Construcción de proyectos con Maven

Estándar de facto en el ecosistema Java para la construcción de proyectos.

por Ivan Ruiz Rube

Agenda de este tema

Conceptos básicos

Maven: definición, características e instalación.

Ciclo de vida y comandos

Fases del ciclo y comandos principales.

Estructura y configuración

POM, coordenadas y gestión de dependencias.

Construcción avanzada

Plugins, perfiles, módulos e integración CI/CD.

Problemas comunes en la construcción de proyectos

Sin herramientas de automatización como Maven, los desarrolladores se enfrentan a numerosos desafíos:

Estructura inconsistente de directorios

Cada desarrollador podría crear estructuras diferentes, dificultando la colaboración y el mantenimiento del código base entre equipos.

Compilación y empaquetado manual

Proceso propenso a errores que consume tiempo valioso, especialmente en proyectos grandes donde la compilación requiere múltiples pasos secuenciales.

Gestión caótica de dependencias

Descargar y actualizar manualmente JARs de terceros causa conflictos de versiones, incompatibilidades y problemas de "JAR hell" en entornos de producción.

Meta-información inconsistente

Sin un sistema estandarizado, la información del proyecto (versión, autores, licencias) puede estar desactualizada o inconsistente entre archivos de código y documentación.

Documentación técnica desactualizada

Generar y mantener la documentación del proyecto se vuelve una tarea independiente y frecuentemente olvidada, resultando en sitios web obsoletos o inexistentes.

Pruebas no estandarizadas

La ejecución y validación de pruebas unitarias varía entre desarrolladores, causando que algunos errores solo aparezcan en entornos específicos.

Maven resuelve estos problemas proporcionando convenciones y automatización para todo el ciclo de vida del proyecto.

¿Qué es Maven?

Maven es una herramienta de automatización de compilación y gestión de dependencias basada en convenciones (convención sobre configuración) para proyectos Java

Características

- Gestión centralizada de dependencias a través de repositorios
- Ciclo de vida de construcción estandarizado (compile, test, package, install, deploy)
- Estructura de proyecto uniforme y predecible
- Reducción de la complejidad en la configuración de proyectos
- Sistema de plugins extensible para personalización
- Reutilización de configuraciones a través de herencia de POM
- Integración con IDEs principales (Eclipse, IntelliJ, NetBeans)
- Soporte para proyectos multi-módulo y administración de versiones

Instalación

Configurar Maven en tu entorno de desarrollo es sencillo y puede realizarse mediante diversos métodos según tus necesidades.

IDEs integrados

Eclipse, IntelliJ IDEA y NetBeans incluyen Maven preconfigurado, evitando la instalación manual.

Gestores de paquetes

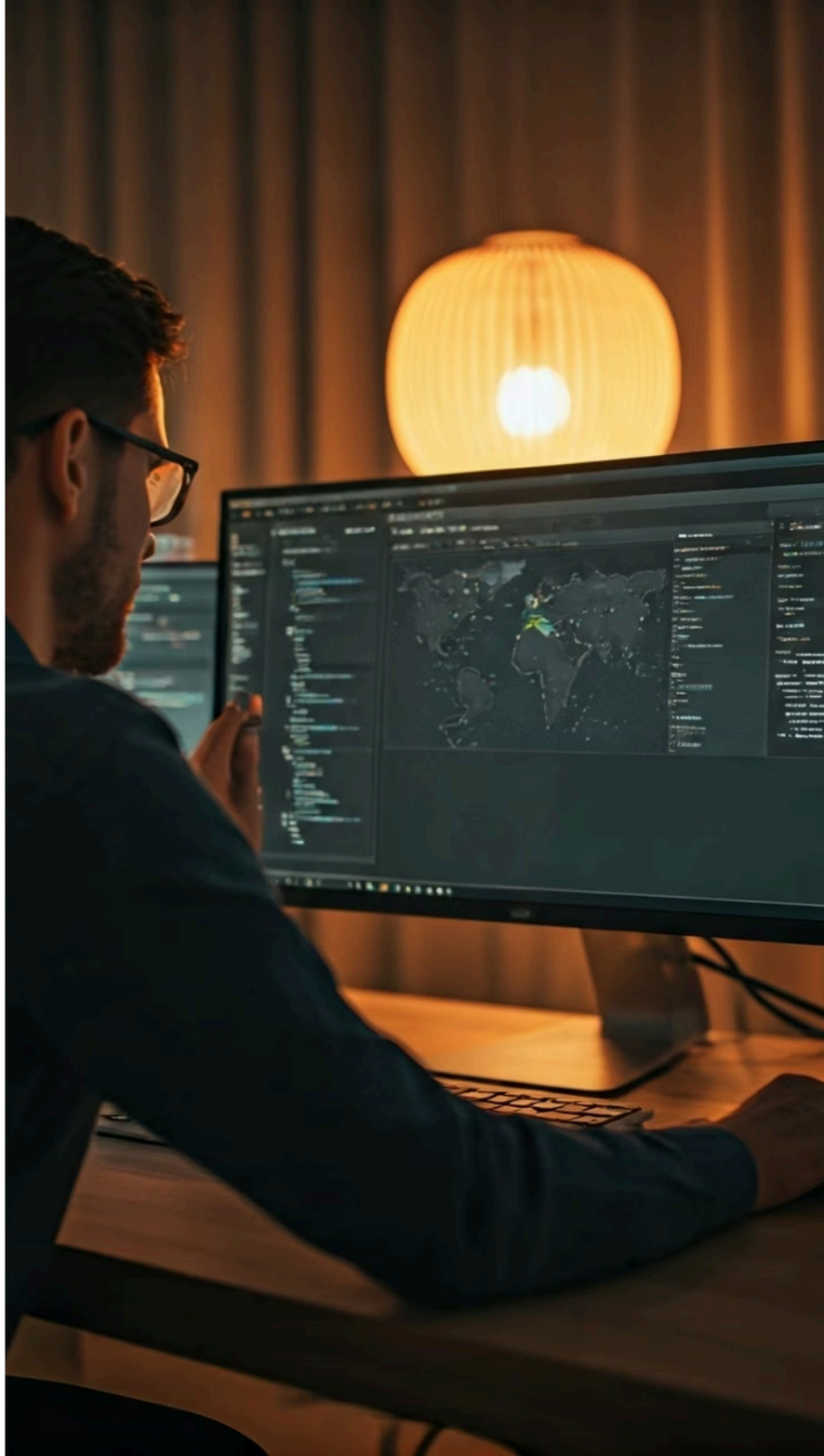
Usa apt-get en Ubuntu, brew en macOS o chocolatey en Windows para instalar Maven con un simple comando.

