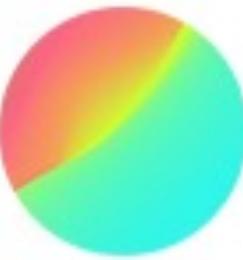
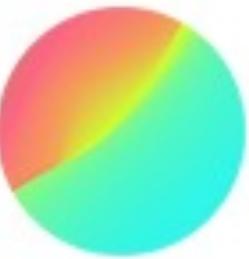


MAVEN



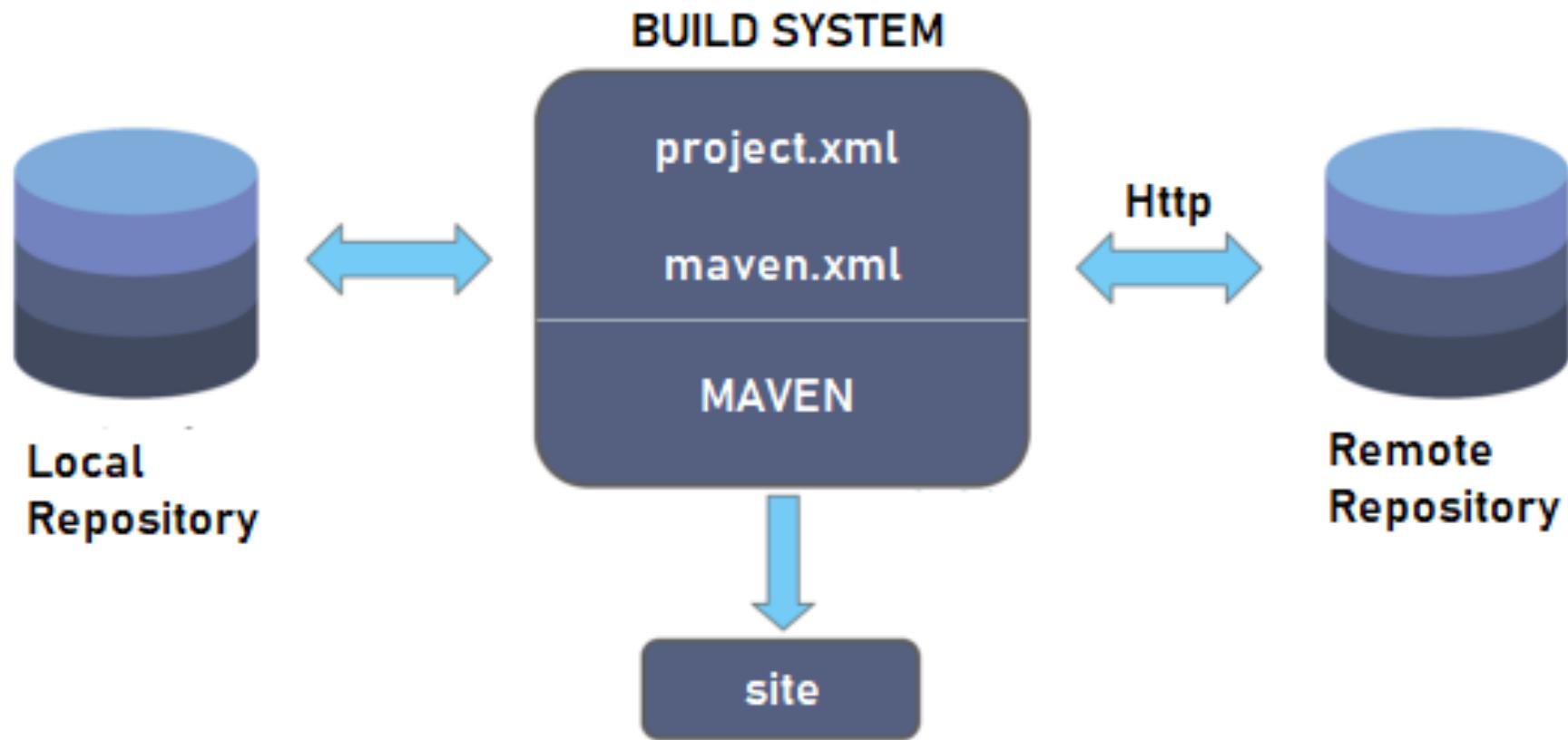
- Es una herramienta de gestión de proyectos de software que brinda un nuevo concepto del modelo de objetos de proyecto (POM).
- Es una herramienta de compilación y construcción de proyectos.
- Maven gestiona repositorios de archivos jar.
- Simplifica y estandariza el proceso de construcción de proyectos.
- Tiene la capacidad de ejecutar JUnit y otras pruebas de integración.

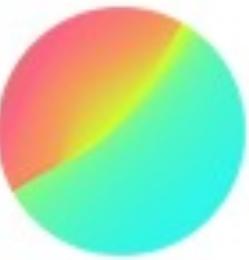




Instalar

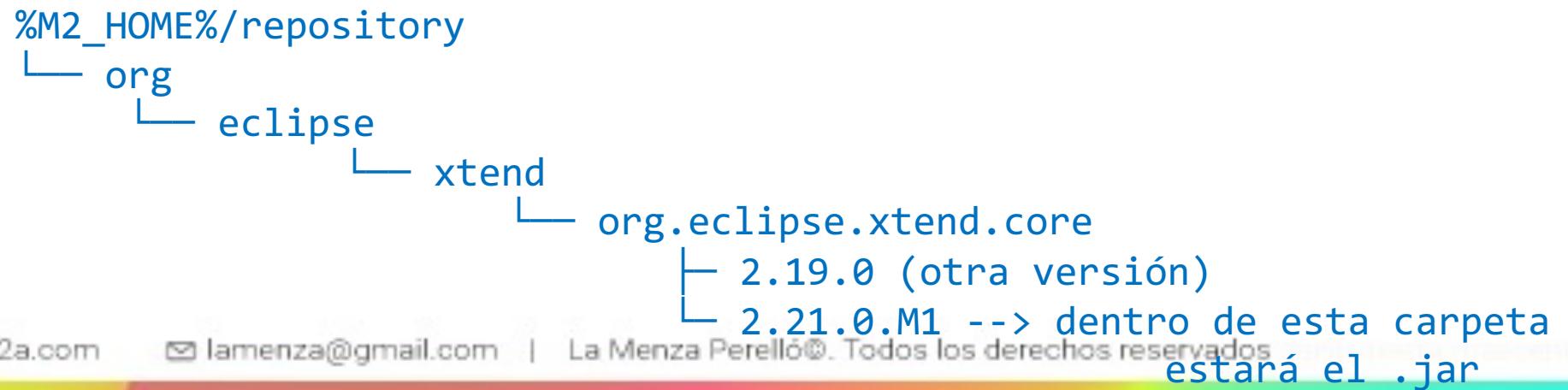
- Instalar JDK y setear **JAVA_HOME** en el **Path**
- Extraer la distro en una carpeta: **unzip apache-maven-3.9.2-bin.zip**
- Agregar la carpeta **/bin** de **Maven** al **Path**
- Confirmar con **mvn -v** (en otra terminal)

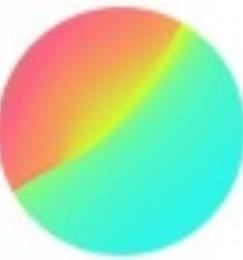




Repositorio local de Maven

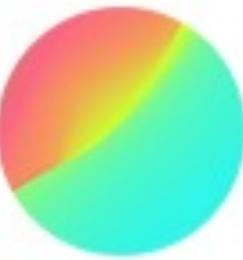
El repositorio local de Maven es donde Maven almacena todos los archivos jar del proyecto, bibliotecas o dependencias. De forma predeterminada, el nombre de la carpeta se establece en '**.m2**'.
De forma predeterminada, se ubica en el directorio raíz del usuario.





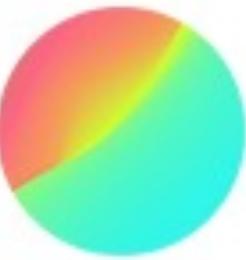
Repositorio central de Maven

El repositorio central de Maven es la ubicación predeterminada en la nube de donde Maven descarga todas las bibliotecas de dependencia de los proyecto. Para cualquier biblioteca involucrada en el proyecto, Maven primero busca en la carpeta .m2 del Repositorio local, y si no encuentra la librería necesaria, busca en el Repositorio central y carga la biblioteca en el repositorio local.

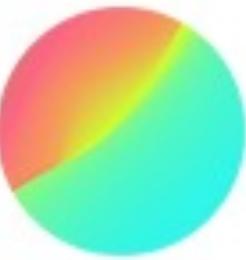


Elementos utilizados en la creación del archivo pom.xml

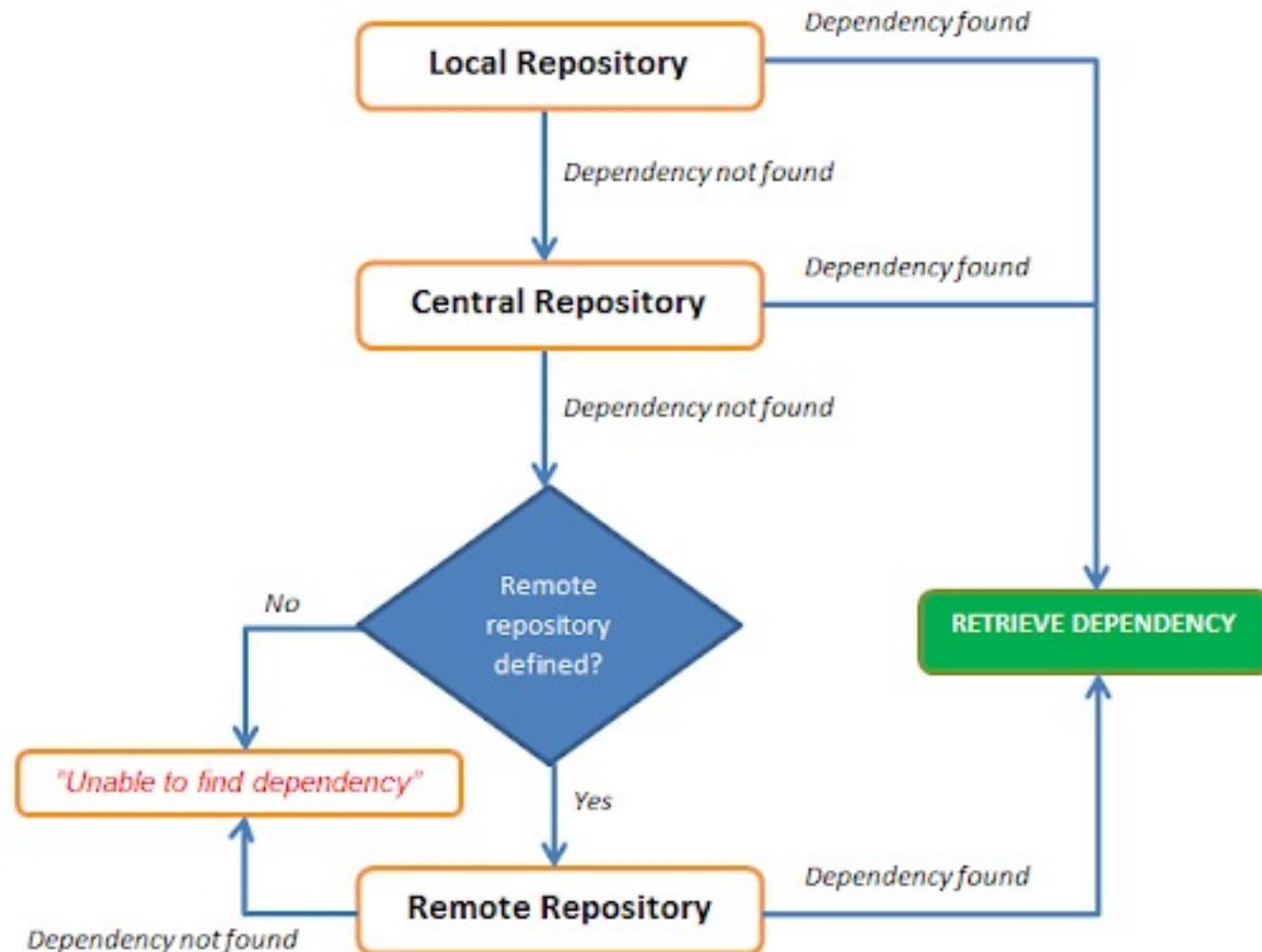
- **proyecto**: el proyecto es el elemento raíz del archivo pom.xml.
- **modelVersion**: significa la versión del modelo POM con el que está trabajando.
- **groupId**: implica el id del grupo del proyecto. Es único y, con mayor frecuencia, aplicará un ID de grupo relacionado con el nombre del paquete raíz de Java.
- **artifactId**: se utiliza para proporcionar el nombre del proyecto que está creando.
- **Versión**: este elemento consta del número de versión del proyecto. Si su proyecto ha sido lanzado en varias versiones, entonces es conveniente presentar la versión de su proyecto.

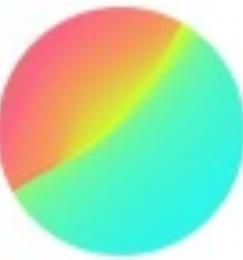


```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.lm2a</groupId>
  <artifactId>project-jsf-ejb</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>
    <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
      <maven.compiler.source>11</maven.compiler.source>
      <maven.compiler.target>11</maven.compiler.target>
      <failOnMissingWebXml>false</failOnMissingWebXml>
    </properties>
    <dependencies>
      <dependency>
        <groupId>jakarta.platform</groupId>
        <artifactId>jakarta.jakartaee-api</artifactId>
        <version>8.0.0</version>
        <scope>provided</scope>
      </dependency>
    </dependencies>
  </project>
```



Flujo general de descarga de componentes





Dependencias transitivas

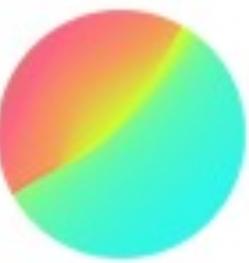
Un detalle no menor de la resolución de dependencias de maven es que también funciona para las dependencias transitivas.

Por ejemplo:

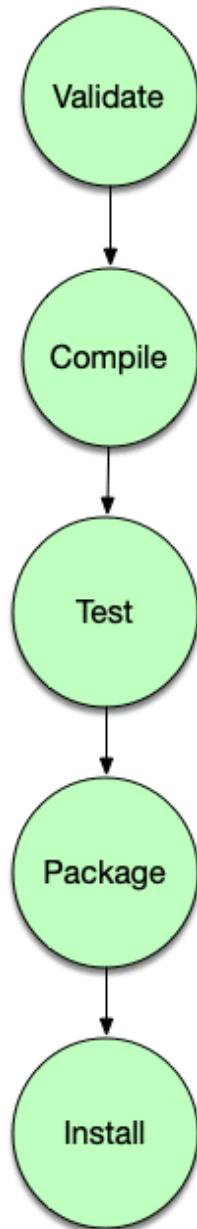
- **proyectoA → proyectoB**
- **proyectoB → proyectoC**
- **proyectoC → proyectoD, proyectoE, proyectoF**

Al resolver las dependencias, el proyectoA necesitará descargar los componentes B, C, D, E y F. Incluso podríamos requerir diferentes versiones de los mismos componentes

Ciclo de vida

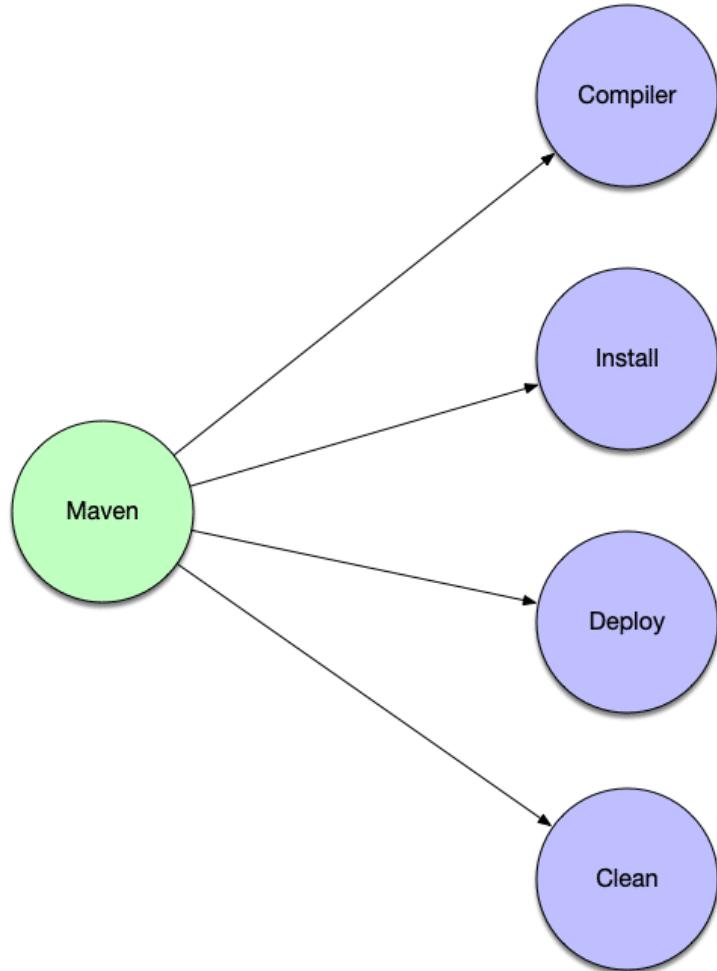
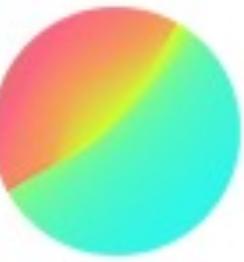


Goals



- **Validate** : Valida el proyecto y revisa que toda la información para construir el software este disponible.
- **Compile**: Compila el código fuente para generar los binarios correspondientes que se ejecutarán.
- **Test** : Ejecuta las pruebas unitarias que validan el código previamente construido.
- **Package** : Empaque la aplicación en un formato distribuible tipo JAR ,WAR .EAR etc
- **Install**: Despliega el Package en el repositorio local de nuestro equipo.

