
                scm:svn:svn://server/test/trunk/Negocio
        </connection>
        <developerConnection>
                scm:svn:svn://server/test/trunk/Negocio
        </developerConnection>
</scm>
```

Plugin Release

- Permite publicar una **Release** en sistema gestor de versiones como parte del ciclo de vida del proyecto.

```
<build><plugins><plugin>
        <artifactId>maven-release-plugin</artifactId>
        <configuration>
                <password>${scm.password}</password>
                <tagBase>svn://server/test/trunk</tagBase>
        </configuration>
</plugin></plugins></build>
```

Plugin Jetty

- Plugin que permite controlar un servidro Jetty desde Maven.

```
https://cwiki.apache.org/WICKET/maven-jetty-plugin.html
```

- El puerto por defecto es el 8080, pero se puede establecer con

```
mvn -D jetty.port=9090  jetty:run-war
```

Plugin Jetty

- Datos generales del plugin

```
<plugin>
        <groupId>org.mortbay.jetty</groupId>
        <artifactId>maven-jetty-plugin</artifactId>
        <version>6.1.10</version>
        <configuration>
                <scanIntervalSeconds>10</scanIntervalSeconds>
                <stopKey>foo</stopKey>
                <stopPort>9999</stopPort>
        </configuration>
        …
</plugin>
```

Plugin Jetty

- Configuración para que se arranque en la fase de **pre-integration-test**

```
<plugin>
        …
        <executions>
                <execution>
                        <id>start-jetty</id>
                        <phase>pre-integration-test</phase>
                        <goals>
                                <goal>run</goal>
                        </goals>
                        <configuration>
                                <scanIntervalSeconds>
                                        0
                                </scanIntervalSeconds>
                        <daemon>true</daemon>
                        </configuration>
                </execution>
                …
        </executions>
</plugin>
```

Plugin Jetty

- Configuración para que se pare en la fase de **post-integration-test**

```
<plugin>
        …
        <executions>
                …
                <execution>
                        <id>stop-jetty</id>
                        <phase>post-integration-test</phase>
                        <goals>
                                <goal>stop</goal>
                        </goals>
                </execution>
        </executions>
</plugin>
```

Plugin Jboss

- Permite configurar un Jboss desde Maven

http://mojo.codehaus.org/jboss-maven-plugin/

- Permite arrancar/parar y desplegar/redesplegar una aplicación en un JBoss

```
<plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>jboss-maven-plugin</artifactId>
        <version>1.5.0</version>
        <configuration>
                <hostName>localhost</hostName>
                <port>8080</port>
                <fileNames>
                        <fileName>target/my-project.war</fileName>
                <fileNames>
        </configuration>
</plugin>
```

Plugin Jboss

- También se puede arrancar/Parar con

```
<plugin>
		<groupId>org.codehaus.mojo</groupId>
		<artifactId>jboss-maven-plugin</artifactId>
		<version>1.5.0</version>
		<configuration>
			<jbossHome>/usr/jboss-4.2.3.GA</jbossHome>
		</configuration>
</plugin>
```

Plugin Jboss

- Los goals a emplear serán
  - mvn jboss:deploy
  - mvn jboss:redeploy
  - mvn jboss:undeploy
  - mvn jboss:start
  - mvn jboss:stop

Plugin Surefire

- Permite ejecutar pruebas unitarias

http://maven.apache.org/plugins/maven-surefire-plugin/index.html

- La configuración

```
<plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>2.7.1</version>
        <configuration>
                <excludes>
                        <exclude>**/integracion/*.java</exclude>
                </excludes>
                <includes>
                        <include>**/unitarias/*.java</include>
                </includes>
        </configuration>
</plugin>
```

Plugin Failsafe

- Permite lanzar pruebas de integración

http://maven.apache.org/plugins/maven-failsafe-plugin/

- La configuración general

```
<plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-failsafe-plugin</artifactId>
        <version>2.8</version>
        <configuration>
                <excludes>
                        <exclude>**/unitarias/*.java</exclude>
                </excludes>
                <includes>
                        <include>**/integracion/*.java</include>
                </includes>
        </configuration>
        …
</plugin>
```

Plugin Failsafe

- La configuración para que se pase en la fase de pruebas de integración.

```
<plugin>
        …
        <executions>
                <execution>
                        <id>pasar test integracion</id>
                        <phase>integration-test</phase>
                        <goals>
                                <goal>integration-test</goal>
                        </goals>
                </execution>
                <execution>
                        <id>validar pruebas integracion</id>
                        <phase>verify</phase>
                        <goals>
                                <goal>verify</goal>
                        </goals>
                </execution>
        </executions>
</plugin>
```

Plugin Sonar

- Permite obtener mediciones de la Calidad del código.