Instalación manual

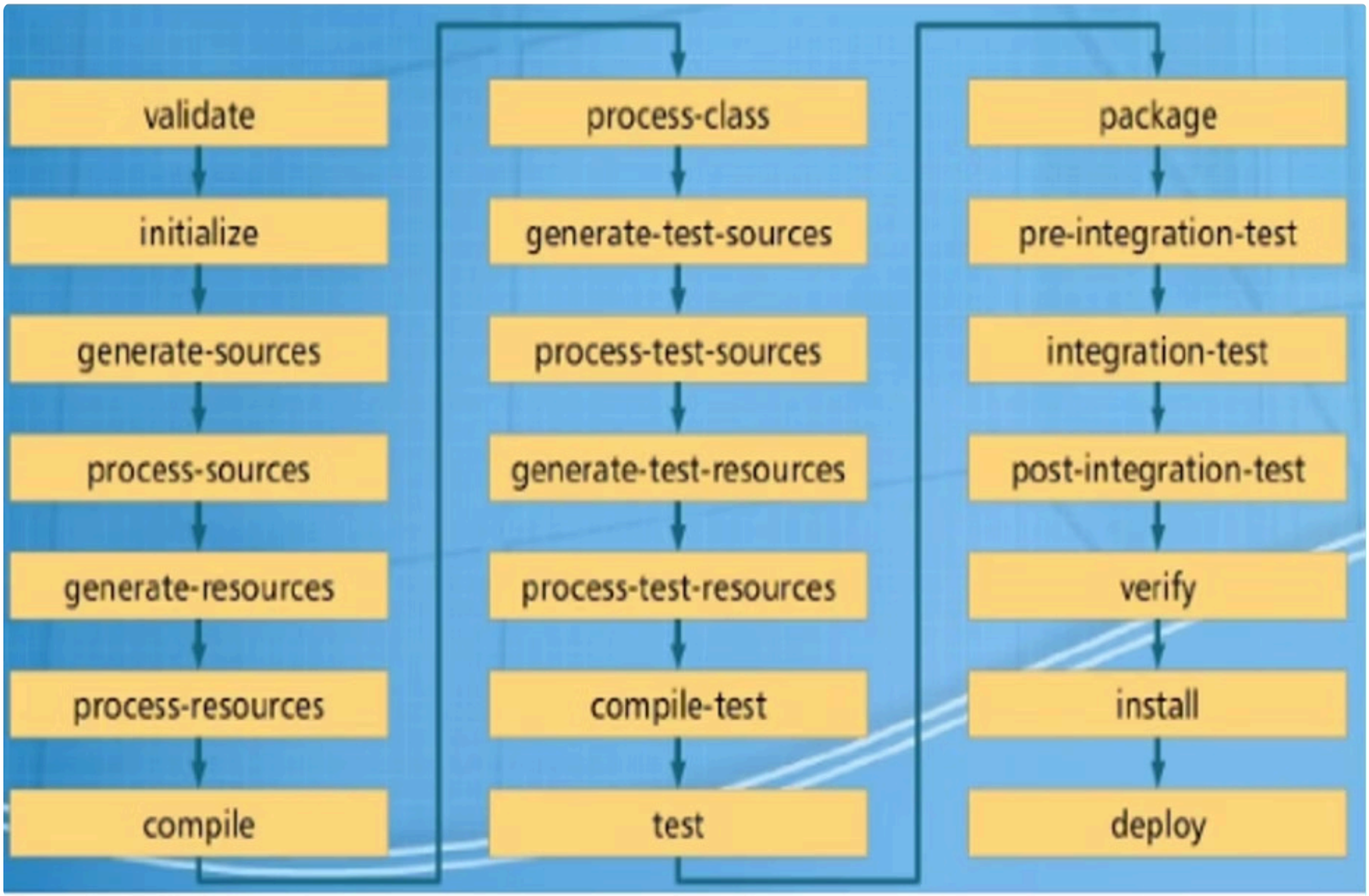
Descarga el archivo ZIP desde la web oficial de Apache Maven y configura las variables de entorno MAVEN_HOME y PATH.

Contenedores Docker

Utiliza imágenes oficiales de Maven para entornos aislados y configuraciones reproducibles entre equipos.