Plugins



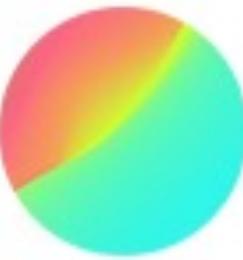
¿Como se encarga Maven de ejecutar todas estas fases de la construcción del software? .

La realidad es que el no lo puede hacer solo .

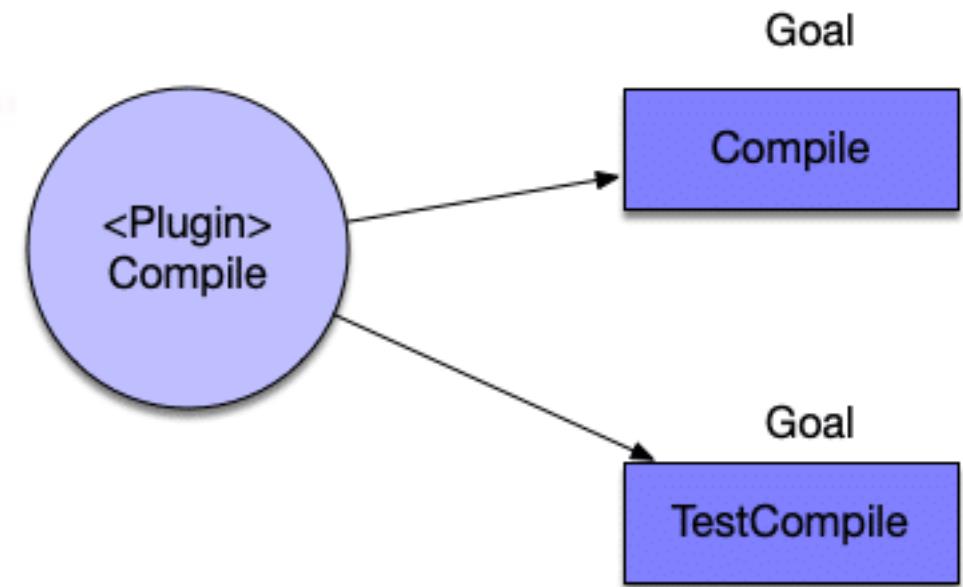
Sino que necesita de un grupo de **Plugins**.

Cada uno de los **Plugins** se encarga de una tarea concreta.

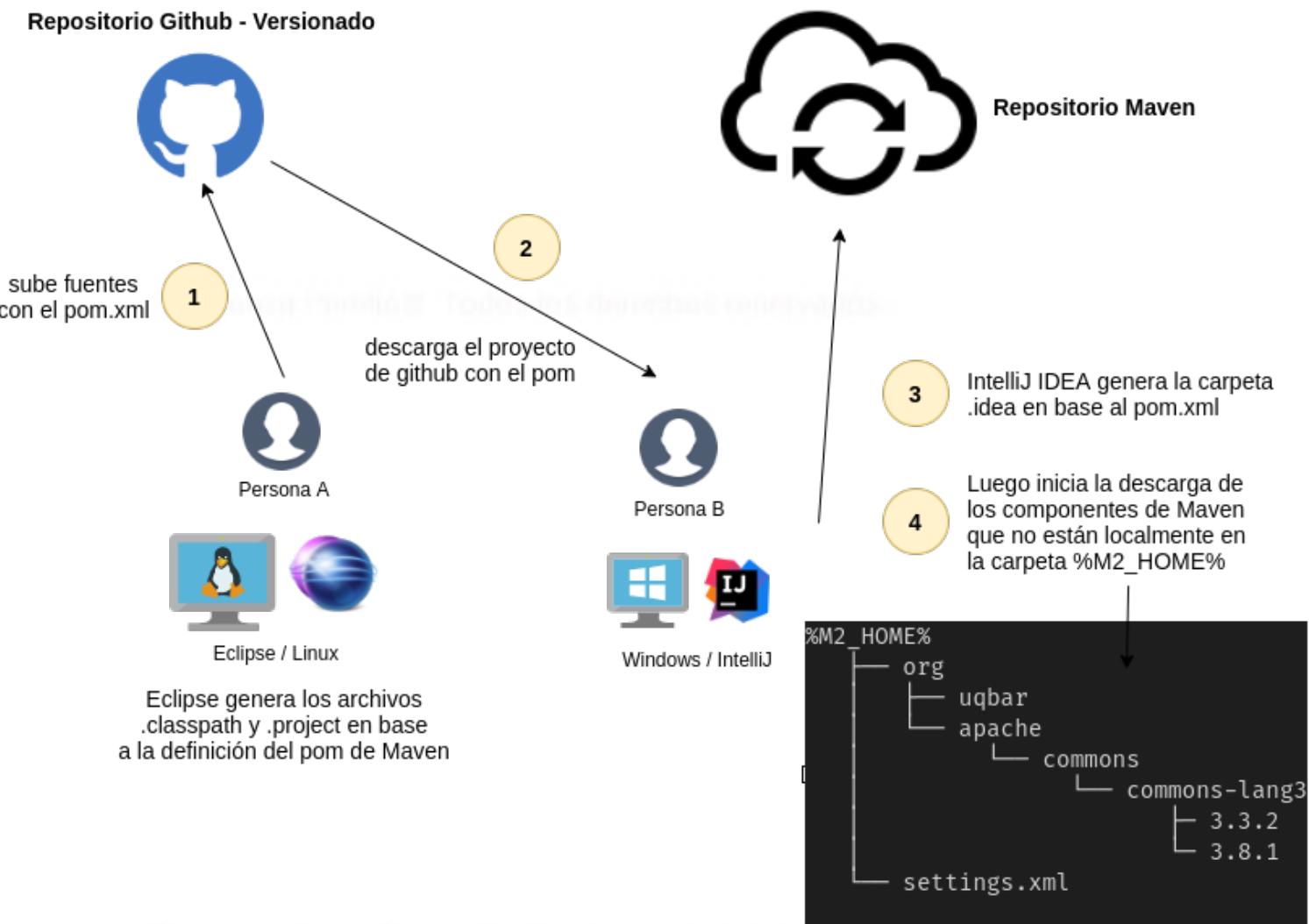
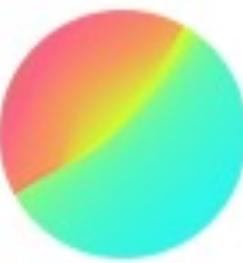
Relacion entre Goal y Plugin



Cada plugin dispone de varias tareas a ejecutar. Por ejemplo el plugin de Compile dispone de dos tareas a cada tarea se la denomina Maven Goal. En este caso la tarea de Compile que compila el código y la tarea o goal de TestCompile que compila la parte de Testing.

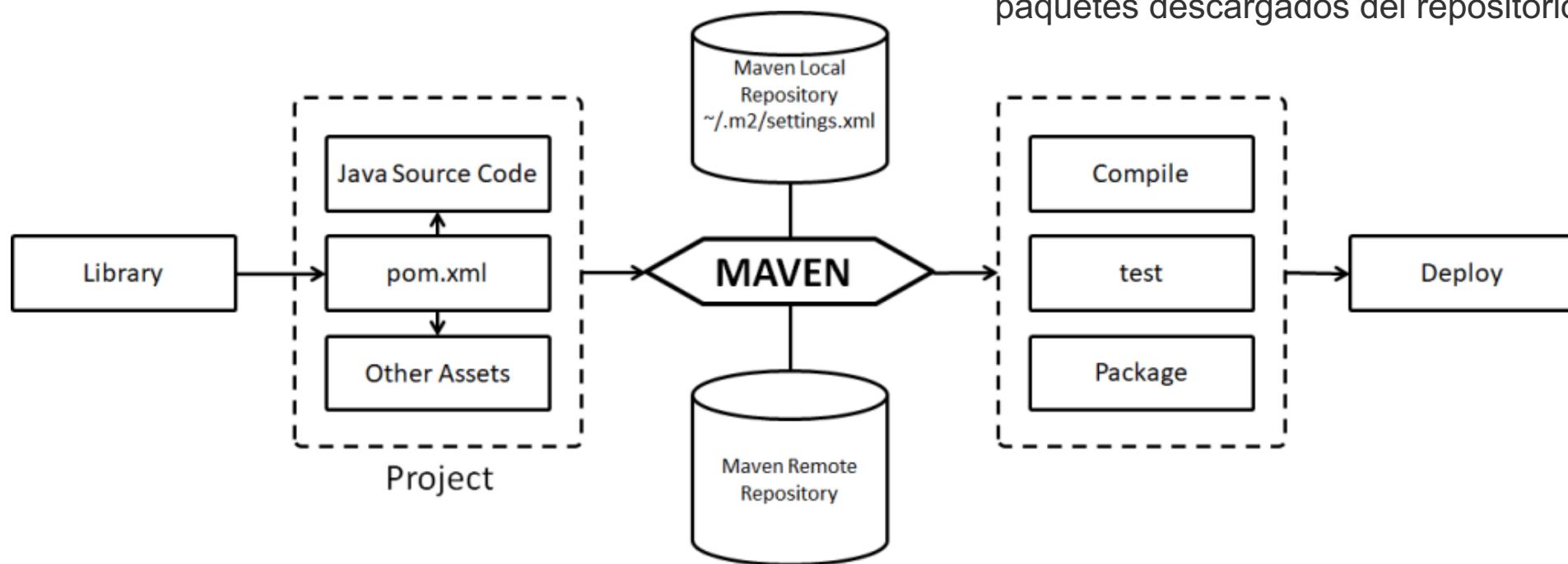


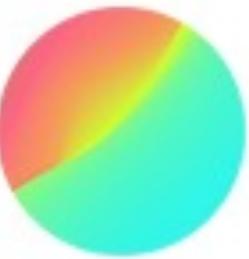
Resumen general de la arquitectura Maven



Maven comenzará un proyecto para compilar, construir, probar, empaquetar e implementar (*compile, build, test, pack y deploy*) en función del archivo pom. Este archivo pom estará disponible en el directorio raíz del proyecto Java. Puede tener archivos sub pom que pueden llamarse como módulos.

Por lo tanto, el ciclo de vida del proceso de Maven comienza con la recopilación de las bibliotecas enumeradas en el pom (que se importarán en el código fuente de Java). Luego, Maven compilará, construirá, probará o implementará según los objetivos definidos en el archivo pom. Todos los complementos y bibliotecas se pueden descargar desde el repositorio remoto y guardar en el repositorio maven local o se pueden reutilizar los paquetes descargados del repositorio maven local.

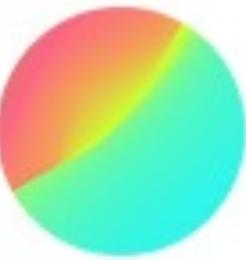




Ciclos de vida

Cada ciclo de vida consiste en una secuencia de fases. Maven tiene tres ciclos de vida definidos:

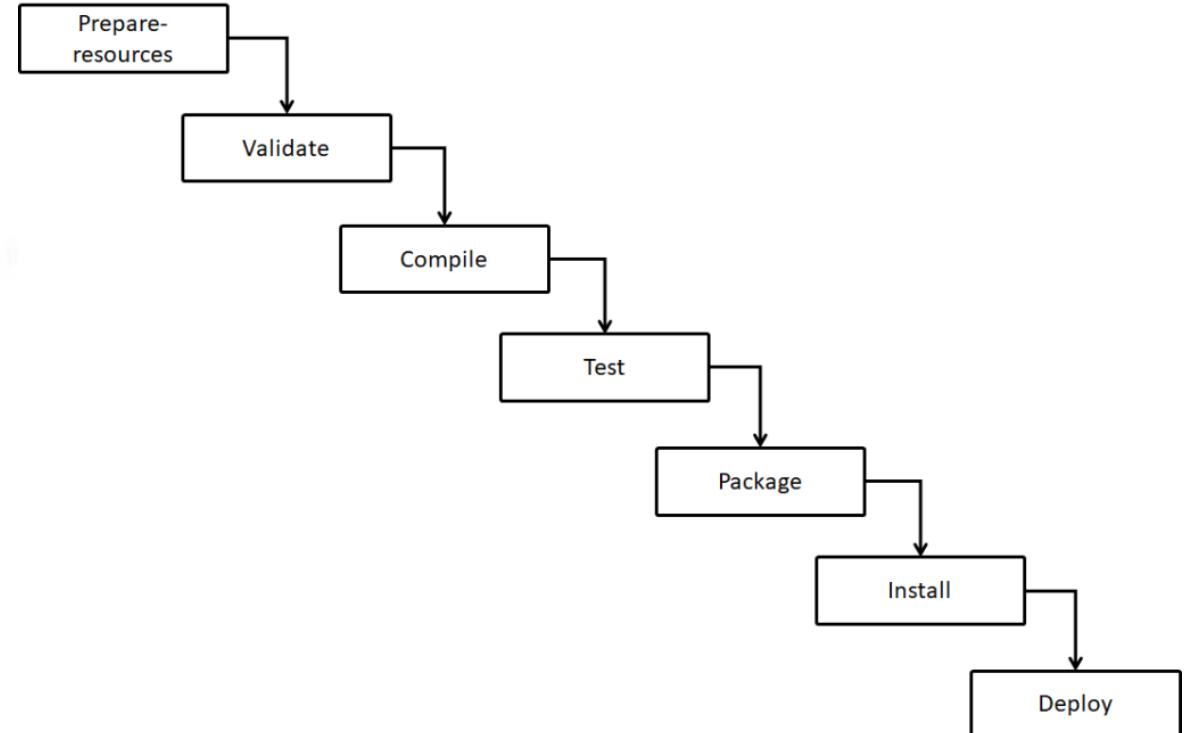
- **Default**: es el principal y es responsable del despliegue del proyecto
- **Clean**: Limpia y remueve todos los archivos generados por el build previo.
- **Site**: Crea la documentación del proyecto



Default

Cuando ejecutamos una fase
Se ejecutan todas las precedentes

mvn <PHASE>



ciclo de vida default

validate: esta fase valida que toda la información del proyecto esté disponible y sea correcta

verify: esta fase ejecuta controles para verificar que el paquete es válido

compile: esta fase compila el código fuente

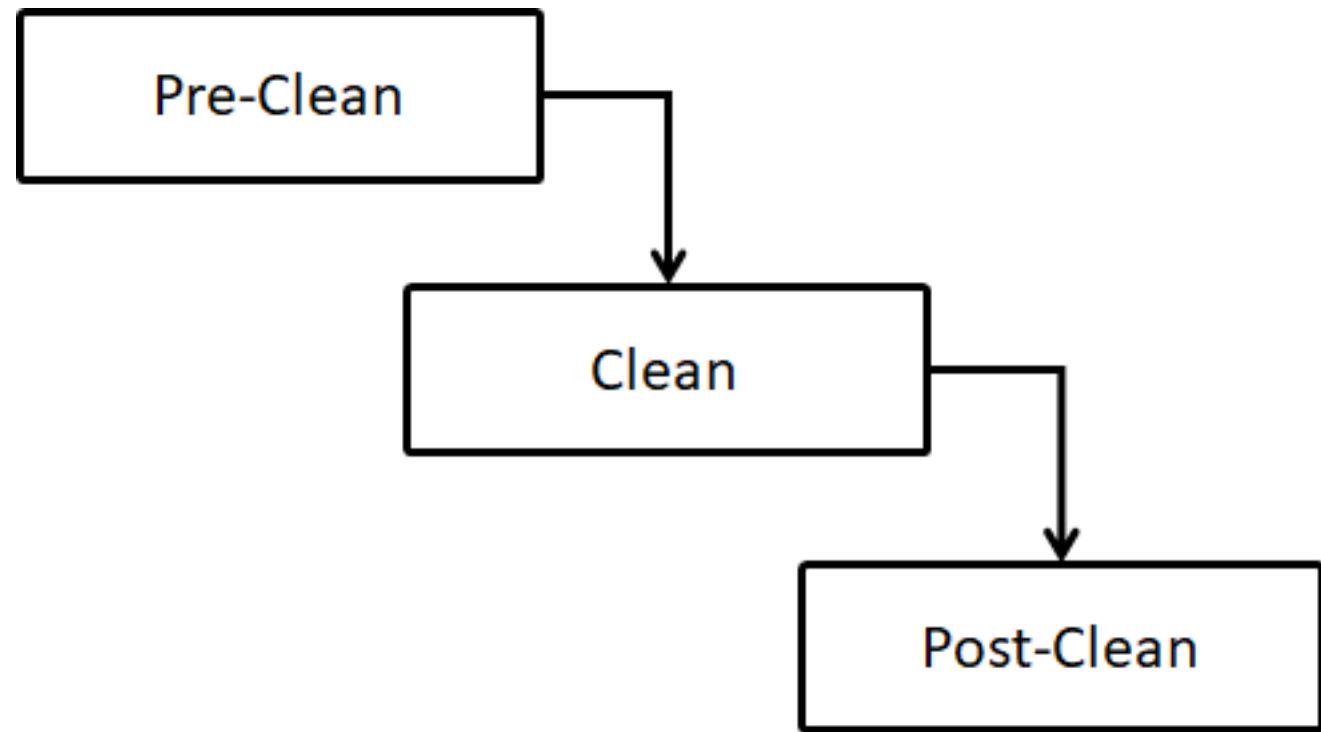
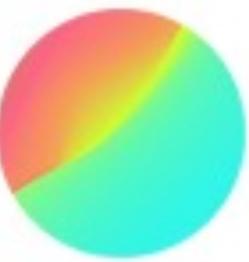
install: esta fase instala el paquete en el repositorio local