```
http://mojo.codehaus.org/sonar-maven-plugin/plugin-info.html
```

- El plugin busca el servidor Sonar en

```
http://localhost:9000
```

- Sonar se descarga de

```
http://www.sonarsource.org/downloads/
```

Plugin Sonar

- La configuración

```
<plugin>
        <groupId>org.codehaus.sonar</groupId>
        <artifactId>sonar-maven-plugin</artifactId>
        <version>2.0</version>
</plugin>
```

Plugin Sonar

- La configuración de Sonar para emplear un sonar en remoto, pasa por modificar el fichero

```
.\conf\sonar.properties
```

- donde se establece la url, el puerto, configuración de la base de datos, …
- Además, habrá que configurar la conexión a Sonar (BD), a través de un **profile**.

Plugin Sonar

- Configuración del **Profile**.

```
<profile>
      <id>sonar</id>
      <activation>
            <activeByDefault>true</activeByDefault>
      </activation>
            <properties>
            <sonar.jdbc.url>
                  jdbc:mysql://localhost:3306/sonar
            </sonar.jdbc.url>
            <sonar.jdbc.driver>com.mysql.jdbc.Driver</sonar.jdbc.driver>
            <sonar.jdbc.username>sonar</sonar.jdbc.username>
            <sonar.jdbc.password>password</sonar.jdbc.password>
            <sonar.host.url>http://tudominio:8080</sonar.host.url>
            </properties>
      </profile>
 </profiles>
```

Plugins personalizados

- Aunque no es necesario, ya que un plugin Maven, es simplemente un conjunto de clases java, es recomendable crear un plugin a partir de un proyecto Maven.
- Existe un arquetipo para crear plugins de Maven.

```
ArchetypeArtifactId = maven-archetype-mojo
```

- Este arquetipo crea una clase que hereda de **org.apache.maven.plugin.AbstractMojo**, base para los plugin de Maven.

Plugins personalizados

- Este plugin creado por defecto simplemente escribe un fichero **touch.txt** en target.
- Es posible enviar parámetros al plugin desde Maven, la forma de hacerlo viene documentada en ese plugin de ejemplo, con el atributo de clase outputDirectory y los comentarios que le acompañan.

```
/** Location of the file.
     * @parameter expression="${project.build.directory}"
     * @required */
    private File outputDirectory;
```

Plugins personalizados

- Los posibles parámetros de Maven a emplear en los plugin se pueden obtener en

```
http://docs.codehaus.org/display/MAVENUSER/MavenPropertiesGuide
```

- También se pueden definir propiedades personalizadas en el pom.xml del proyecto que emplea el plugin, de la siguiente forma.

```
<properties>
        <texto.salida>
                Este texto se incluira en el fichero de salida
        </texto.salida>
</properties>
```

Plugins personalizados

- Una vez definido el plugin, habrá que instalarlo en el repositorio para tener acceso a el.

```
mvn install
```

- Una vez instalado se podrá hacer referencia a el desde otro proyecto Maven con

```
<plugins>
        <plugin>
                <groupId>com.ejemplo</groupId>
                <artifactId>MiPlugin</artifactId>
                <version>0.0.1-SNAPSHOT</version>
        </plugin>
</plugins>
```

Plugins personalizados

- Y ejecutando el goal **touch**

```
mvn com.ejemplo:MiPlugin:0.0.1-SNAPSHOT:touch
```

- El goal se define como parámetro en la cabecera de la clase del plugin

```
/**
 * Goal which touches a timestamp file.
 * @goal touch
 * @phase compile
 */
```

Plugins personalizados

- También se puede configurar el plugin para que se lance en una fase del ciclo de vida del proyecto.