Ciclo de Vida de Maven



Todas las fases se ejecutan secuencialmente

Comandos Clave de Maven

mvn clean

Elimina los artefactos generados en ejecuciones previas

mvn compile

Compila el código fuente del proyecto

mvn test

Ejecuta las pruebas unitarias del proyecto

mvn package

Empaqueta el proyecto compilado en formato JAR/WAR

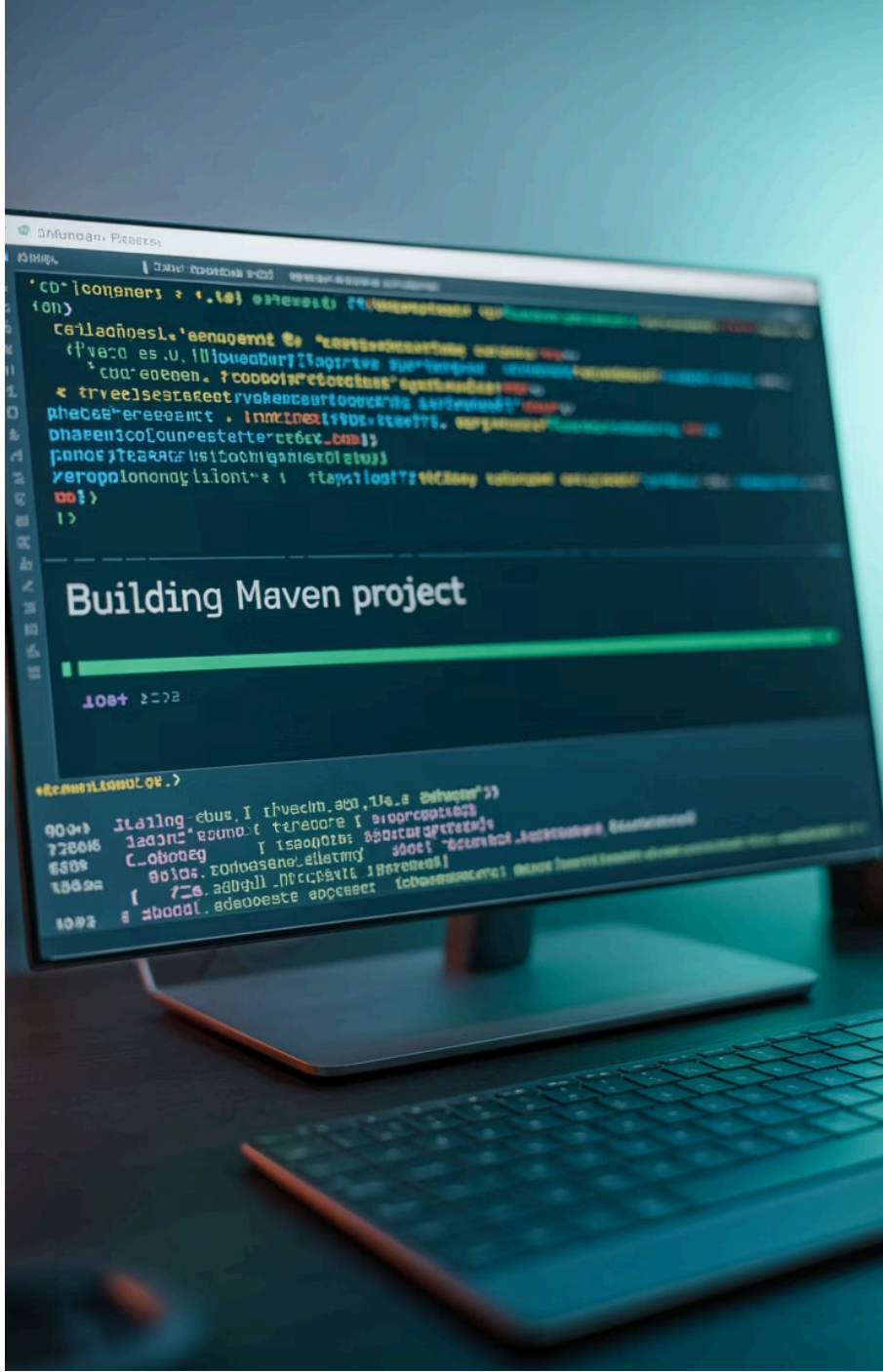
mvn install

Instala el artefacto en el repositorio local de Maven

mvn deploy

Publica el artefacto en un repositorio remoto

- Los comandos Maven son acumulativos: ejecutan automáticamente todas las fases anteriores del ciclo de vida, excepto **clean** que constituye un ciclo de vida en sí mismo

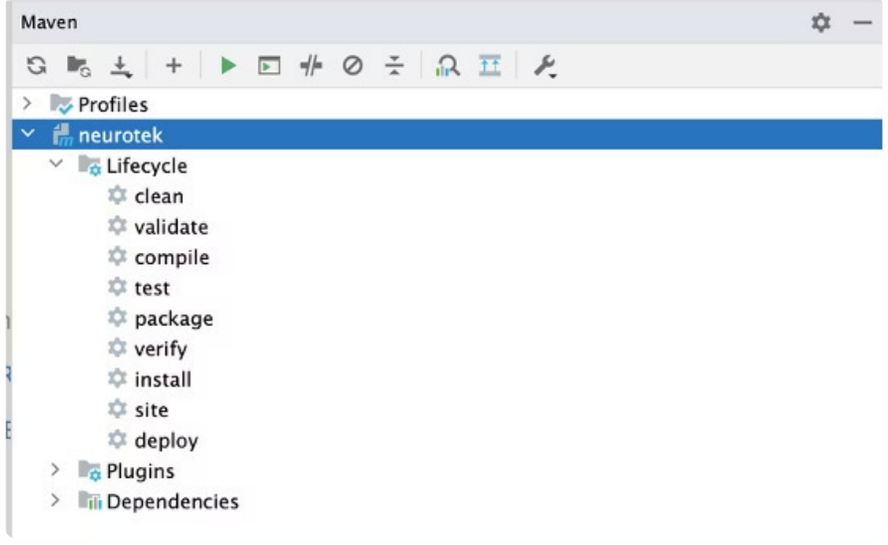


Formas de uso

La versatilidad de Maven permite a los desarrolladores trabajar de manera eficiente a través de la línea de comandos o con los entornos de desarrollo integrados.

Integración con IDE

Integración fluida de Maven en Eclipse, IntelliJ IDEA y NetBeans. Ejecutar comandos con solo unos pocos clics.



Interfaz de línea de comandos

Acceso directo a todos los comandos de Maven a través de la terminal. Ideal para scripts de automatización y pipelines de CI/CD.

```
ivanruizrube@wlandpr0219 webapp % mvn spring-boot:run
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< com.academitime:webapp >----------------------
[INFO] Building AcademiTime 2.9.0
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] >>> spring-boot:3.4.5:run (default-cli) > test-compile @ webapp >>>
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ webapp ---
[INFO] Copying 1 resource from src/main/resources to target/classes
[INFO] Copying 206 resources from src/main/resources to target/classes
[INFO]
[INFO] --- vaadin:24.7.3:prepare-frontend (default) @ webapp ---
```

Estructura de un Proyecto Maven

src/main/java

Código fuente principal

src/test/java

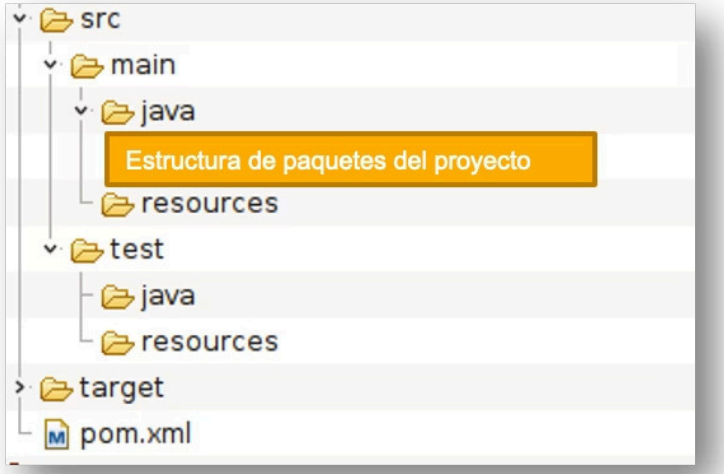
Código de pruebas

src/main/resources

Recursos (configuraciones, etc.)

pom.xml

Archivo de configuración principal



POM (Project Object Model)

Metadatos

Descripción, organización, desarrolladores y licencias del proyecto

Coordenadas del proyecto

groupId, artifactId, version que identifican unívocamente el proyecto

Dependencias

Bibliotecas externas necesarias para el proyecto

Repositorios

Ubicaciones para descargar dependencias y plugins

POM Padre

Herencia de configuraciones de un POM principal

Módulos

Subproyectos en caso de proyectos multi-módulo

Configuración de build

Directorio de salida, recursos y otras configuraciones de construcción

Plugins

Extensiones que añaden funcionalidad al proceso de build

Perfiles

Configuraciones alternativas para diferentes entornos

Fichero 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></modelVersion>

  <!-- Coordenadas -->
  <groupId></groupId>
  <artifactId></artifactId>
  <version></version>
  <packaging></packaging>

<!-- POM padre (opcional) -->
<parent>
  <groupId></groupId>
  <artifactId></artifactId>
  <version></version>
</parent>

<!-- Metadatos -->
<name></name>
<description></description>
<url></url>

<!-- Propiedades globales -->
<properties></properties>

<!-- Gestión de versiones compartidas -->
<dependencyManagement></dependencyManagement>

<!-- Dependencias -->
<dependencies></dependencies>

<!-- Configuración de compilación y empaquetado -->
<build>
  <finalName></finalName>
  <plugins></plugins>
  <pluginManagement></pluginManagement>
</build>