test: esta fase ejecuta pruebas unitarias

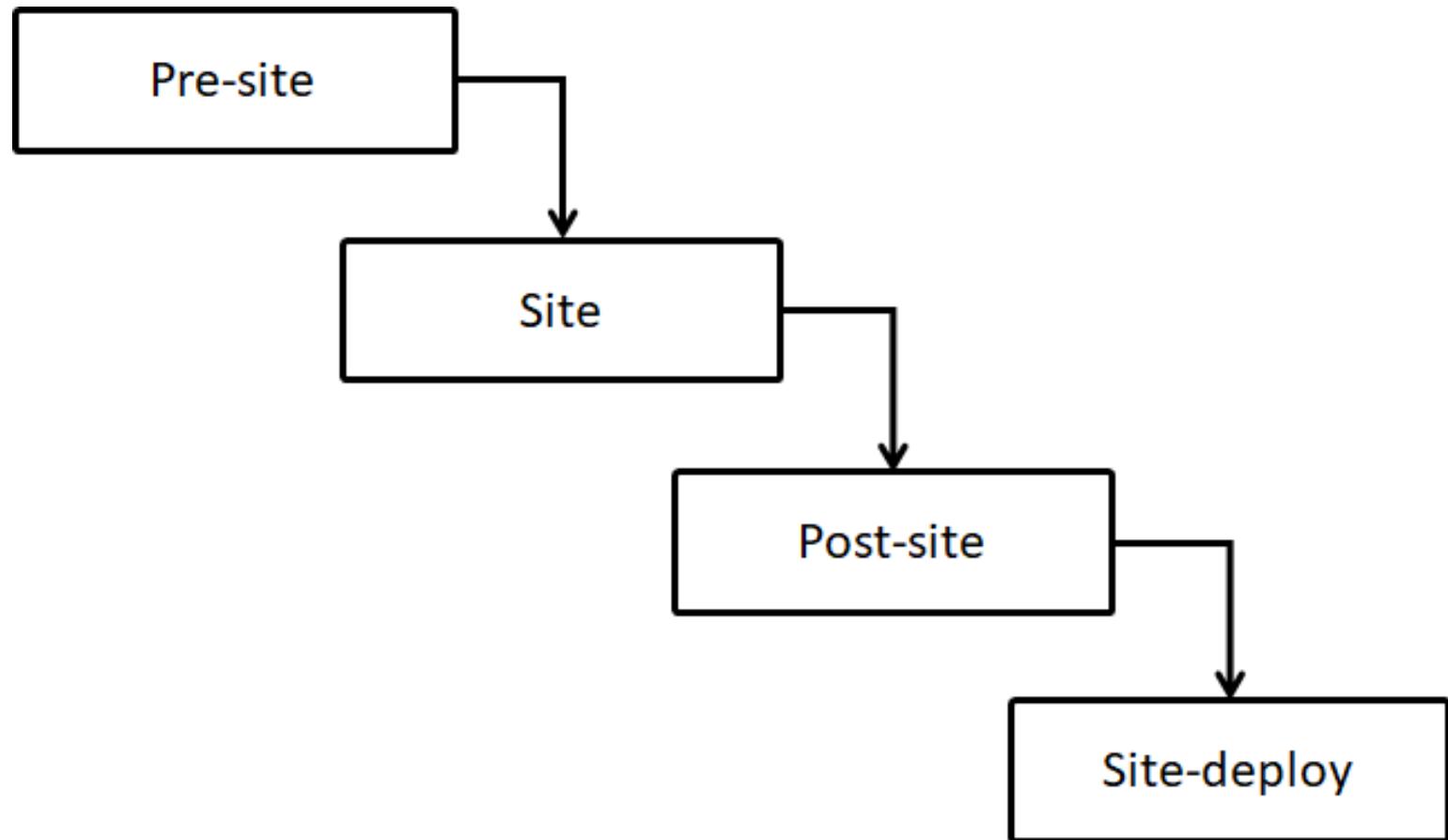
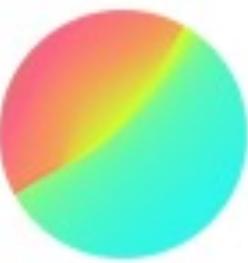
deploy: esta fase instala el paquete final en el repositorio configurado

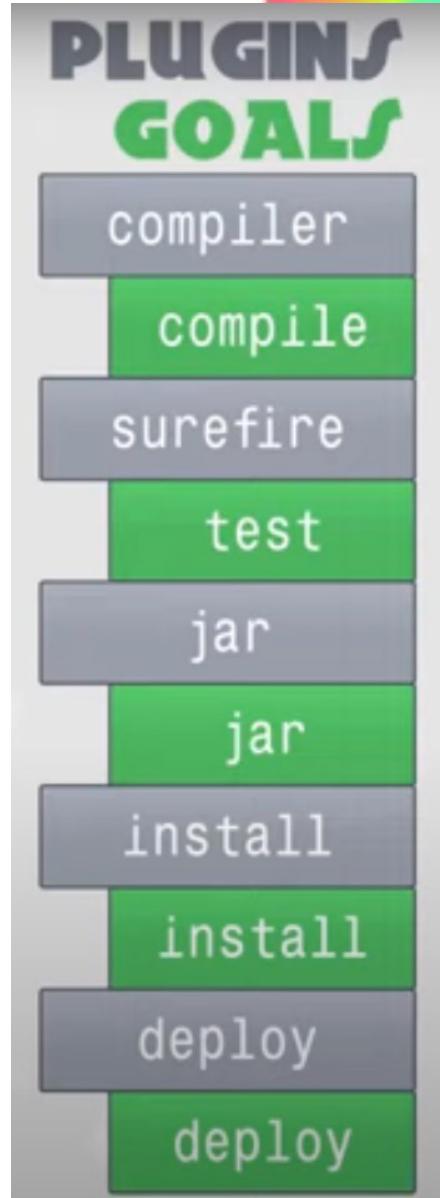
package: esta fase empaqueta el código compilado en su formato de distribución

Clean



Site

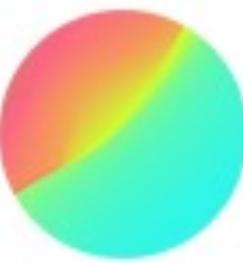




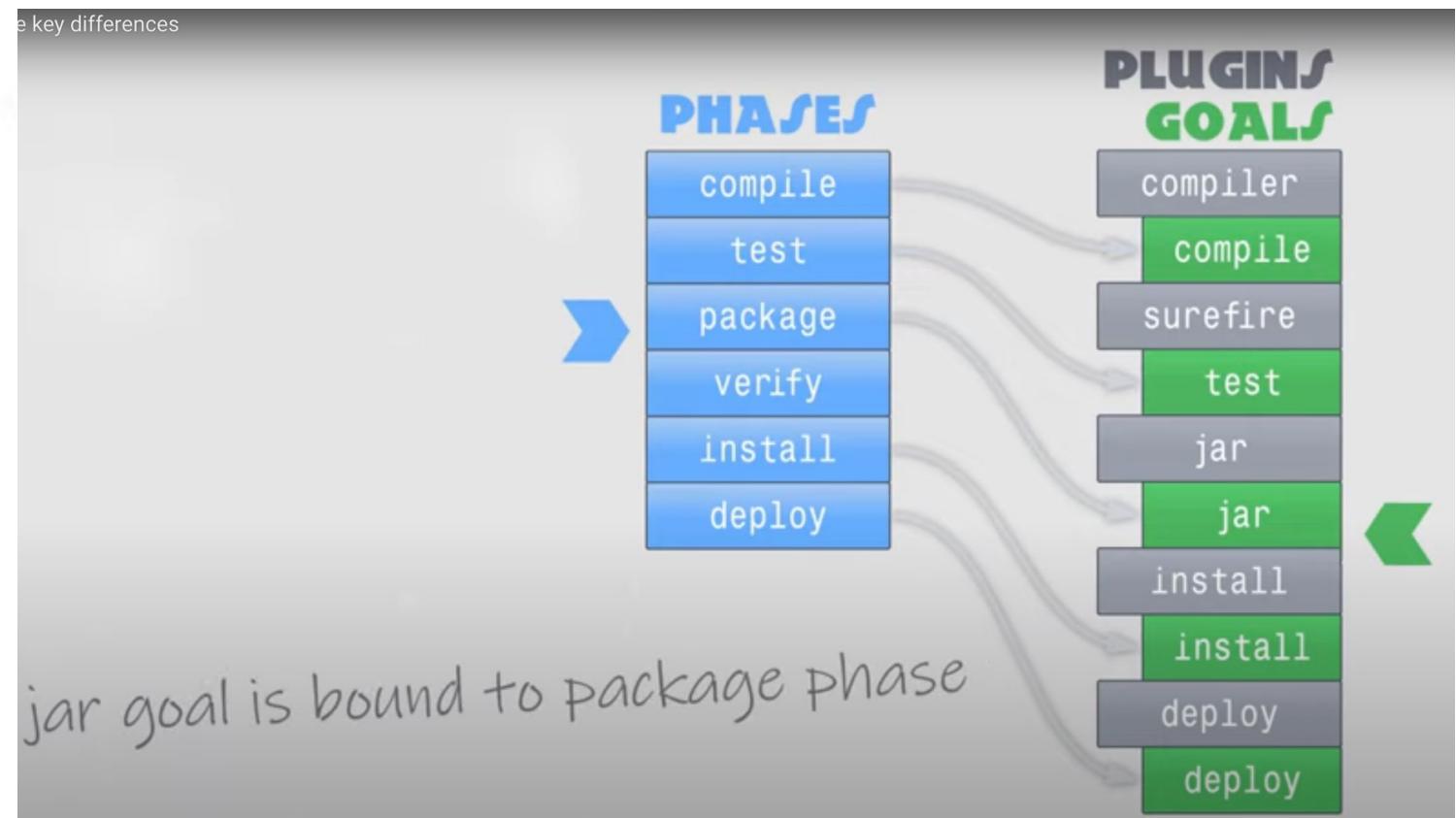
Conceptos Goals etc

- MOJO, Maven Plain Old Java Object, con un solo método **void execute**
- Goal, método java, contenido en una clase MOJO
- Para que un goal este disponible en la construcción es necesario que el POM tenga el correspondiente plugin.
- Plugin, es un fichero jar que contiene uno o mas goals
- Además de los plugins incluidos por defecto se pueden usar 3er party plugins

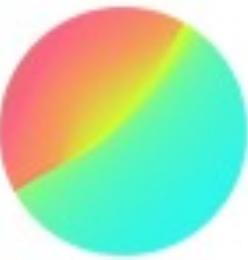
Conceptos Goals etc



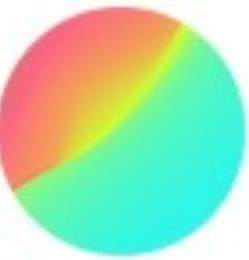
- Phase, es un grupo de goals, puede contener, uno, muchos o ninguno



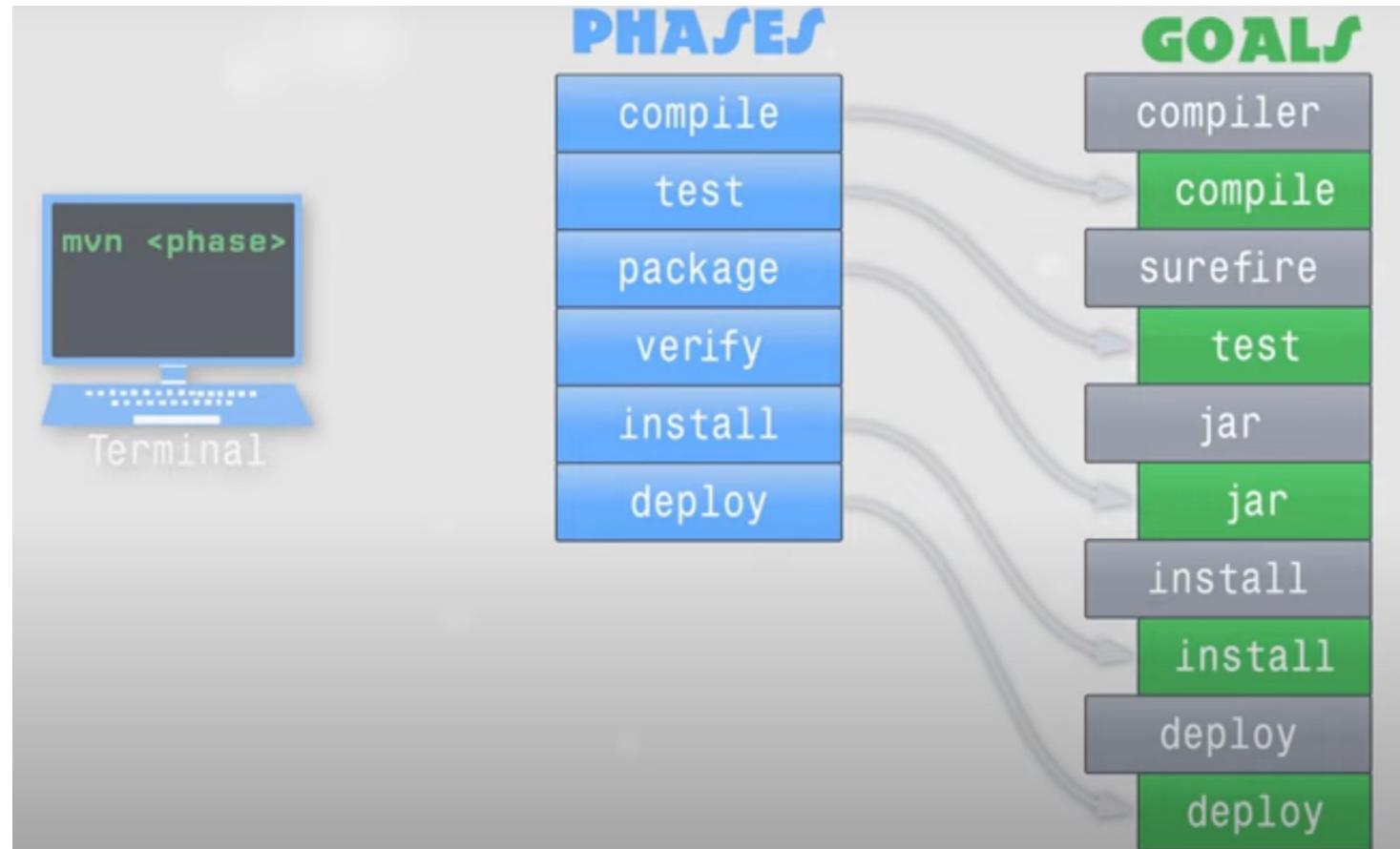
Conceptos Goals etc



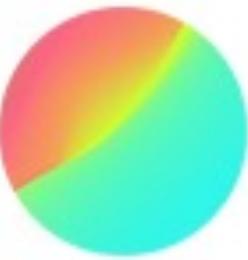
- Build Lifecycle es una secuencia ordenada de Phases. La idea es que contenga todo lo necesario para construir un proyecto.
- El Default Lifecycle cubre todo desde validación a inicialización o deployment y esta compuesto de 23 fases, pero las mas importantes son las que vemos en la siguiente PPT.



Fases importantes del Default Build LifeCycle

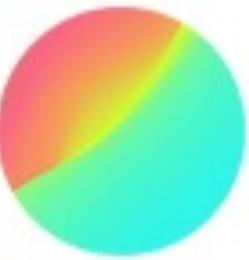


Lifecycles

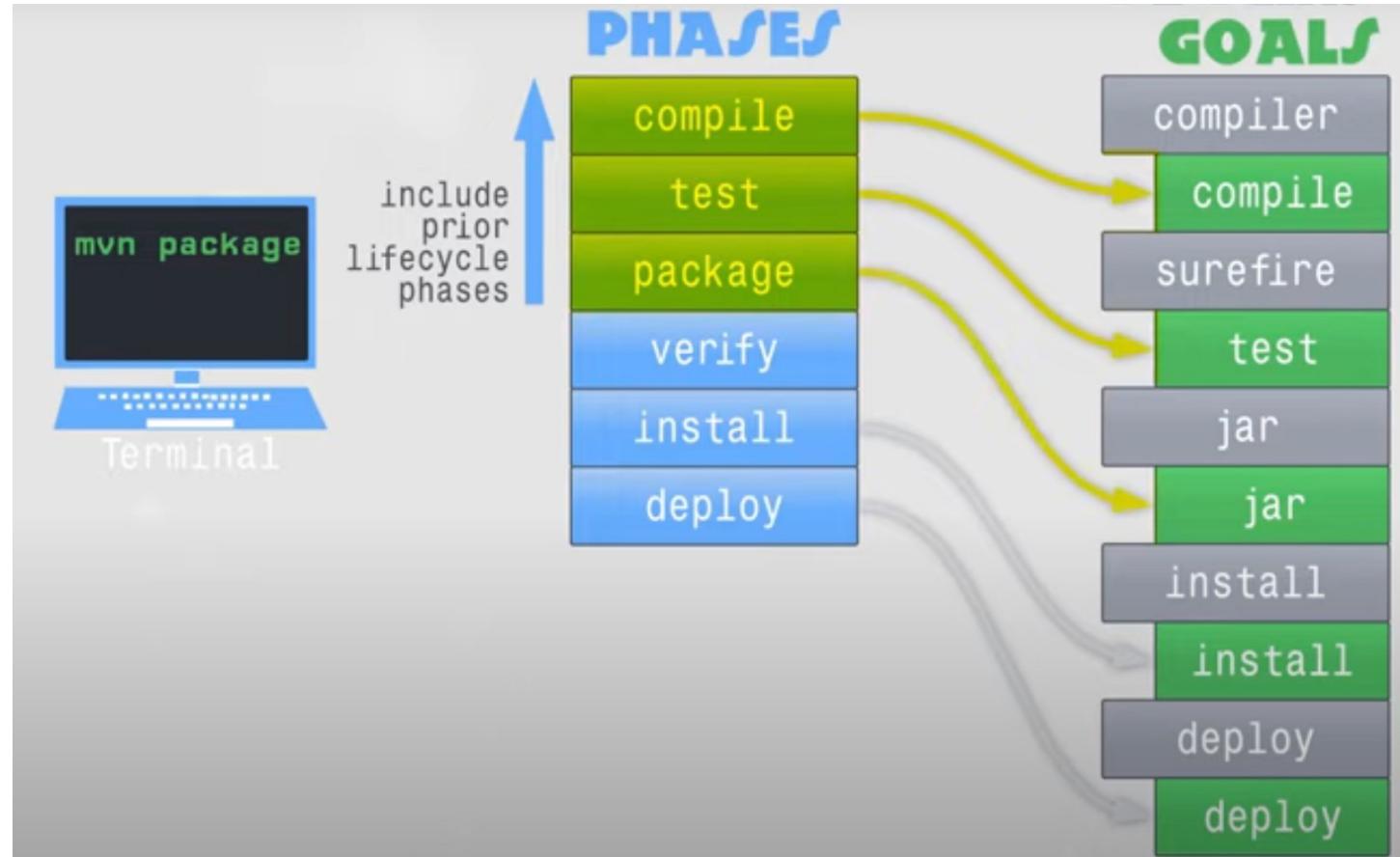


- Default
- Clean: limpia los .class
- Site: documenta el sitio

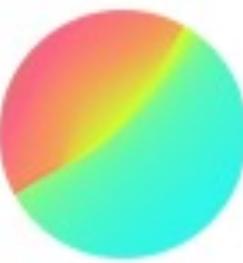
La Menza Perelló®. Todos los derechos reservados.