```
<plugins><plugin>
    <groupId>com.ejemplo</groupId>
    <artifactId>MiPlugin</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <executions><execution>
        <phase>package</phase>
        <goals>
            <goal>touch</goal>
        </goals>
    </execution></executions>
</plugin></plugins>
```

Profiles

- Los perfiles nos permiten definir propiedades concretas para la construcción de los proyectos en distintas situaciones.
  - Producción.
  - Pruebas.
  - Desarrollo.

Profiles

- La configuración pasa por añadir al pom.xml
- <profiles>
- <profile>
- <id>build-release</id>
- <properties>
- <maven.compiler.debug>false</maven.compiler.debug>
- <maven.compiler.optimize>true</maven.compiler.optimize>
- </properties>
- </profile>
- </profiles>
- Para seleccionar el perfil a emplear, habrá que indicarlo con –P
- mvn package -P build-release

Repositorios

- Existen varios repositorios
  - Archiva. (Gratuito)
  - Nexus. (De pago)
  - Artifactory. (De pago barato)
- Instalamos Archiva, para ello descargamos la versión stand-alone del sitio Apache
- http://archiva.apache.org/

Repositorios

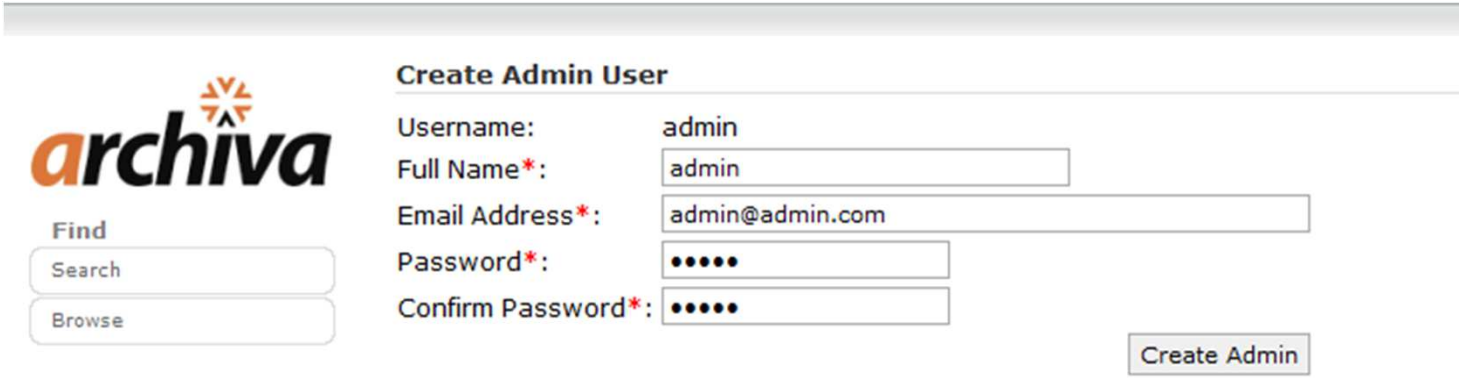
- Se descarga un zip, en cuyo interior encontramos un fichero bat en el directorio bin.
- ./bin/archiva.bat
- Se ejecuta el comando
- archiva console
- Que arranca un jetty en el puerto 8080
- http://localhost:8080/archiva/security/addadmin.action

Repositorios

- Por defecto Archiva incluye una base de datos derby, para modificarla y emplear otra, habrá que configurar el nodo
- <!-- Archiva database -->
- Del fichero ./conf/jetty.xml

Repositorios

- Una vez se accede a Archiva, se ha de dar de alta un administrador.



- Con este usuario habrá que generar los usuarios con permisos de despliegue, es decir los que tengan un rol Repository Manager.

Repositorios

- Para configurar el repositorio intermedio (de empresa), se ha de añadir al fichero settings.xml lo siguiente.
- <settings><servers><server>
- <id>archiva.internal</id>
- <username>despliegue</username>
- <password>despliegue1</password>
- </server><server>
- <id>archiva.snapshots</id>
- <username>despliegue</username>
- <password>despliegue1</password>
- </server></servers></settings>

Repositorios

- <profiles><profile>
- <id>Repositorios Locales</id>
- <activation>
- <activeByDefault>true</activeByDefault>
- </activation>
- <repositories><repository>
- <id>archiva.internal</id>
- <url>http://localhost:8080/archiva/repository/internal/</url>
- <releases>
- <enabled>true</enabled>
- </releases>
- <snapshots>
- <enabled>false</enabled>
- </snapshots>

Repositorios

- </repository><repository>
- <id>archiva.snapshots</id>
- <url>http://localhost:8080/archiva/repository/snapshots/</url>
- <releases>
- <enabled>false</enabled>
- </releases>
- <snapshots>
- <enabled>true</enabled>
- </snapshots>
- </repository></repositories>
- </profile></profiles>

Repositorios

- Para que el goal deploy instale el artefacto en Archiva, habrá que añadir en el pom.xml
- <distributionManagement>
- <repository>
- <id>archiva.internal</id>
- <name>Internal Release Repository</name>
- <url>http://localhost:8080/archiva/repository/internal/</url>
- </repository>
- <snapshotRepository>
- <id>archiva.snapshots</id>
- <name>Internal Snapshot Repository</name>
- <url>http://localhost:8080/archiva/repository/snapshots/</url>
- </snapshotRepository>
- </distributionManagement>

Repositorios

- Dentro Archiva, se puede configurar como se conecta a internet, configurando un proxy. Esto se realiza en la opción Network proxies y posteriormente asignando dicha conexión a los Proxy connectors.

Fin