<!-- Reportes -->
<reporting>
  <plugins></plugins>
</reporting>

  <!-- Repositorios adicionales -->
  <repositories></repositories>
  <pluginRepositories></pluginRepositories>

  <!-- Módulos (multi-módulo) -->
  <modules></modules>

  <!-- Perfiles de construcción -->
  <profiles></profiles>

  <!-- Distribución/Despliegue -->
  <distributionManagement></distributionManagement>

</project>
```

Coordenadas

El sistema de coordenadas de Maven identifica de forma única cada artefacto en el ecosistema.

groupId

Identifica la organización o grupo. Suele seguir la convención de dominio invertido.

artifactId

Nombre del proyecto o módulo. Debe ser único dentro del groupId.

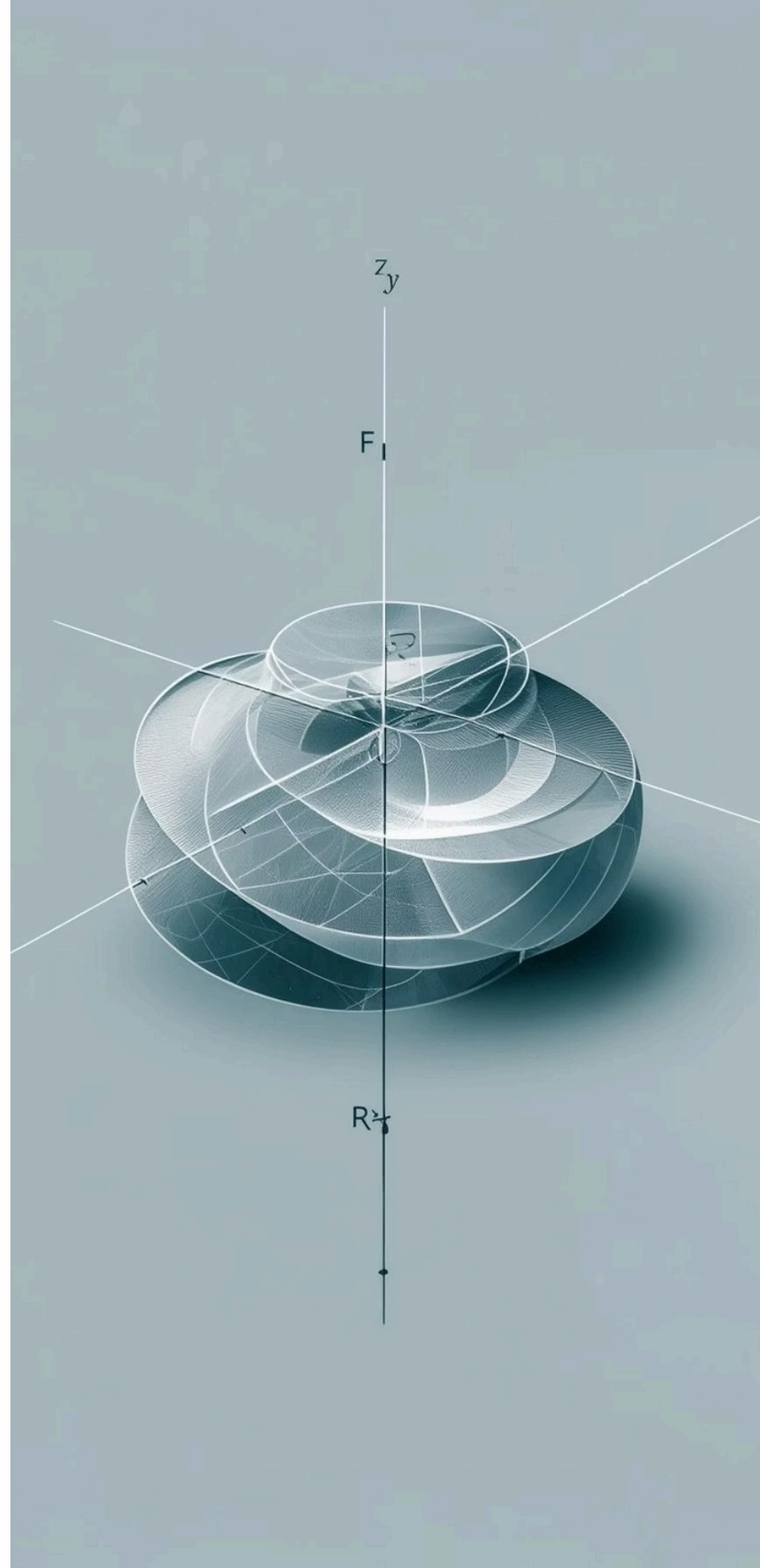
3 version

Número de versión específico. Puede incluir sufijos como SNAPSHOT para versiones en desarrollo.

packaging

Formato del artefacto generado: jar, war, ear, pom, etc.

```
<!-- coordenadas -->
<groupId>es.uca.cursoJava</groupId>
<artifactId>ejemplo</artifactId>
<version>1.0.0-SNAPSHOT</version>
<packaging>jar</packaging>
```



Coordenadas: versionado semántico

Sistema estandarizado para numerar versiones de software que facilita la gestión de dependencias en Maven.

Semantic Versioning

Semantic Versioning 2.0.0

Semantic Versioning spec and website

MAJOR (X.y.z)

Cambios incompatibles con versiones anteriores. Requiere actualizar código cliente.

MINOR (x.Y.z)

Funcionalidades nuevas manteniendo compatibilidad. No rompe API existente.

PATCH (x.y.Z)

Correcciones de errores compatibles. Soluciona bugs sin añadir funciones.

Sufijos especiales

SNAPSHOT para desarrollo, RC para candidatas a finales, ALPHA/BETA para versiones preliminares.

Metadatos

Los metadatos en Maven proporcionan información descriptiva esencial sobre el proyecto.

Información del Proyecto

Nombre, descripción, URL y año de inicio del proyecto.

Organización

Datos sobre la entidad responsable del desarrollo.

Desarrolladores

Lista de contribuyentes con roles y datos de contacto.

Licencias

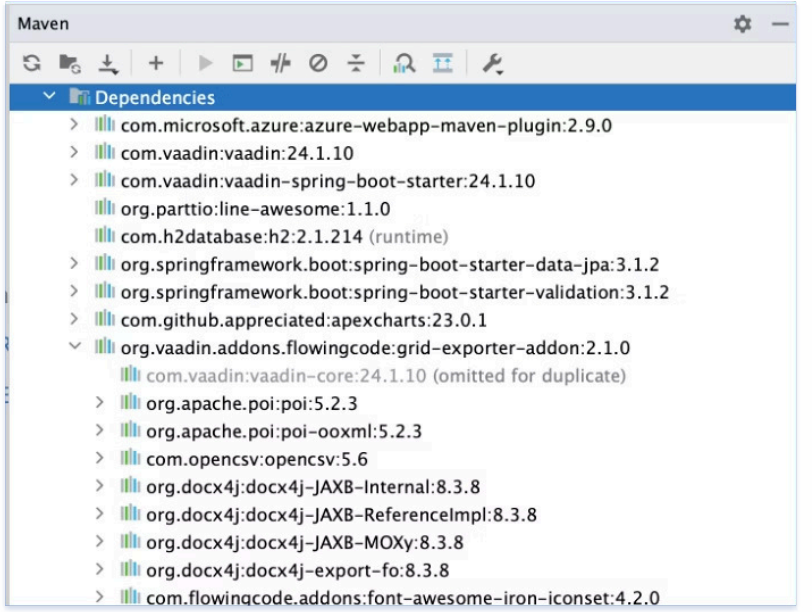
Términos legales que rigen la distribución del software.

Metadatos: ejemplo