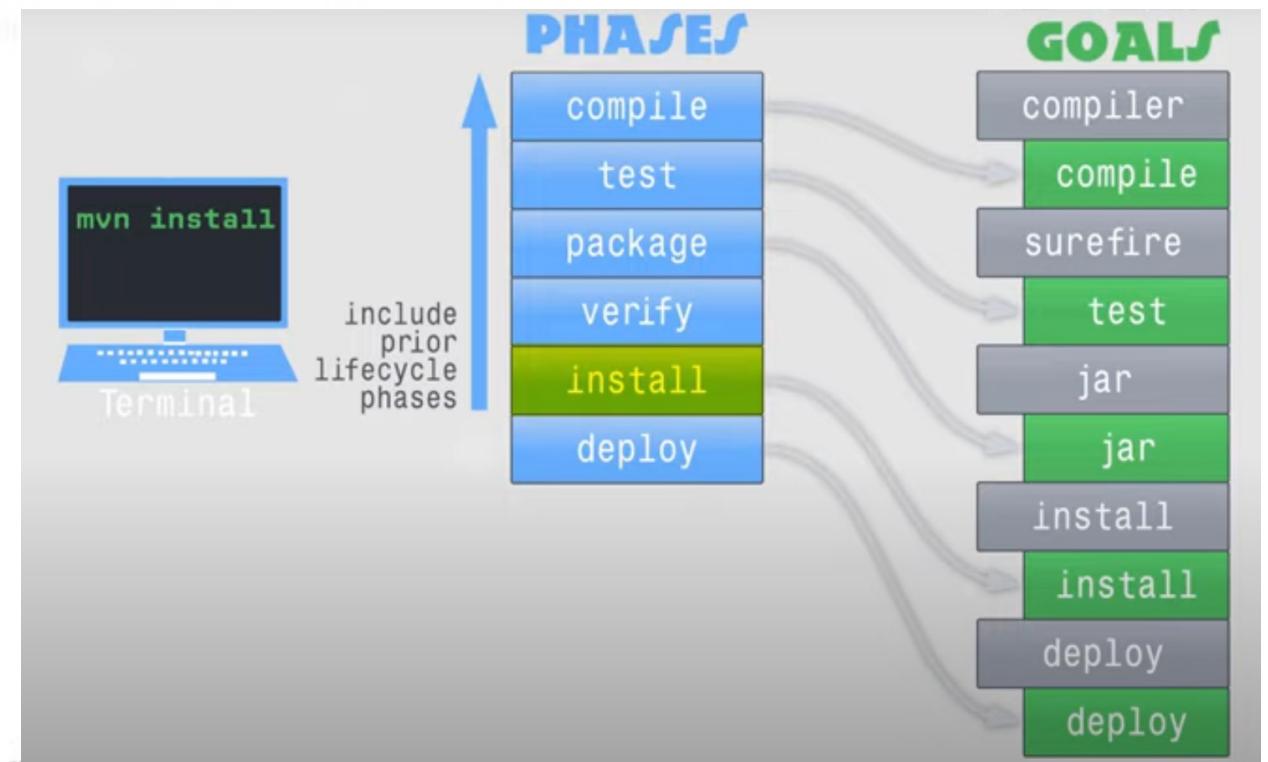
Si ejecuto mvn package ejecuta los anteriores

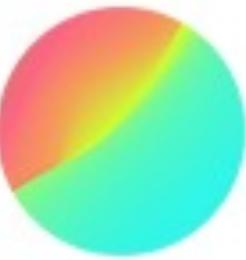


Necesidad del Build Lifecycle

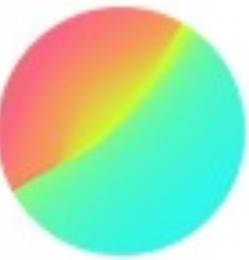


- Si hago **mvn install:install** me dará un error
- Esta es la manera correcta:



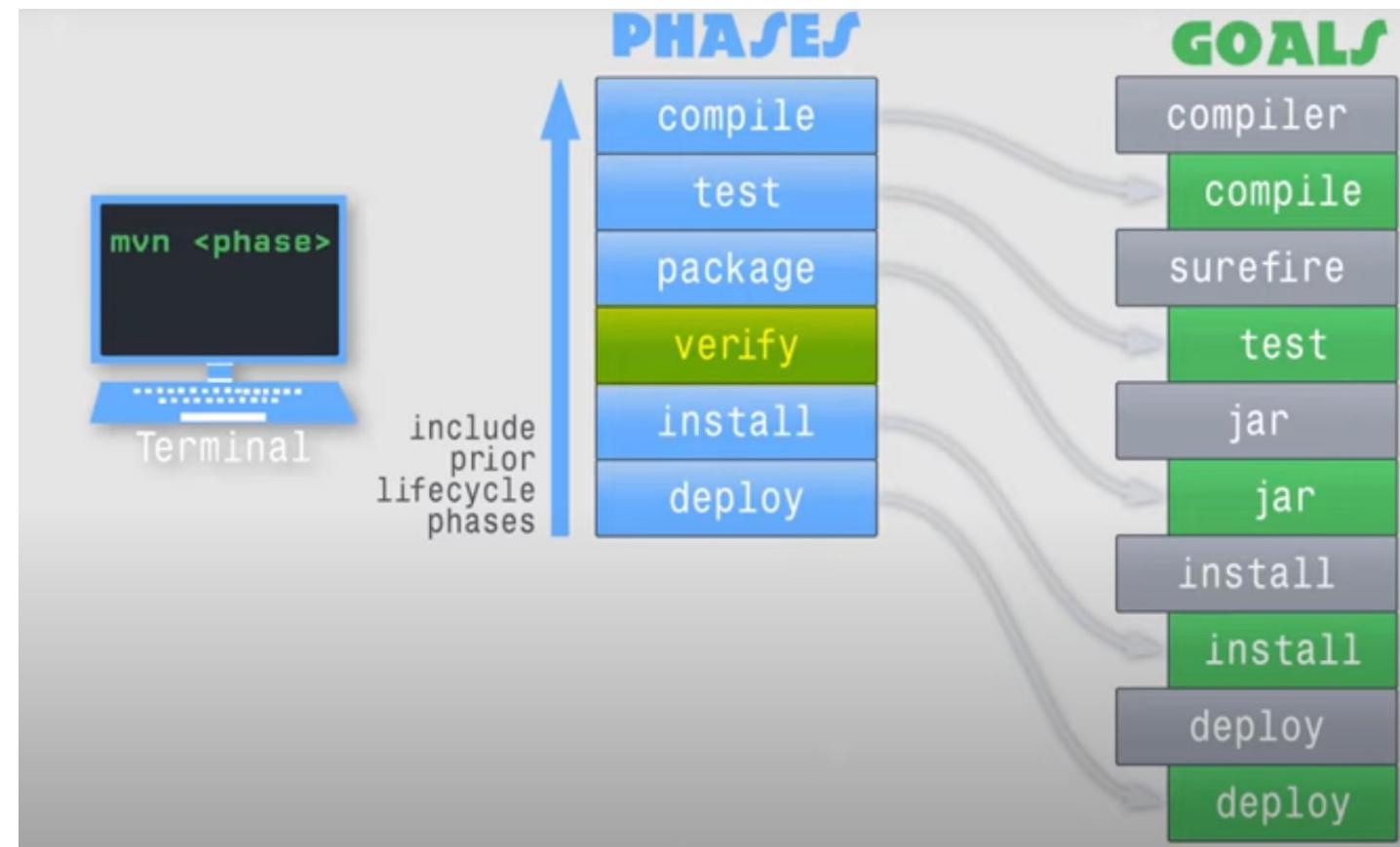


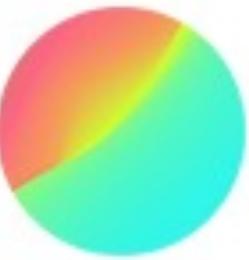
Demo 1



No todas las fases contienen un goal

- Verify no tiene

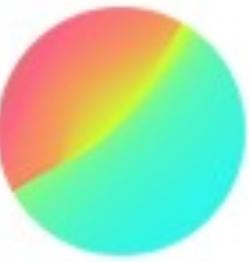




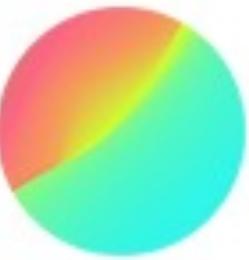
Usando Verify

```
</dependencies>
<build>
    <plugins>
        <plugin>
            <artifactId>maven-failsafe-plugin</artifactId>
            <version>2.22.2</version>
            <executions>
                <execution>
                    <goals>
                        <goal>integration-test</goal>
                        <goal>verify</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

binds to
verify phase



Demo 2



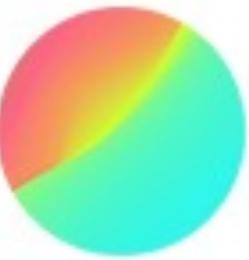
Fases y Goals

Cada fase (phase) es una secuencia de objetivos (goals), y cada objetivo es responsable de una tarea específica.

Cuando ejecutamos una fase, todos los goals vinculados a esta fase se ejecutan en orden.

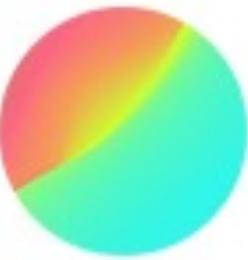
Estas son algunas de las fases y los objetivos predeterminados vinculados a ellas:

- *compiler:compile* – the *compile* goal from the *compiler* plugin is bound to the *compile* phase
- *compiler:testCompile* is bound to the *test-compile* phase
- *surefire:test* is bound to the *test* phase
- *install:install* is bound to the *install* phase
- *jarjar* and *war:war* is bound to the *package* phase



Arquitectura de Maven

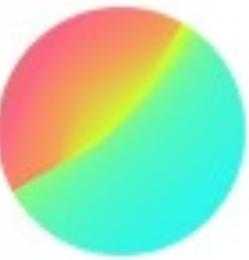
Repositorios



Local
Es el repositorio en el sistema de archivos de su computadora

Remoto
Es el repositorio desde donde se descargan los archivos Maven requeridos

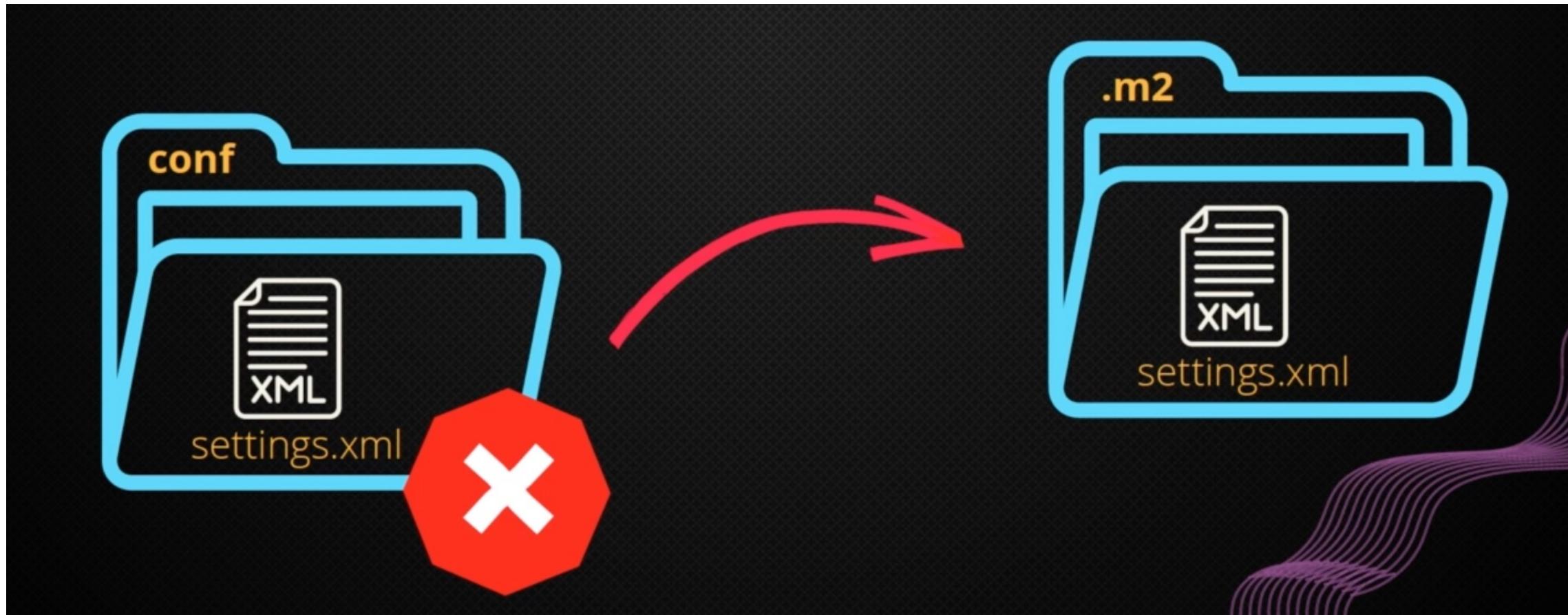
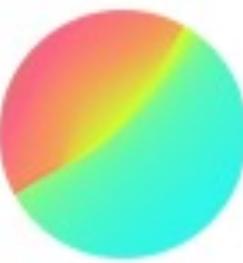
Espejos
Son repositorios administrados, como por ejemplo Nexus y Artifactory

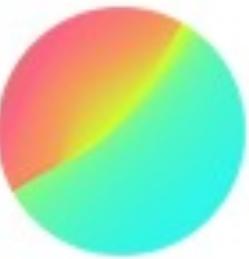


Cambiar el repositorio local



El settings de .m2 tiene prioridad





Cambiar el repositorio local

```
< >  untitled ✓  
1  <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0" xmlns:xsi="  
    http://www.w3.org/2001/XMLSchema-instance"  
2  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 https://maven.apache.org/xsd/  
    settings-1.0.0.xsd">  
3  
4      <localRepository>/Users/jose/Documents/CURSOS/MAVEN/Practicas/temp_repository</localRepository>  
5  
6  </settings>  
7
```

Copio el settings que esta en /conf, lo edito como arriba y le pongo la dirección del repo y a ese archivo lo meto en .m2 como settings.xml

Demo: Hacerlo y tirar la ejecución de un proyecto para ver que ahora los jars se van adonde dice el settings.xml del .m2



Ejecutar Maven mediante un servidor proxy

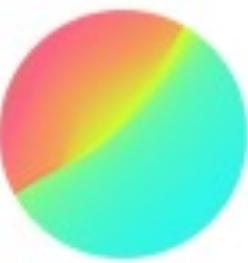
```
<proxies>
  <proxy>
    <id>mi-proxy</id>
    <active>true</active>
    <protocol>http</protocol>
    <host>proxy.mi-organizacion.com</host>
    <port>8080</port>
    <username>usuario-proxy</username>
    <password>contraseña</password>
    <nonProxyHosts>*.google.com|ibiblio.org</nonProxyHosts>
  </proxy>
</proxies>
```

settings.xml

<nonProxyHosts> los host que no deben pasar por el proxy. Por ejemplo los hosts de nuestra red local.

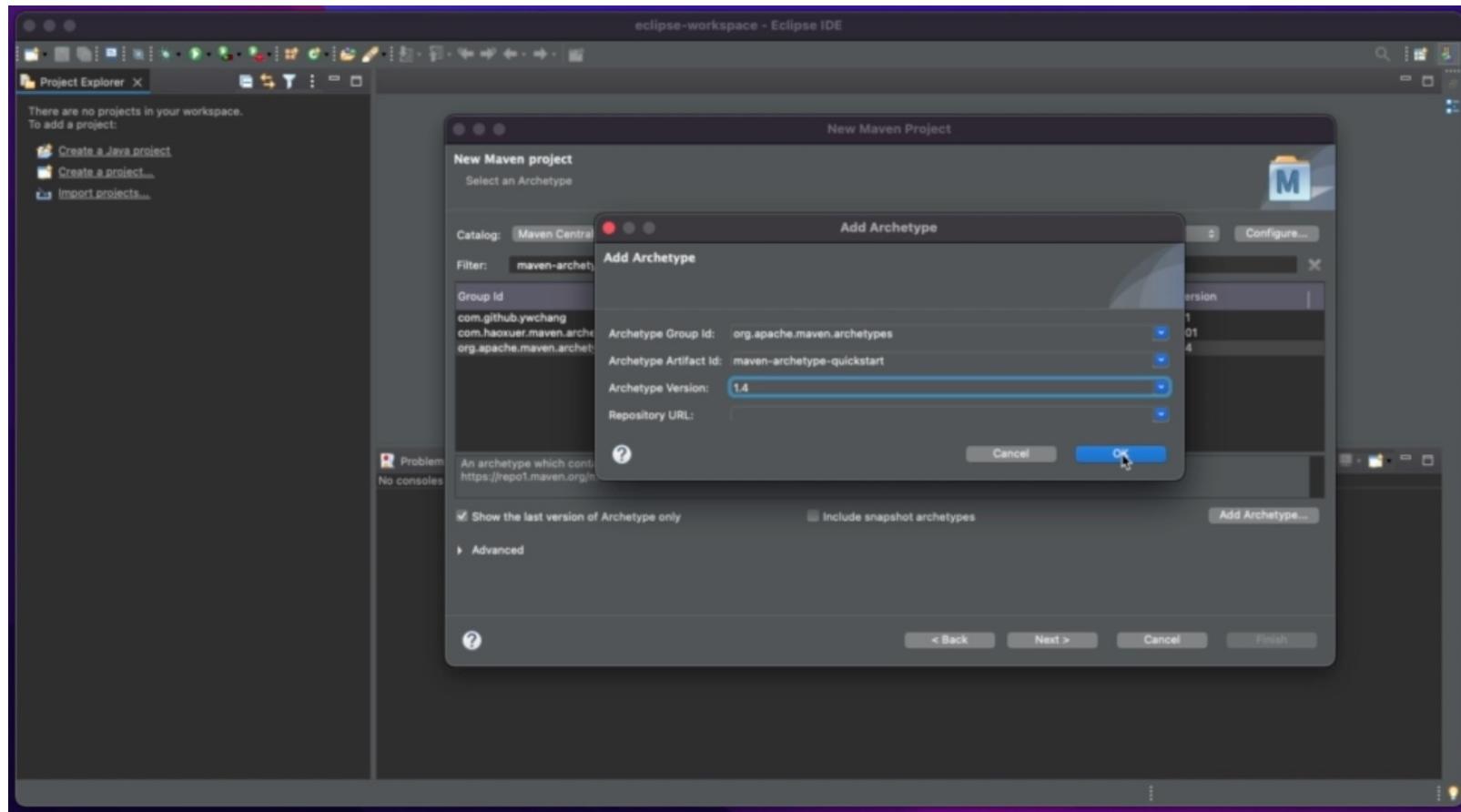
Si de repente estoy fuera de la zona proxy desactivo el proxy con <active>false

Contenido de /target

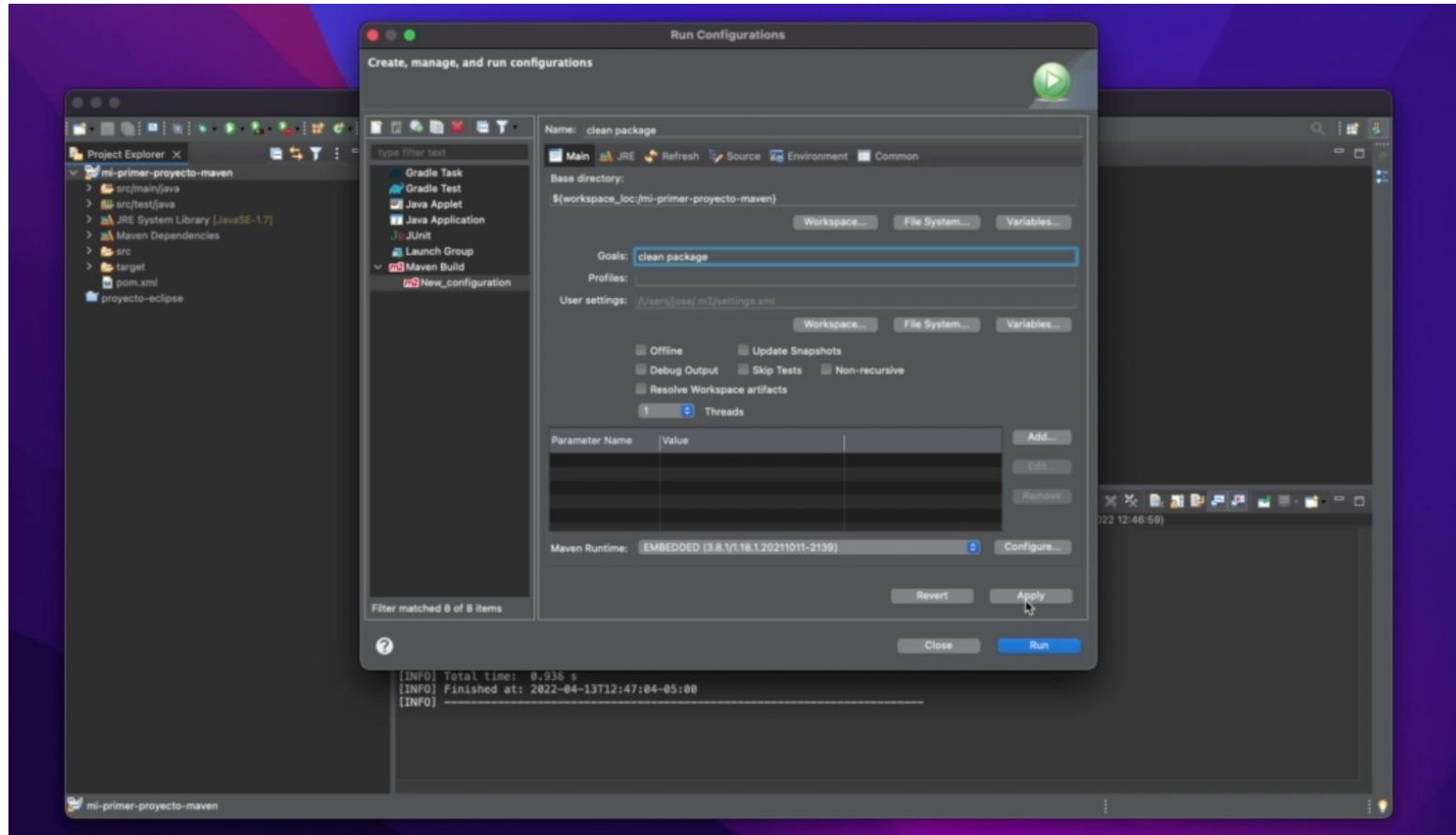


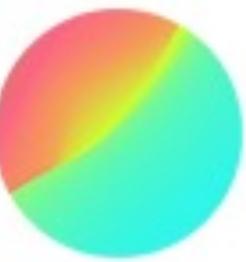
-  **La carpeta classes:** los archivos fuente compilados se colocan en esta carpeta.
-  **La carpeta test-classes:** los archivos fuente de prueba compilados están disponibles en esta carpeta.
-  **La carpeta surefire-reports:** los informes de prueba se colocan por separado en esta carpeta.
-  **El archivo .jar de salida:** es el artefacto generado del proyecto.
-  **Las carpetas maven-archiver y maven-status:** contienen información utilizada por Maven durante la compilación.

Si no encuentra un arquetipo en Eclipse



Agregar configuración para ejecutar goals específicos en Eclipse





Elementos del POM



Elementos básicos del POM

dependencies

Define todos los proyectos de los que depende el proyecto

```
<dependencies>...</dependencies>
```

parent

Se utiliza para indicar una relación, específicamente una relación padre-hijo

```
<parent>...</parent>
```

properties

Son marcadores de posición. Se puede acceder a sus valores en cualquier parte del archivo pom usando \${key}

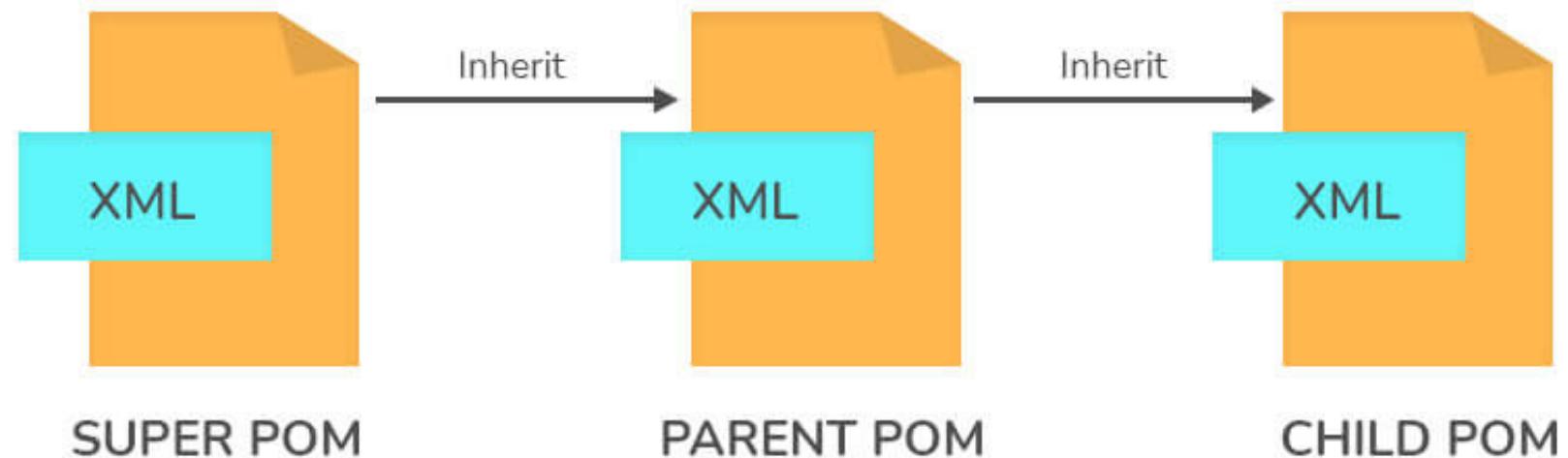
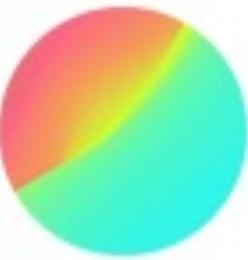
```
<properties>...</properties>
```

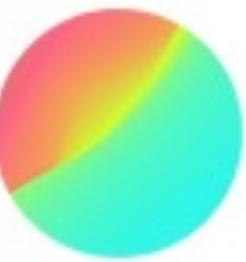
modules

Se emplea para identificar los módulos

```
<modules>...</modules>
```

Super-pom





Configuraciones de settings.xml

localRepository

```
<localRepository>${user.home}/.m2/repository</localRepository>
```

offline

Esta configuración indica si Maven debe operar en modo fuera de línea

```
<offline>...</offline>
```

proxies

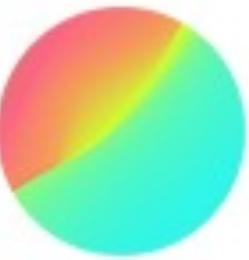
Se emplea para indicar la información de los servidores proxy

```
<proxies>...</proxies>
```

mirror

Se emplea para descargar dependencias de un repositorio espejo

```
<mirror>...</mirror>
```



Configuraciones de settings.xml

repositories

Permite configurar los tipos de repositorios releases o snapshots

```
<repositories>...</repositories>
```

pluginRepositories

Almacena bibliotecas de complementos y archivos asociados

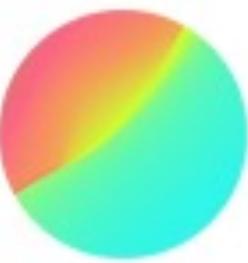
```
<pluginRepositories>
  ...
</pluginRepositories>
```

servers

Es empleado para almacenar usuarios, contraseñas, llaves privadas, etc

```
<servers>...</servers>
```

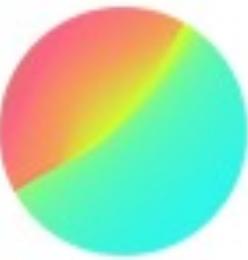
Perfiles de compilacion



Profile dev
Configuraciones de ambiente de desarrollo

Profile pro
Configuraciones de ambiente productivo

Alcance de perfiles



Por proyecto
Definido en el propio POM

Por usuario
Definido en la configuración de Maven en
el archivo
`%USER_HOME%/.m2/settings.xml`

Global
Definido en la configuración global de
Maven en el archivo
`#{maven.home}/conf/settings.xml`

Formas de definir perfiles

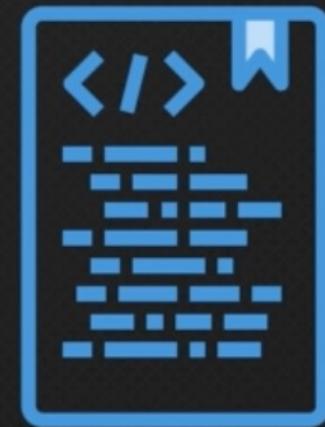


Archivo POM

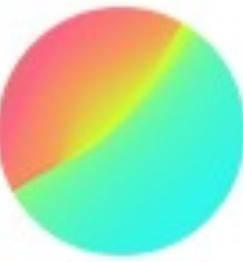


pom.xml

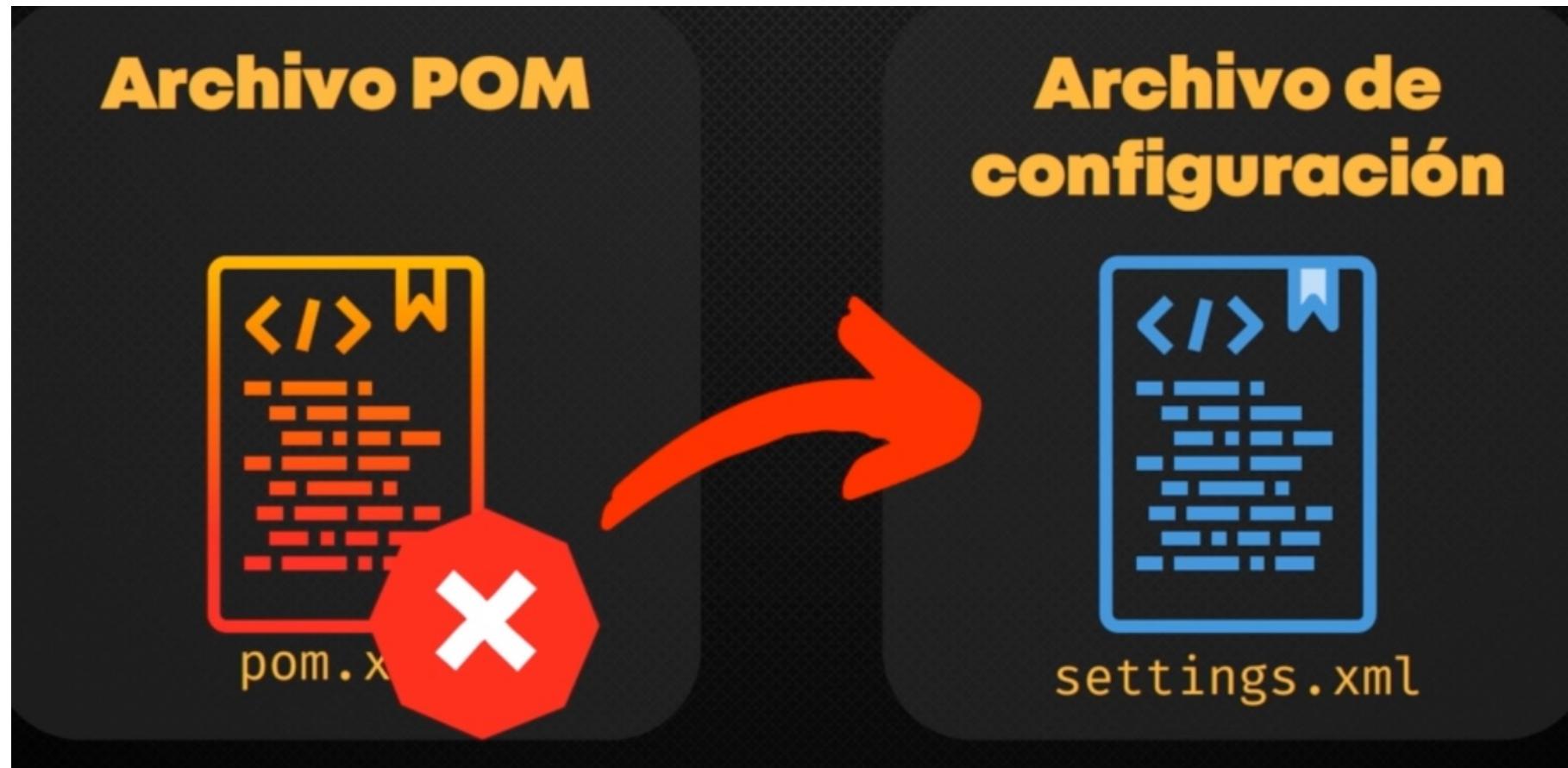
Archivo de configuración

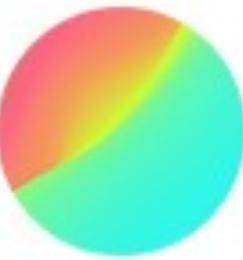


settings.xml



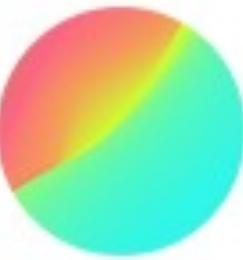
Lo que se define en `settings.xml` tiene prioridad sobre lo de POM (con el mismo id)





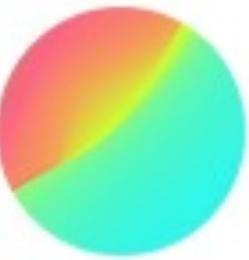
Perfil definido en POM