```
<!-- Metadatos del proyecto -->
  <name>Mi Aplicación de Ejemplo</name>
  <description>Este ejemplo demuestra la estructura de un proyecto Maven</description>
  <url>https://github.com/ejemplo/mi-aplicacion</url>

  <!-- Organización responsable del proyecto -->
  <organization>
    <name>Ejemplo S.A.</name>
    <url>https://www.ejemplo.com</url>
  </organization>

  <!-- Información de licenciamiento -->
 <licenses>
   <license>
     <name>Apache License, Version 2.0</name>
     <url>https://www.apache.org/licenses/LICENSE-2.0.txt</url>
     <distribution>repo</distribution>
   </license>
 </licenses>

 <!-- Equipo de desarrolladores -->
 <developers>
   <developer>
     <id>jdoe</id>
     <name>Juan Pérez</name>
     <email>juan.perez@ejemplo.com</email>
     <organization>Ejemplo S.A.</organization>
     <organizationUrl>https://www.ejemplo.com</organizationUrl>
   </developer>
   <developer>
     <id>msanchez</id>
     <name>María Sánchez</name>
     <email>maria.sanchez@ejemplo.com</email>
     <organization>Ejemplo S.A.</organization>
     <organizationUrl>https://www.ejemplo.com</organizationUrl>
   </developer>
 </developers>