```
<profile>
    <id>...</id>
    <activation>...</activation>
    <build>...</build>
    <modules>...</modules>
    <repositories>...</repositories>
    <pluginRepositories>...</pluginRepositories>
    <dependencies>...</dependencies>
    <reporting>...</reporting>
    <dependencyManagement>...</dependencyManagement>
    <distributionManagement>...</distributionManagement>
</profile>
```



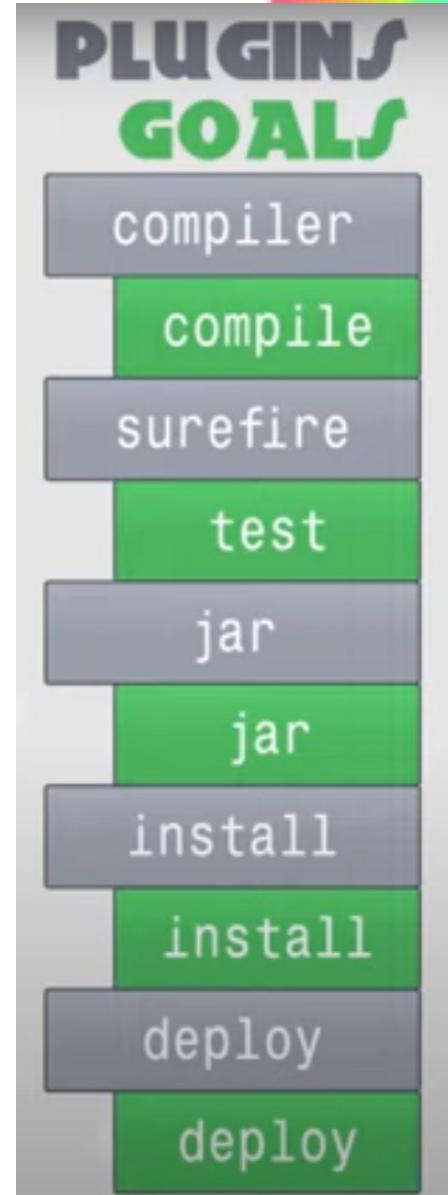
Perfil definido en settings

```
<profile>
  <id>...</id>
  <activation>...</activation>
  <repositories>...</repositories>
  <pluginRepositories>...</pluginRepositories>
  <properties>...</properties>
</profile>
```



Tom Gregory

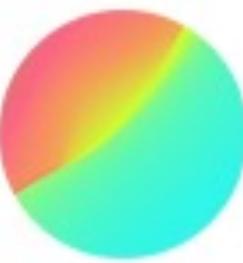
EXCELENTE
Ver para Gradle y Maven



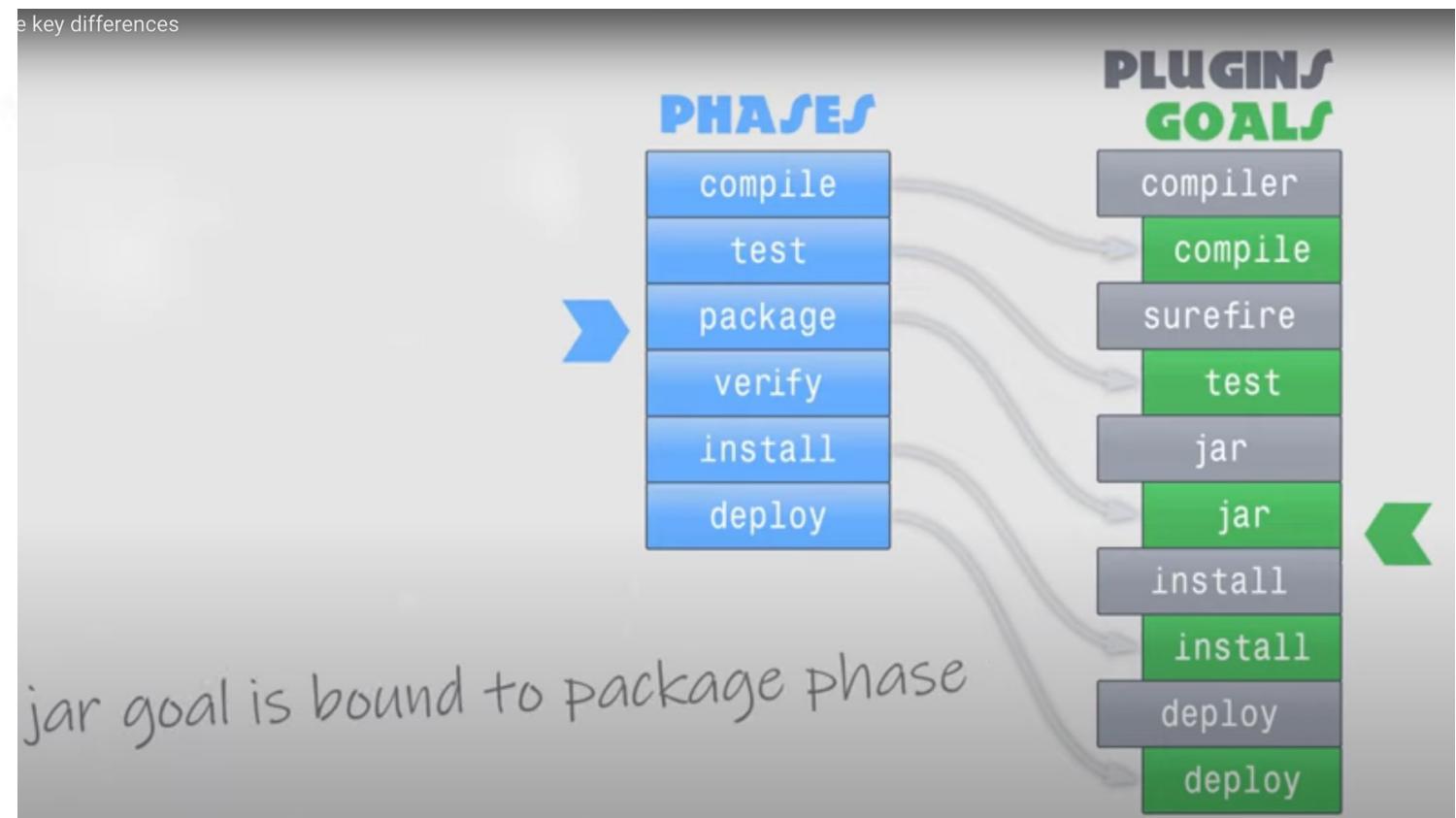
Conceptos Goals etc

- MOJO, Maven Plain Old Java Object, con un solo método **void execute**
- Goal, método java, contenido en una clase MOJO
- Para que un goal este disponible en la construcción es necesario que el POM tenga el correspondiente plugin.
- Plugin, es un fichero jar que contiene uno o mas goals
- Además de los plugins incluidos por defecto se pueden usar 3er party plugins

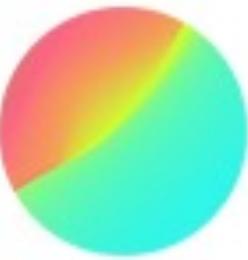
Conceptos Goals etc



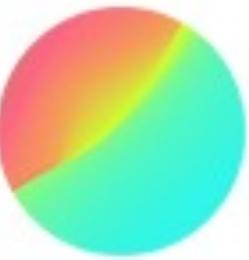
- Phase, es un grupo de goals, puede contener, uno, muchos o ninguno



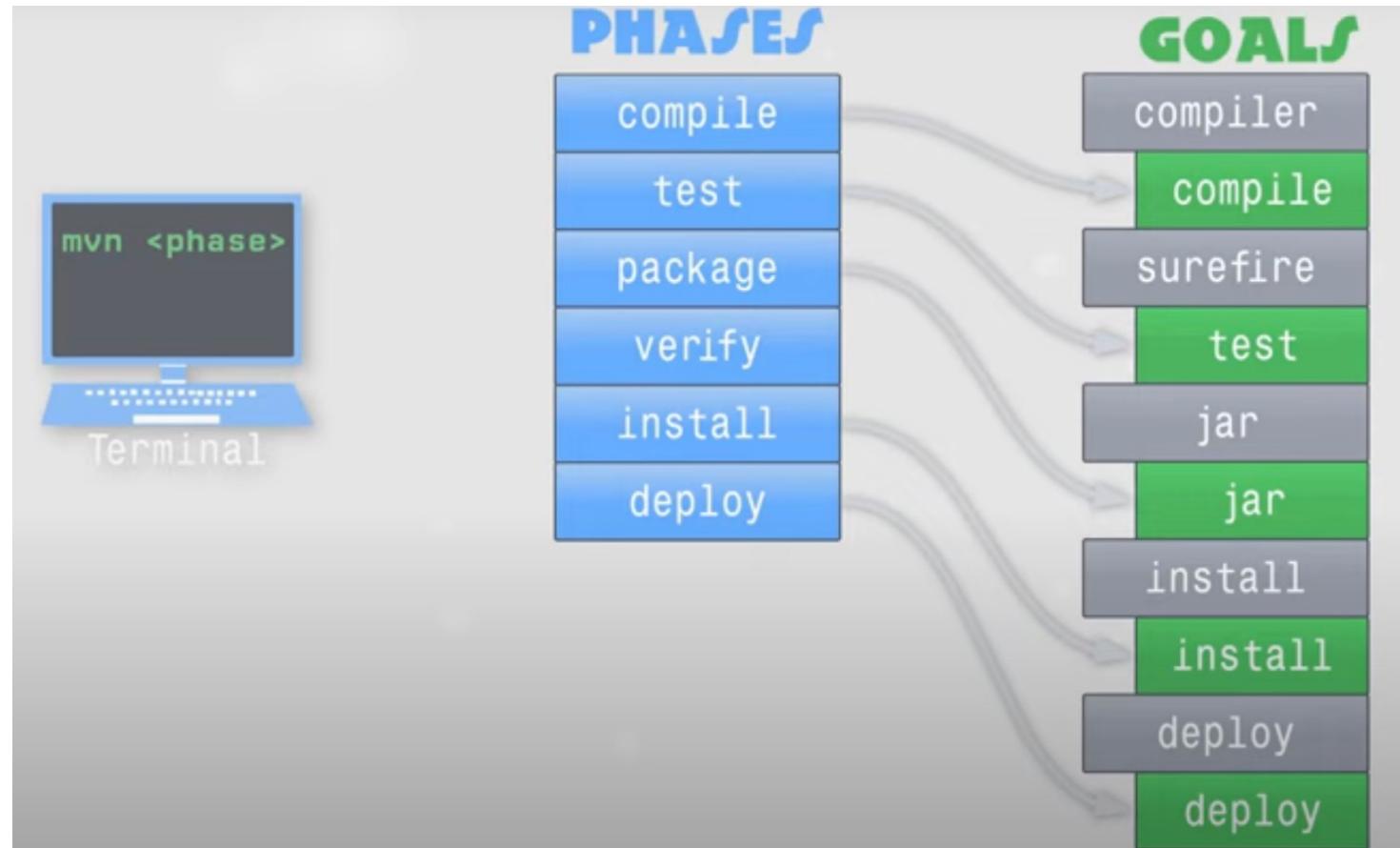
Conceptos Goals etc



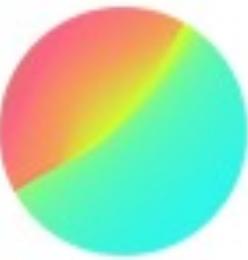
- Build Lifecycle es una secuencia ordenada de Phases. La idea es que contenga todo lo necesario para construir un proyecto.
- El Default Lifecycle cubre todo desde validación a inicialización o deployment y esta compuesto de 23 fases, pero las mas importantes son las que vemos en la siguiente PPT.



Fases importantes del Default Build LifeCycle

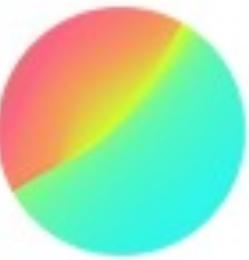


Lifecycles

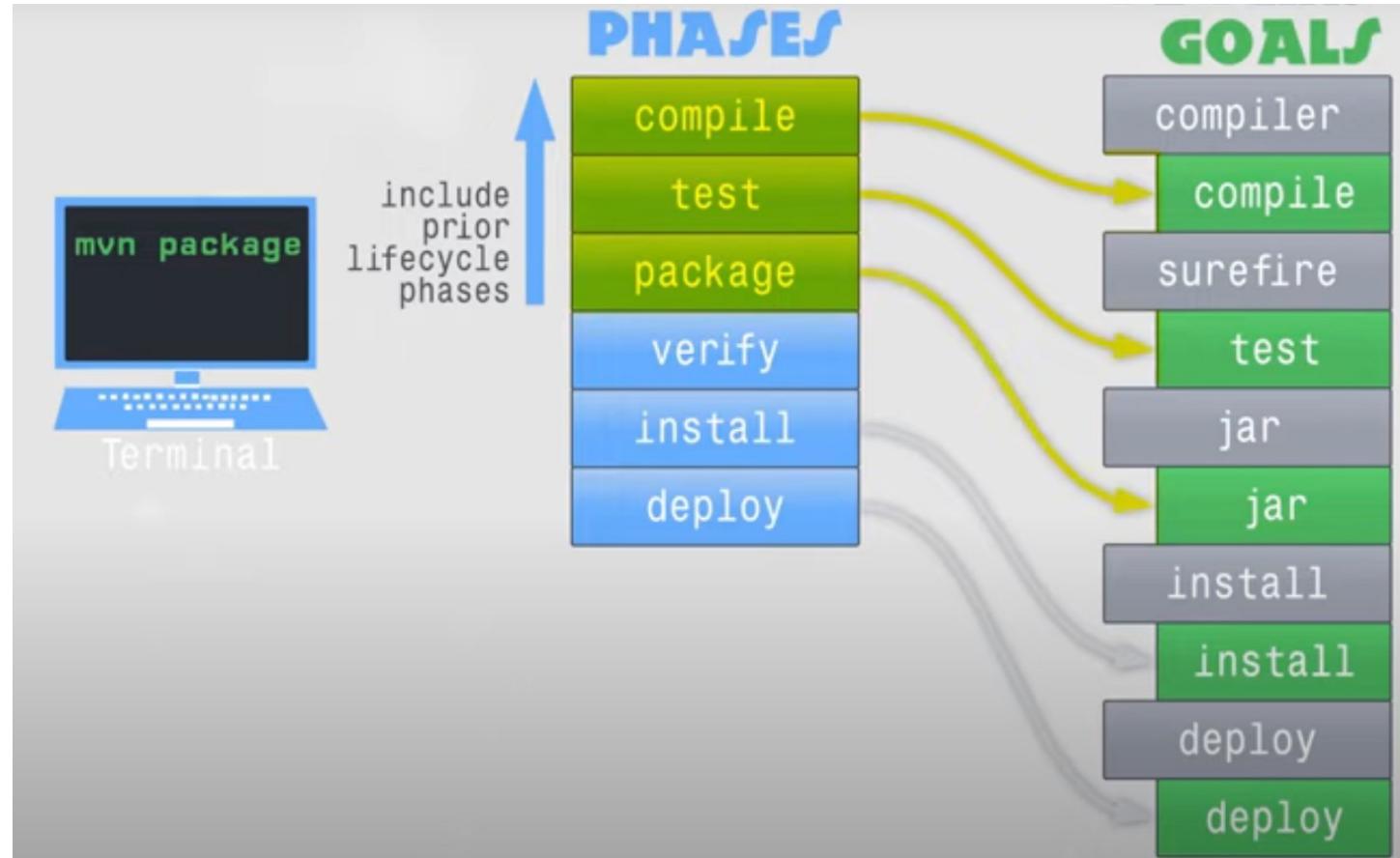


- Default
- Clean: limpia los .class
- Site: documenta el sitio

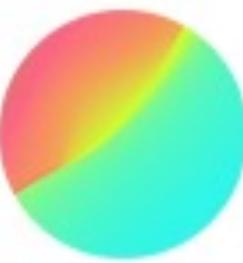
La Menza Perelló®. Todos los derechos reservados.



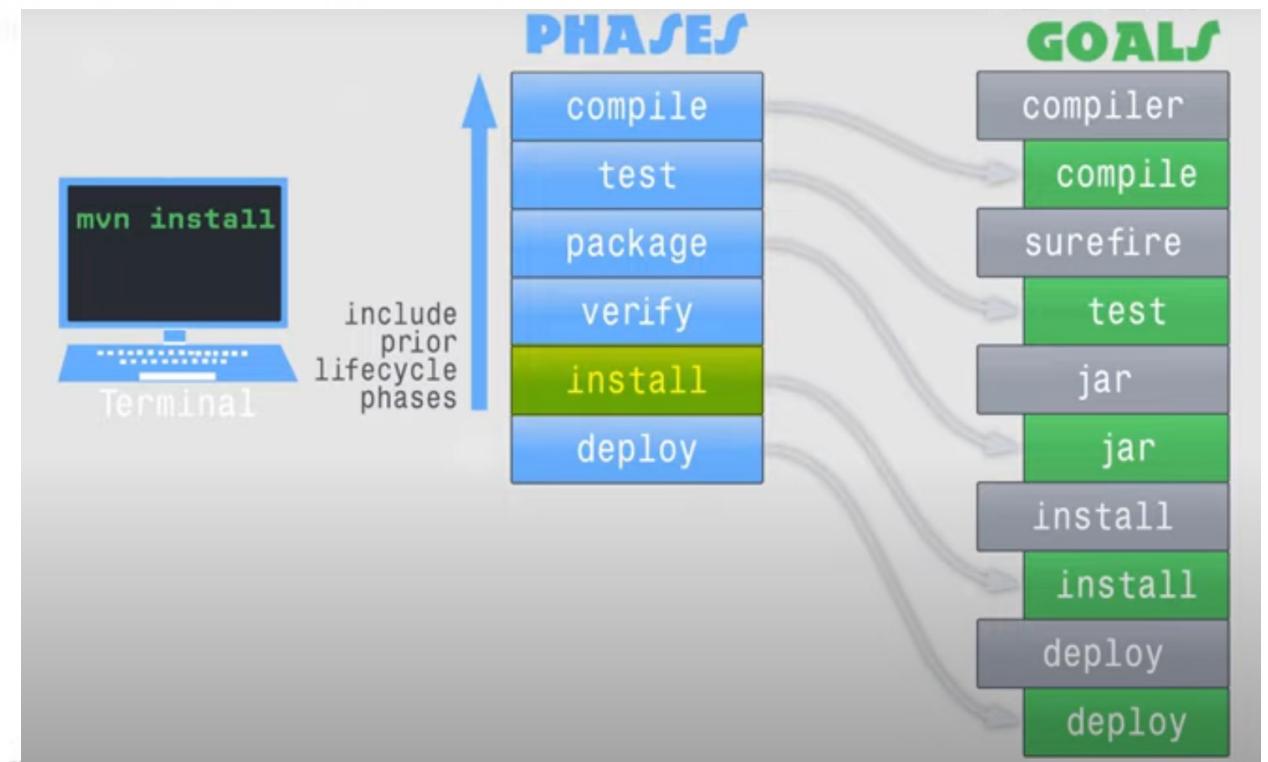
Si ejecuto mvn package ejecuta los anteriores

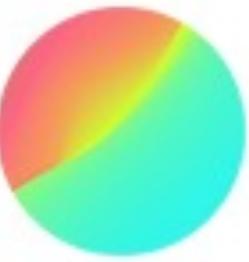


Necesidad del Build Lifecycle

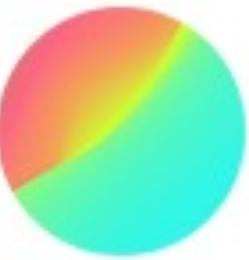


- Si hago **mvn install:install** me dará un error
- Esta es la manera correcta:



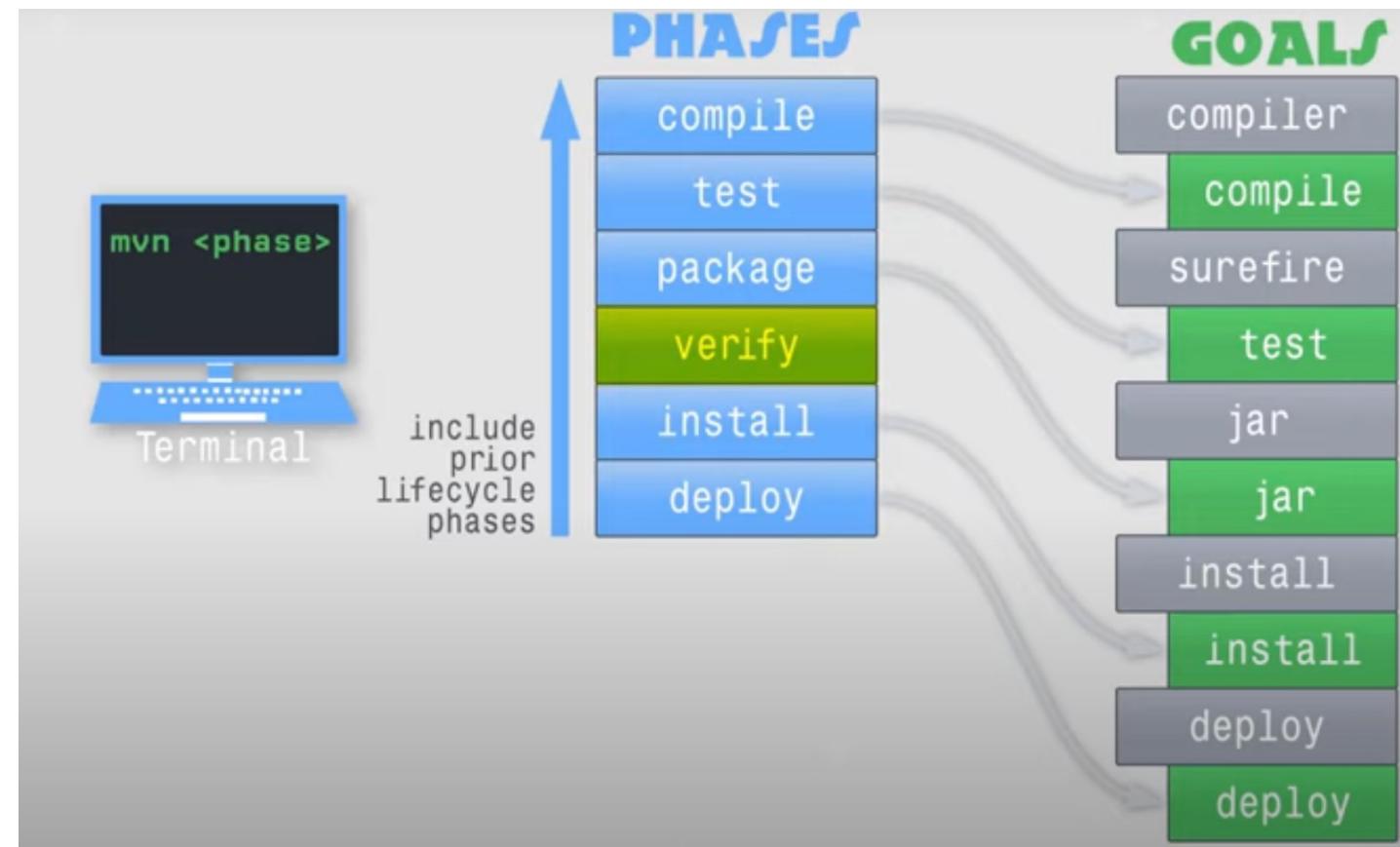


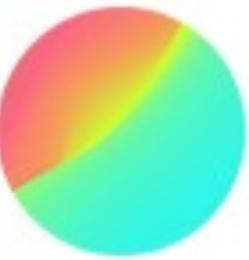
Demo 2



No todas las fases contienen un goal

- Verify no tiene

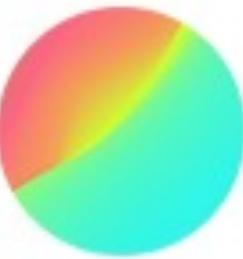




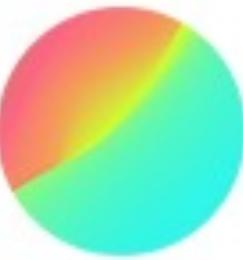
Usando Verify

```
</dependencies>
<build>
    <plugins>
        <plugin>
            <artifactId>maven-failsafe-plugin</artifactId>
            <version>2.22.2</version>
            <executions>
                <execution>
                    <goals>
                        <goal>integration-test</goal>
                        <goal>verify</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

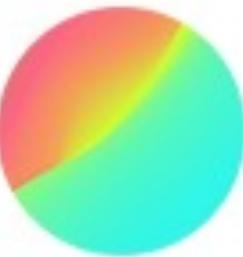
binds to
verify phase



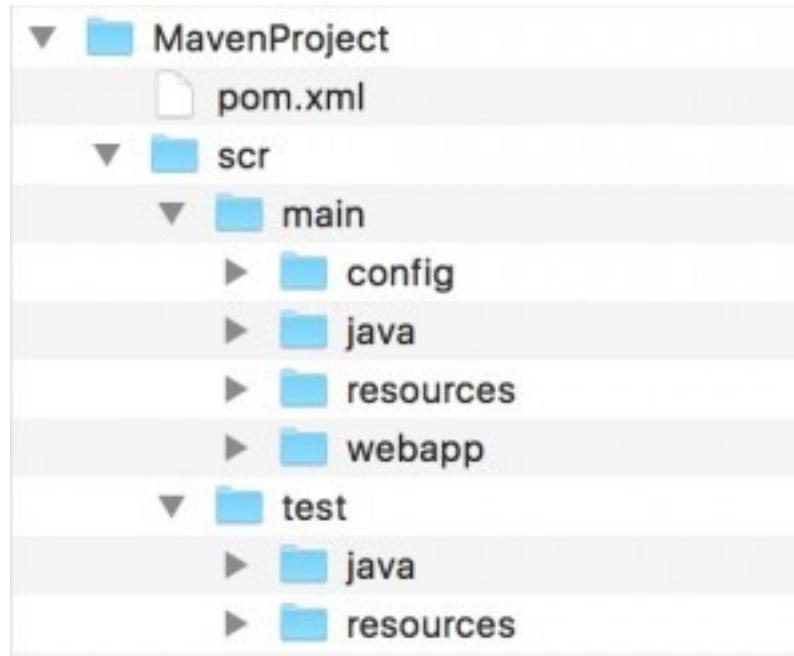
La Menza Perelló®. Todos los derechos reservados.



Mavenizar un proyecto legado

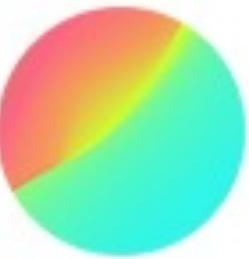


Estructura de un proyecto Maven



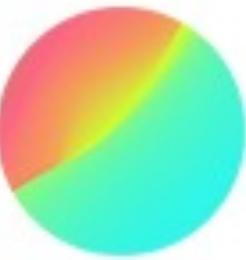
Cada una de las carpetas de la jerarquía contiene lo siguiente:

- **src**: código fuente, ficheros de configuración, recursos y demás. Es decir, todo salvo el directorio de salida (target) y el pom.xml
- **main**: desarrollo de la aplicación, independientemente de los test
- **test**: desarrollo de los test de la aplicación
- **config**: ficheros de configuración en el proyecto
- **java**: clases java que contienen el código fuente de la aplicación
- **resources**: recursos que se incluirán en el empaquetado y serán usados por la aplicación, como pueden ser ficheros de texto o scripts
- **webapp**: contiene todos los ficheros correspondientes a la propia aplicación web que no sean código java



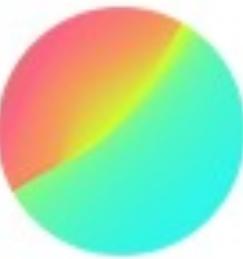
Formas de migrar

- Semi automática: Eclipse – Configure > Convert to Maven Project
- Manual: Crear Proyecto en Maven y mover clases y dependencias



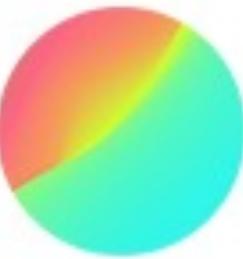
La Menza Perelló®. Todos los derechos reservados.

Variables



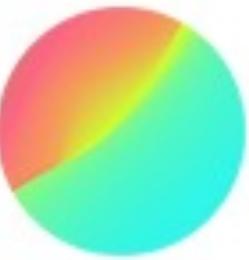
VARIABLES PARA VERSIONES DE LIBRERIAS

```
1 <properties>
2     <spring.version>3.5.7.RELEASE</spring.version>
3 </properties>
4
5 <dependencies>
6
7     <dependency>
8         <groupId>org.springframework</groupId>
9         <artifactId>spring-core</artifactId>
10        <version>${spring.version}</version>
11    </dependency>
12
13    <dependency>
14        <groupId>org.springframework</groupId>
15        <artifactId>spring-context</artifactId>
16        <version>${spring.version}</version>
17    </dependency>
18
19 </dependencies>
```

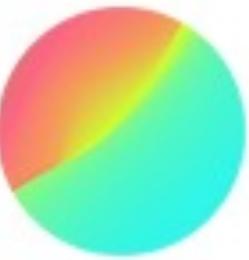


VARIABLES PARA REFERENCIAR UBICACIONES

```
1 <properties>
2   <project.theme.name>default</project.theme.name>
3   <project.resources.build.folder>
4     ${project.build.directory}/temp-resources/${project.theme.name}/
5   </project.resources.build.folder>
6 </properties>
7
8 <plugin>
9   <groupId>org.apache.maven.plugins</groupId>
10  <artifactId>maven-resources-plugin</artifactId>
11  <version>2.5</version>
12  <executions>
13    <execution>
14      <id>copy-resources</id>
15      <goals>
16        <goal>copy-resources</goal>
17      </goals>
18      <configuration>
19        <outputDirectory>
20          ${project.resources.build.folder}
21        </outputDirectory>
22    //...
```



Perfiles en Maven

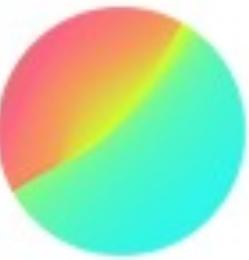


Perfiles

```
<profiles>
  <profile>
    <id>dev</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <properties>
      <db.location>URL_to_dev_system</db.location>
      <logo.image>companylogo.png</logo.image>
    </properties>
  </profile>
  <profile>
    <id>production</id>
    <properties>
      <db.location>URL_to_prod_system</db.location>
      <logo.image>companylogo2.png</logo.image>
    </properties>
  </profile>
</profiles>
```

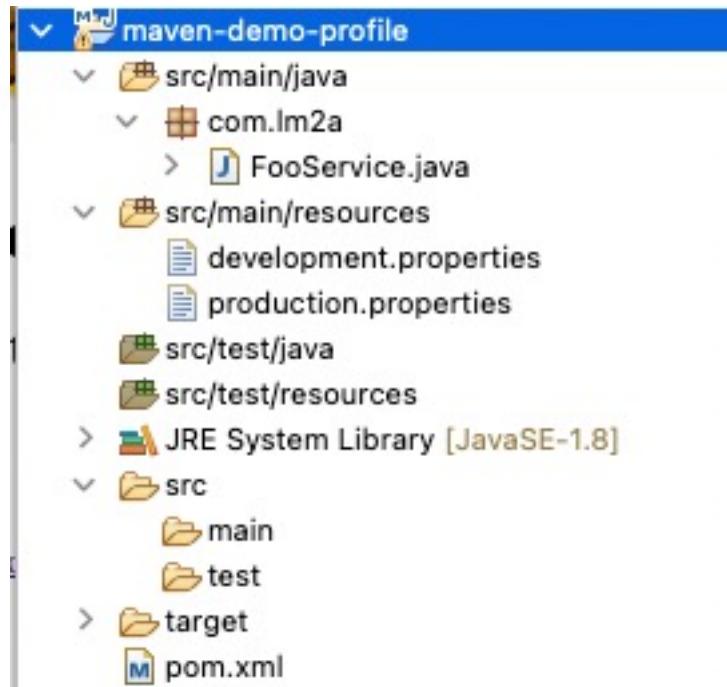
COPY

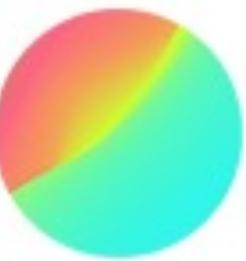
X



Creo un proyecto

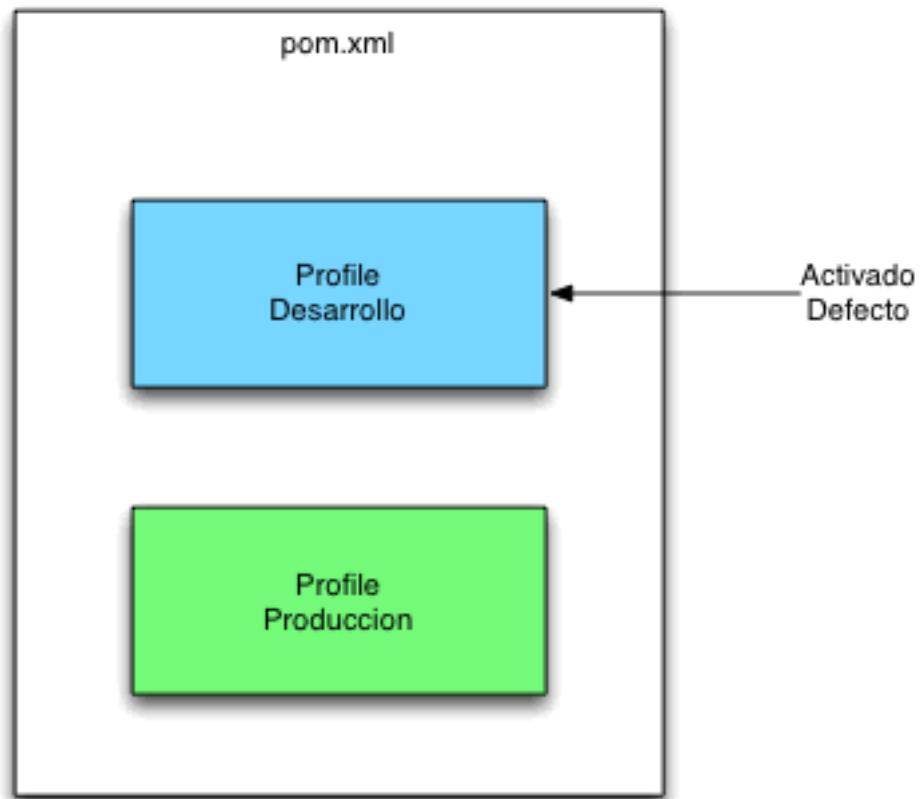
- Agrego dos ficheros .properties uno por cada entorno



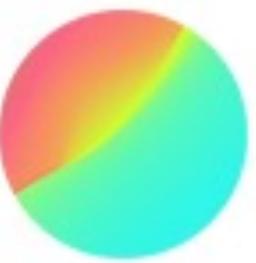


Perfiles Maven

- La idea es poder seleccionar el perfil correcto

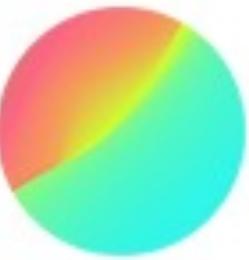


Agregar los perfiles al POM



```
https://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation)
1@<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2  <modelVersion>4.0.0</modelVersion>
3  <groupId>com.lm2a</groupId>
4  <artifactId>demo-profile</artifactId>
5  <version>0.0.1-SNAPSHOT</version>
6@  <properties>
7    <maven.compiler.target>1.8</maven.compiler.target>
8    <maven.compiler.source>1.8</maven.compiler.source>
9  </properties>
10 </project>
```

```
1@<project xmlns="http://maven.apache.org/POM/4.0.0"
2  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
4  <modelVersion>4.0.0</modelVersion>
5  <groupId>com.lm2a</groupId>
6  <artifactId>demo-profile</artifactId>
7  <version>0.0.1-SNAPSHOT</version>
8@  <properties>
9    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
10   <maven.compiler.source>1.8</maven.compiler.source>
11   <maven.compiler.target>1.8</maven.compiler.target>
12 </properties>
13@  <profiles>
14@    <profile>
15      <id>Production</id>
16@      <activation>
17        <activeByDefault>true</activeByDefault>
18      </activation>
19      <build>
20        <resources>
21          <resource>
22            <directory>src/main/resources</directory>
23@          <excludes>
24            <exclude>production.properties</exclude>
25          </excludes>
26        </resource>
27      </resources>
28    </build>
29  </profile>
30@    <profile>
31      <id>Development</id>
32      <build>
33        <resources>
34          <resource>
35            <directory>src/main/resources</directory>
36          <excludes>
37            <exclude>development.properties</exclude>
38          </excludes>
39        </resource>
40      </resources>
41    </build>
42  </profile>
43</profiles>
44</project>
```