 <!-- Repositorio de control de versiones -->
 <scm>
   <connection>scm:git:git://github.com/ejemplo/mi-aplicacion.git</connection>
   <developerConnection>scm:git:ssh://github.com:ejemplo/mi-aplicacion.git</developerConnection>
   <url>https://github.com/ejemplo/mi-aplicacion</url>
   <tag>HEAD</tag>
 </scm>
```

Dependencias

Maven simplifica la gestión de bibliotecas mediante un sistema de resolución de dependencias anidadas. Las dependencias declaradas en el pom.xml son automáticamente descargadas.



Dependencias Directas

bibliotecas declaradas explícitamente en el POM

Dependencias Transitivas

Bibliotecas requeridas indirectamente por tus dependencias

> El motor de Maven analiza y resuelve automáticamente toda la jerarquía de dependencias, evitando conflictos y versiones incompatibles.

Dependencias: repositorio local de Maven

El lugar centralizado donde Maven almacena todas las dependencias descargadas de tu proyecto.

Ubicación Predeterminada

Normalmente se encuentra en la carpeta *${user.home}*/.m2/repository dentro del directorio personal del usuario.

Caché Inteligente

Evita descargas repetidas de artefactos, mejorando la velocidad de construcción en proyectos grandes.

Estructura Organizativa

Organizado por groupId/artifactId/version, facilitando la gestión de múltiples versiones de bibliotecas.

Personalización

Configurable mediante *settings.xml* para cambiar la ubicación o usar repositorios espejo.

Dependencias: repositorios remotos de Maven

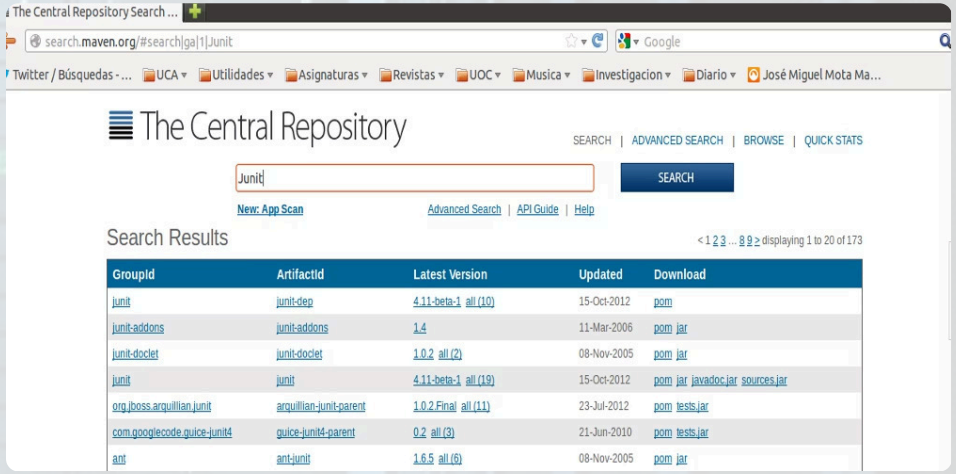
Los almacenes centralizados donde Maven busca dependencias que no encuentra en el repositorio local.

Central Maven

Repositorio oficial y predeterminado de Maven.

Corporativos

Repositorios internos de empresas (Nexus, Artifactory, Github).



Dependencias: definición

1

Declaración

Cada dependencia se especifica con su groupId, artifactId y version.

2

Scope

Existen cuatro posibles ámbitos:

- **Compile:** Necesaria para compilar y se incluye en el artefacto final.
- **Runtime:** No se necesita para compilar, pero sí para ejecutar la aplicación.
- **Test:** Sólo se necesita para las pruebas unitarias, no se incluye en el artefacto.
- **Provided:** La proporciona el entorno de ejecución, no se incluye en el artefacto.

3

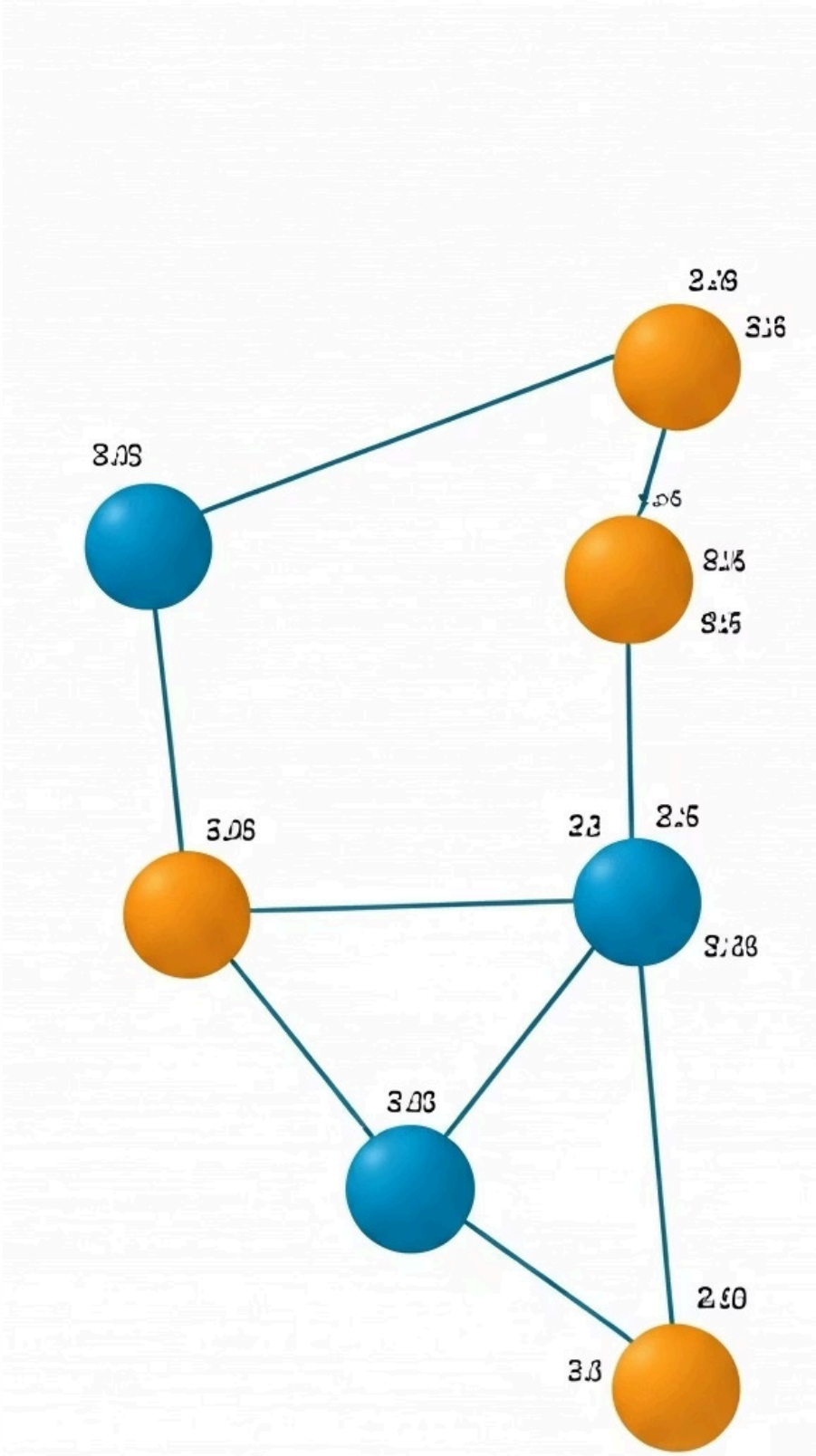
Exclusiones

Permite eliminar dependencias transitivas no deseadas mediante etiquetas exclusions.

4

Opcional

Con optional=true, indica que la dependencia no es obligatoria para quienes usen tu proyecto.

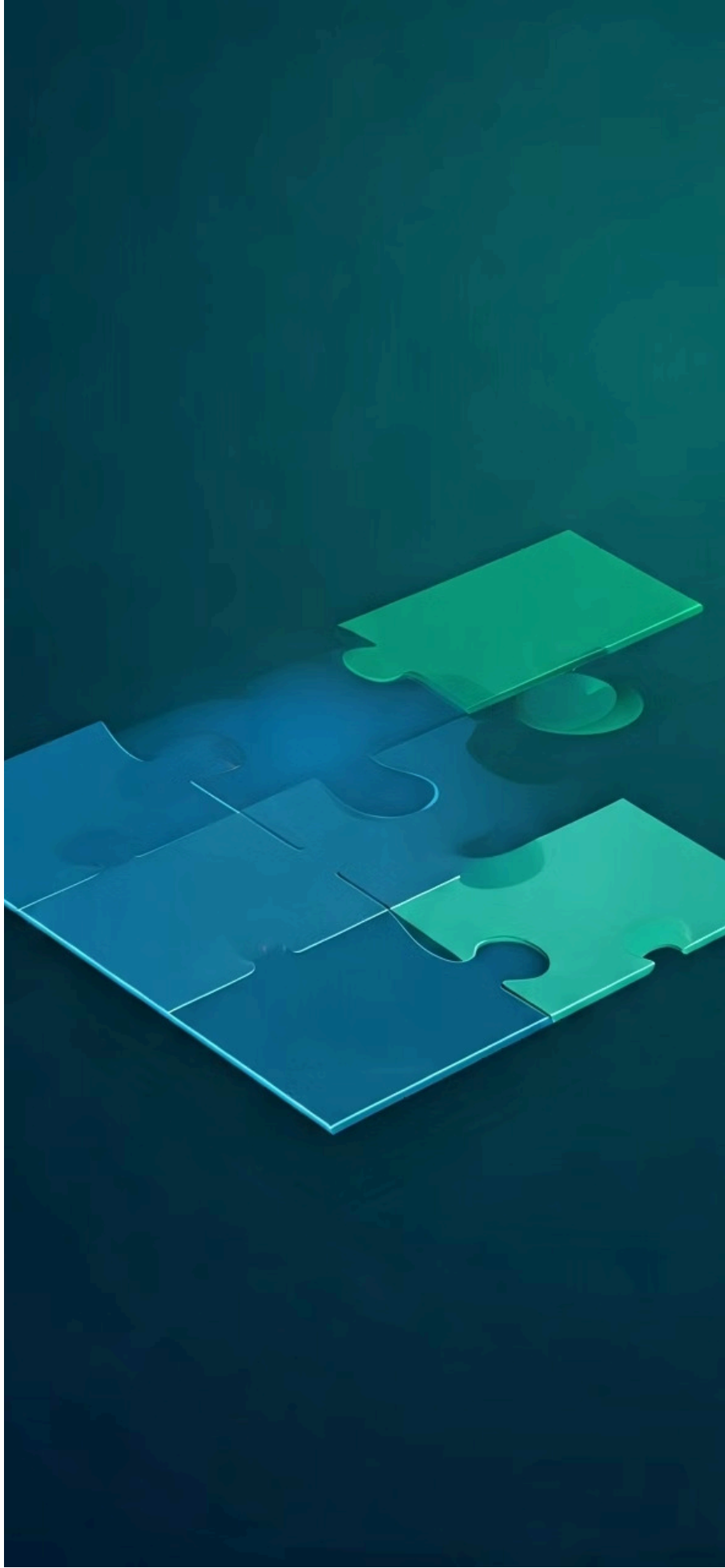


Dependencias: resolución de conflictos

En proyectos grandes, es posible tener múltiples dependencias que requieran versiones diferentes de la misma biblioteca

```
<project>
  ...
  <dependencies>
    <dependency>
      <groupId>com.ejemplo</groupId>
      <artifactId>A</artifactId>
      <version>1.0</version>
    </dependency>
    <!-- Dep. directa A:1.0 → trae C:2.0 -->

    <dependency>
      <groupId>com.ejemplo</groupId>
      <artifactId>B</artifactId>
      <version>1.5</version>
    </dependency>
    <!-- Dep. directa B:1.5 → trae C:3.1 -->
  </dependencies>
</project>
```

Visualizar el árbol de dependencias con **mvn dependency:tree** facilita identificar y resolver conflictos complejos.



Dependencias: estratégias de resolución de conflictos

Más cercana

Por defecto, Maven toma la versión de la dependencia que esté mas cerca de la raíz. La primera declaración prevalece en caminos de igual longitud, sin importar si es la más nueva o antigua.

Forzar versión concreta

Podemos indicar explícitamente qué versión de la librería conflictiva queremos usar.

```
<dependencyManagement>
   <dependencies>
    <!-- Forzamos C a la versión 3.1 -->
    <dependency>
     <groupId>com.ejemplo.libs</groupId>
     <artifactId>C</artifactId>
     <version>3.1</version>
    </dependency>
   </dependencies>
  </dependencyManagement>
```

Exclusiones manuales

Podemos indicar que una determinada dependencia transitiva sea excluida.

```
<dependency>
  <groupId>com.ejemplo</groupId>
  <artifactId>A</artifactId>
  <version>1.0</version>
  <exclusions>
   <exclusion>
    <groupId>com.ejemplo.libs</groupId>
    <artifactId>C</artifactId>
   </exclusion>
  </exclusions>
</dependency>
```

Dependencias: cuando la unificación no es posible

Shading (reubicación)

Renombra paquetes para evitar conflictos entre las clases.

- Utiliza maven-shade-plugin
- Internaliza dependencias problemáticas
- Evita colisiones de versiones

Módulos independientes

Separa funcionalidades con dependencias conflictivas. Mayor complejidad

- Aplicaciones OSGI
- Microservicios
- CassLoader personalizado

Dependencias: escaneo de vulnerabilidades

Maven ofrece plugins para escanear las dependencias de tu proyecto en busca de vulnerabilidades de seguridad conocidas, ayudándote a mantener un código seguro.

Dependency Check

Utiliza la base de datos NVD para identificar componentes con vulnerabilidades conocidas en tus dependencias.

Versions Plugin

Ayuda a detectar dependencias obsoletas que podrían contener problemas de seguridad.

OWASP Plugin

Crea informes detallados de vulnerabilidades basados en estándares de seguridad de la industria.

Dependabot de GitHub

Herramienta integrada en GitHub que detecta automáticamente vulnerabilidades y propone actualizaciones mediante pull requests.

Ejecuta escaneos de seguridad regularmente con **mvn dependency-check:check** para identificar y mitigar posibles riesgos de seguridad.

Repositorios

Maven busca en los repositorios en el siguiente orden:

1. **Repositorio local** (cache en `~/.m2/repository`).
2. **Repositorios definidos en** `settings.xml` (si el usuario tiene repositorios globales configurados).
3. **Repositorios del POM** (`<repositories>`, en el orden declarado).
4. **Repositorio central de Maven** (`https://repo.maven.apache.org/maven2`), a menos que lo deshabilites explícitamente.

```
<repositories>
  <!-- Repositorio de librerías de snapshots internas -->
  <repository>
    <id>nexus-internal-snapshots</id>
    <url>https://nexus.miempresa.local/repository/maven-snapshots/</url>
    <releases>
      <enabled>false</enabled> <!-- No buscar actualizaciones para artefactos estables -->
    </releases>
    <snapshots>
      <enabled>true</enabled>       <!-- Sí para artefactos SNAPSHOT -->
      <updatePolicy>always</updatePolicy> <!-- Forzar comprobación en cada build -->
    </snapshots>
  </repository>
</repositories>
```

Build

Maven permite personalizar diversos aspectos del proceso de construcción mediante configuraciones en el POM.

Directorio de salida

Configura dónde se generarán los artefactos compilados con **outputDirectory** o **build.directory**.

Recursos

Define ubicaciones y filtrado de recursos no-Java mediante **resources** en la sección **build**.

Exclusiones

Especifica archivos que no deben procesarse usando patrones de exclusión en las configuraciones de recursos.

Propiedades

Establece variables del proyecto para filtrar recursos y parametrizar la construcción.

Plugins

Extensiones que añaden funcionalidad al ciclo de construcción estándar de Maven.

Core

maven-compiler-plugin, maven-jar-plugin, maven-surefire-plugin, etc.

Packaging

maven-jar-plugin, maven-war-plugin, spring-boot-maven-plugin, etc.

Reporting

maven-javadoc-plugin, maven-site-plugin, maven-checkstyle-plugin, etc.

Herramientas

maven-deploy-plugin, maven-release-plugin, dockerfile-maven-plugin, azure-webapp-maven-plugin, etc.

Perfiles de Maven

Los perfiles permiten adaptar la construcción del proyecto a diferentes contextos de ejecución (entorno, sistema operativo o propiedades JVM).

Configura diferentes repositorios, dependencias o plugins según el entorno de desarrollo, test o producción.

```
<profiles>
 <profile>
  <!-- Production mode is activated using -Pproduction -->
  <id>test</id>

  <properties>
      <spring.profiles.active>test</spring.profiles.active>
  </properties>

  <dependencies>
    <!-- Usamos un driver de BD distinto al de desarrollo -->
    <dependency>
      <groupId>com.oracle.database.jdbc</groupId>
      <artifactId>ojdbc11</artifactId>
      <version>23.5.0.24.07</version>
    </dependency>
  </dependencies>
 </profile>
</profiles>
```

POM padre

La herencia en Maven permite compartir configuración entre proyectos relacionados mediante un POM padre.

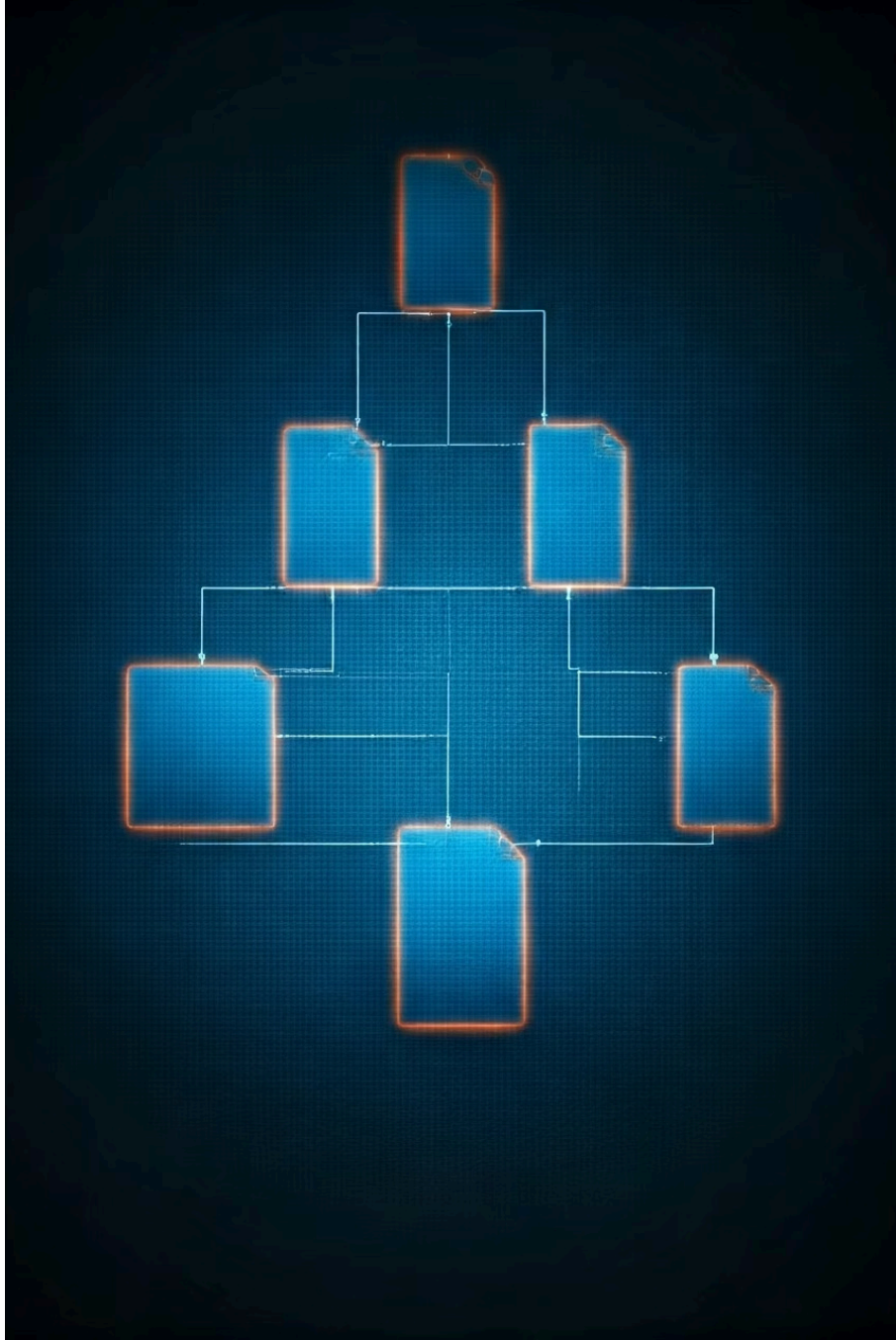
Configuración centralizada

Gestiona versiones, plugins y dependencias desde un único punto para múltiples módulos o proyectos.

Estructura del proyecto

Los proyectos hijo heredan y pueden sobrescribir la configuración del POM padre según sus necesidades específicas.

```
<parent>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-parent</artifactId>
 <version>3.4.5</version>
</parent>
```



Módulos: proyectos modulares

Maven permite organizar proyectos complejos en módulos independientes con un POM padre común.

```
mi-proyecto-raiz/
 ├──── pom.xml        ← POM padre
 ├──── core/
 │     └──── pom.xml     ← POM módulo "core"
 ├──── service/
 │     └──── pom.xml     ← POM módulo "service"
 └──── web/
       └──── pom.xml     ← POM módulo "web"
```

Módulo Raíz

Contiene el POM padre y coordina la construcción de todos los módulos.

```
<modules>
  <module>core</module>
  <module>web</module>
  <module>service</module>
</modules>

<dependencyManagement>
    <!-- Versiones compartidas -->
</dependencyManagement>

<properties>
    <!-- Propiedades comunes -->
</properties>
```

Módulos Independientes

Cada uno con su propio POM que hereda configuración del padre.

```
<parent>
  <groupId>com.ejemplo.multimodulo</groupId>
  <artifactId>mi-proyecto-raiz</artifactId>
  <version>1.0.0</version>
  <relativePath>../pom.xml</relativePath>
</parent>

<artifactId>service</artifactId>
<packaging>jar</packaging>

<dependencies>
  <dependency>
    <groupId>com.ejemplo.multimodulo</groupId>
    <artifactId>core</artifactId>
  </dependency>
...
</dependencies>
```

CI/CD con Maven

Maven organiza el ciclo de vida de compilación, pruebas y empaquetado. Al incorporarlo en los pipeline de **integración y despliegue continuo**, cada cambio en el repositorio podría disparar este proceso automáticamente. Veámos un ejemplo con Github Actions:

```
name: App deploy Testing

on:
  push:
    branches: [ "main" ]
  workflow_dispatch:

jobs:
  build:
    runs-on: ubuntu-local

    steps:
      - name: Checkout code, setup Maven, cache dependencies and more...
        uses: s4u/setup-maven-action@v1.18.0
    with:
      java-distribution: 'graalvm'
      java-version: '21'

  - name: Get Artifact version from pom.xml
    uses: entimaniac/read-pom-version-action@1.0.0
    id: getVersion

  - name: Build and package
    run: mvn clean package -Dmaven.test.skip=true -U -B -Pprod
    env:
      GITHUB_TOKEN: ${{ github.token }}

  - name: Rename jar file
    run: cp target/webapp-${{ steps.getVersion.outputs.version }}.jar ~/webapp.jar

  - name: Reiniciar la aplicación con systemd
    run: |
      systemctl restart app.service
      systemctl status app.service --no-pager
```

Alternativas a Maven

Ecosistema Java

Gradle

Combina lo mejor de Ant y Maven con scripts basados en Groovy.

Ant + Ivy

Más flexible pero requiere configuración manual detallada.

Bazel

Desarrollado por Google, optimizado para proyectos grandes.

Otros Ecosistemas

JavaScript/Node.js

npm, Yarn, pnpm para gestión de dependencias

Python

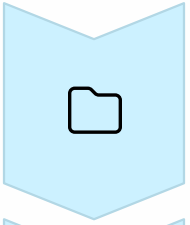
pip, Poetry, Conda para entornos y paquetes

.NET

NuGet para gestión de paquetes, MSBuild para compilación

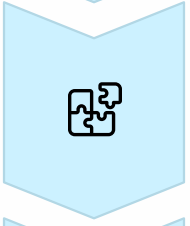
Resumen

Maven se ha establecido como una herramienta esencial para la gestión del ciclo de vida de proyectos Java, proporcionando automatización, estandarización y simplicidad.



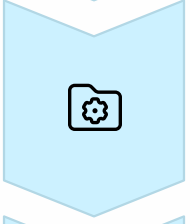
Estructura Estandarizada

Convención sobre configuración para una organización clara y predecible de proyectos.



Gestión de Dependencias

Resolución automática de dependencias desde repositorios locales y remotos.



Ciclo de Vida

Fases predefinidas que simplifican la compilación, prueba y empaquetado.

Integración CI/CD

Perfecta integración con sistemas de entrega continua para despliegues automatizados.