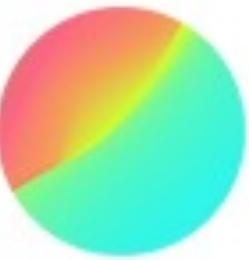
Perfiles basado en propiedades

```
<profile>
  <id>active-on-property-environment</id>
  <activation>
    <property>
      <name>environment</name>
    </property>
  </activation>
</profile>
```

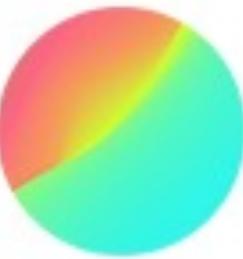
(1) Lo activo usando
mvn package -Denvironment

```
<property>
  <name>environment</name>
  <value>test</value>
</property>
```

(2) Loactivo usando
mvn package -Denvironment=test

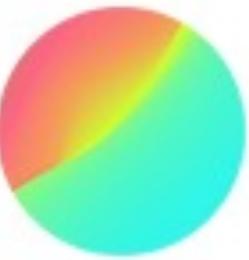


Maven properties

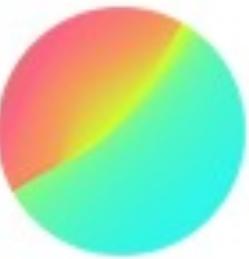


Principales propiedades de Maven

- `${project.basedir}` referencia el root del proyecto donde esta el pom.xml
- `${project.build.directory}` /target
- `${project.build.outputDirectory}` target/classes
- `${project.build.testOutputDirectory}` target/test-classes
- `${project.build.sourceDirectory}` src/main/java
- `${project.build.testSourceDirectory}` src/test/java
- `${project.build.finalName}` viene a ser: `${project.artifactId}-${project.version}`
- `${project.version}` versión del proyecto



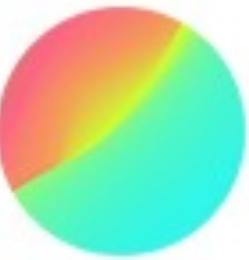
Despliegue de artefactos



El mojo deploy:deploy-file

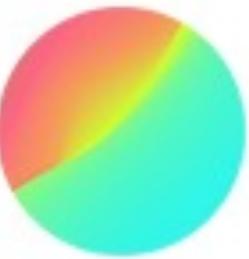
```
1. mvn deploy:deploy-file -Durl=file:///C:/m2-repo \
2.           -DrepositoryId=some.id \
3.           -Dfile=your-artifact-1.0.jar \
4.           [-DpomFile=your-pom.xml] \
5.           [-DgroupId=org.some.group] \
6.           [-DartifactId=your-artifact] \
7.           [-Dversion=1.0] \
8.           [-Dpackaging=jar] \
9.           [-Dclassifier=test] \
10.          [-DgeneratePom=true] \
11.          [-DgeneratePom.description="My Project Description"] \
12.          [-DrepositoryLayout=legacy]
```

Por línea de comandos



Incluyendo el despliegue en el POM

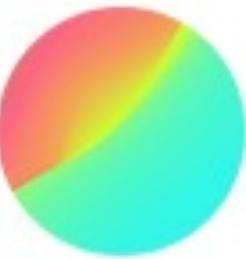
```
<plugin>
    <artifactId>maven-deploy-plugin</artifactId>
    <version>2.8.2</version>
    <executions>
        <execution>
            <id>deploy-file</id>
            <phase>deploy</phase>
            <goals>
                <goal>deploy-file</goal>
            </goals>
            <configuration>
                <file><!-- path-to-file --></file>
                <url><!-- url-of-the-repository-to-deploy --></url>
                <groupId><!-- group-id --></groupId>
                <artifactId><!-- artifact-id --></artifactId>
                <version><!-- version --></version>
                <packaging><!-- type-of-packaging --></packaging>
            </configuration>
        </execution>
    </executions>
</plugin>
```



Despliegue en Tomcat

1. En Tomcat nos aseguramos de tener un User y un password con rol admin/manager

```
<tomcat-users>
    <role rolename="manager"/>
    <role rolename="admin"/>
    <user username="admin" password="qwerty" roles="admin,manager"/>
</tomcat-users>
```

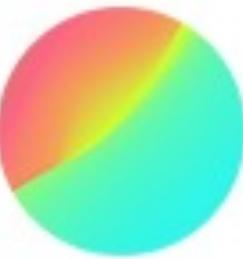


Despliegue en Tomcat

- En **settings.xml** dentro de .m2 (si no esta lo creamos) damos de alta el servidor.

```
<server>
    <id>TomcatServer</id>
    <username>admin</username>
    <password>qwerty</password>
</server>
```

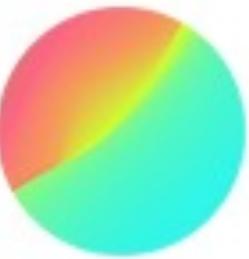
```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
  http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <localRepository/>
  <interactiveMode/>
  <usePluginRegistry/>
  <offline/>
  <pluginGroups/>
  <servers>
    <server>
      <id>TomcatServer</id>
      <username>admin</username>
      <password>qwerty</password>
    </server>
  </servers>
  <mirrors/>
  <proxies/>
  <profiles/>
  <activeProfiles/>
</settings>
```



Despliegue en Tomcat

3. En el POM declaramos el siguiente plugin:

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>tomcat-maven-plugin</artifactId>
    <configuration>
        <url>http://127.0.0.1:8080/manager</url>
        <server>TomcatServer</server>
        <path>/Pizza</path>
    </configuration>
</plugin>
```



Despliegue en Tomcat

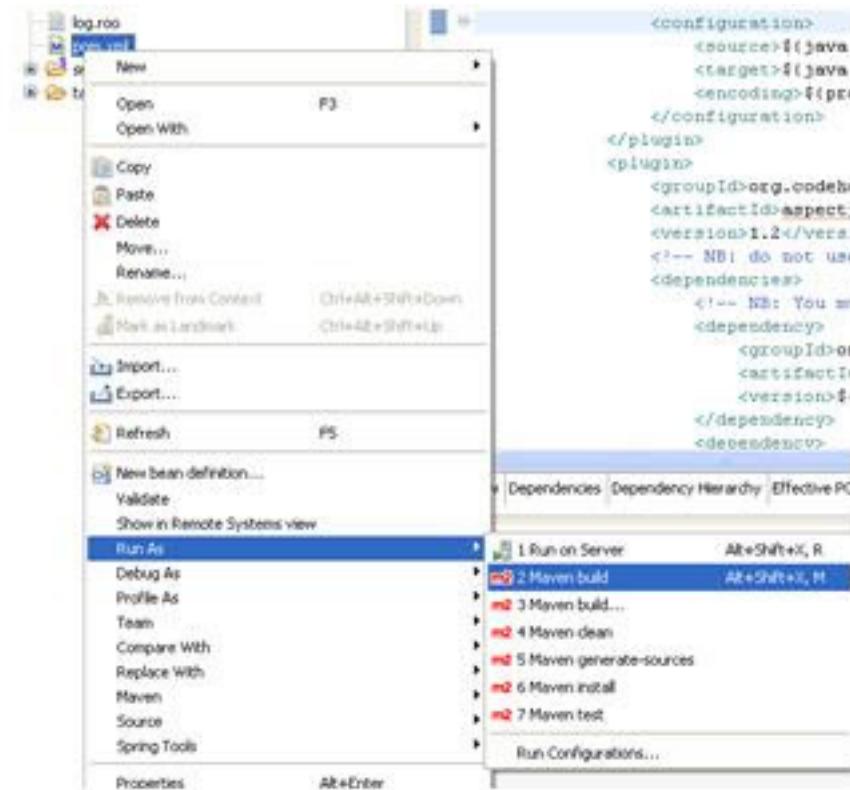
4. Verificar que el plugin de compilación de Maven este incluido:

```
<!-- Maven compiler plugin -->
<plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
        <source>2.5.1</source>
        <target>2.5.1</target>
    </configuration>
</plugin>
```

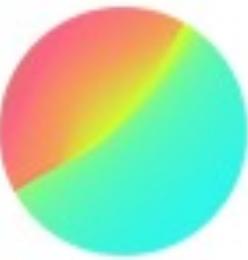
Despliegue en Tomcat



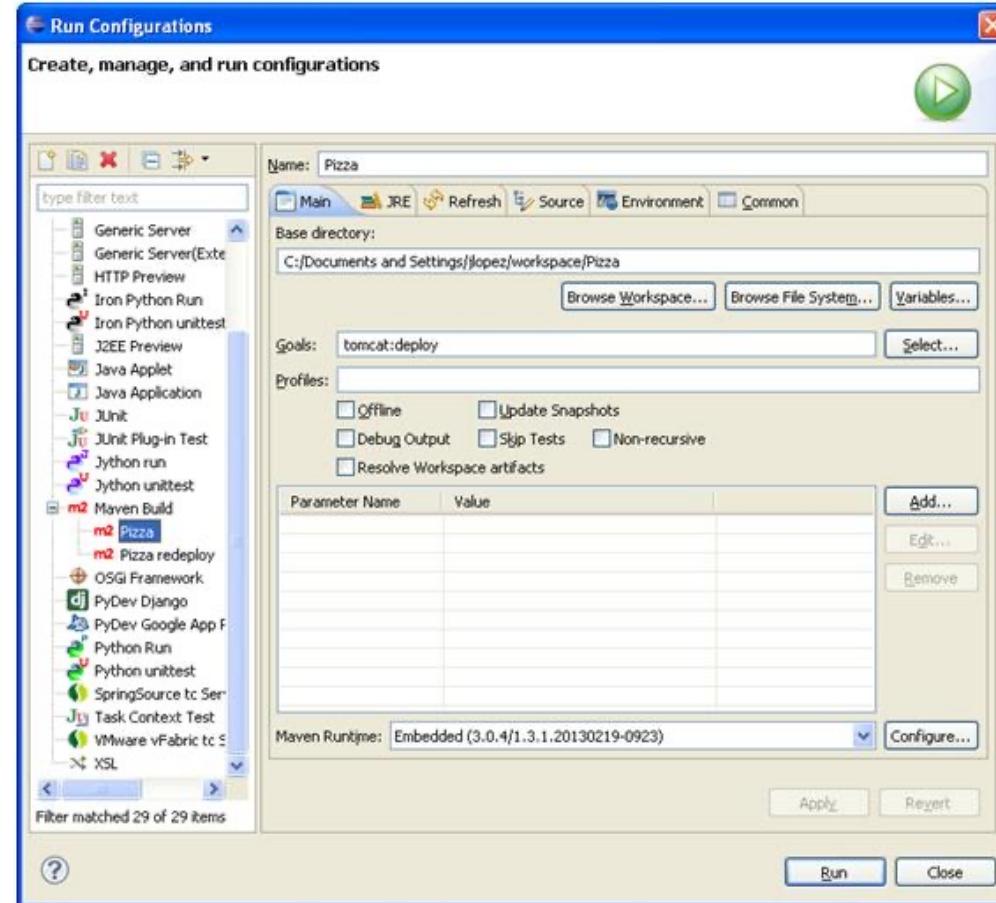
5. Crear tarea Maven:

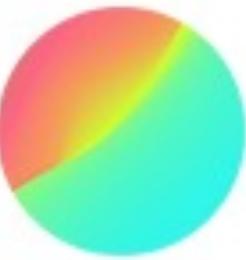


Despliegue en Tomcat



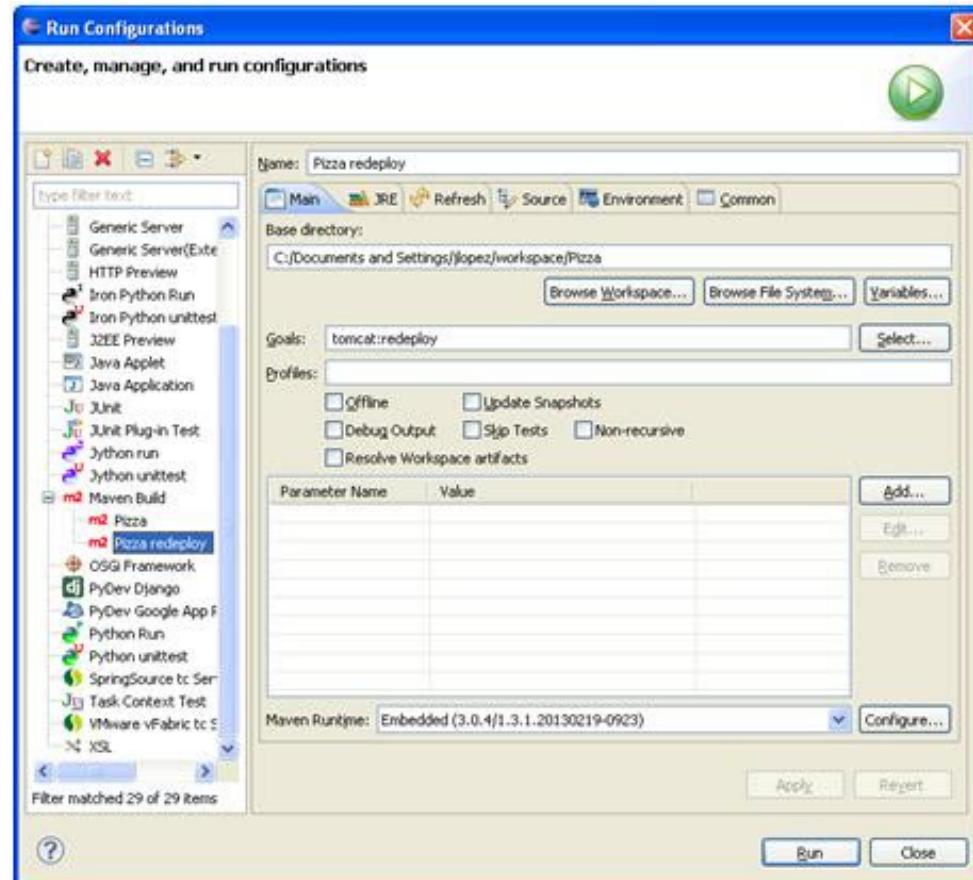
6. Definimos el goal:



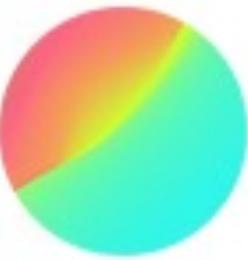


Despliegue en Tomcat

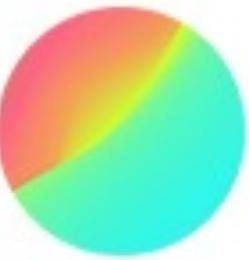
7. Definimos el goal como redeploy:



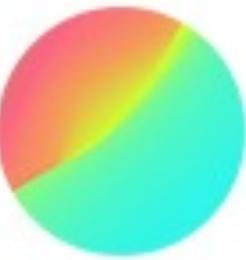
Fin deploy



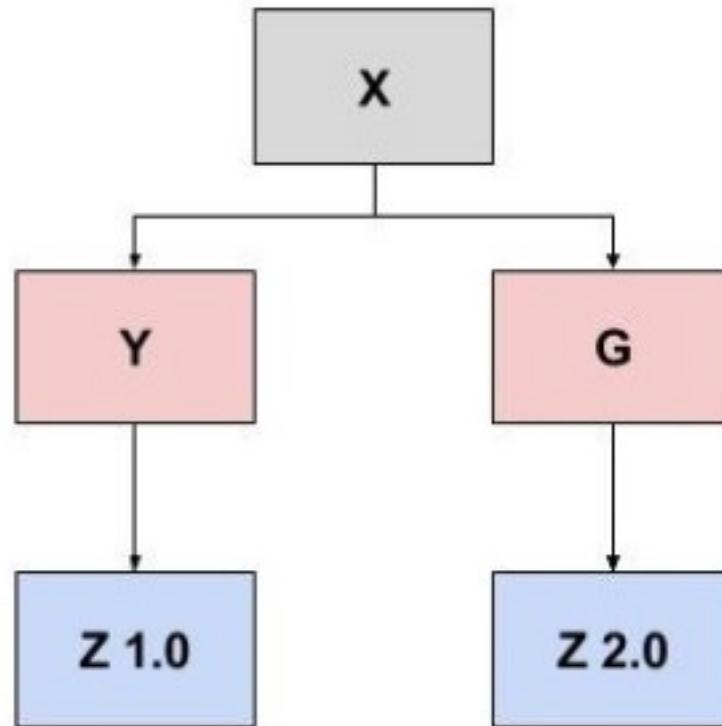
La Menza Perelló®. Todos los derechos reservados.

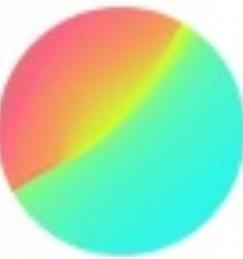


Problemas de dependencias



El proyecto X tiene dos dependencias iguales con distintas versiones



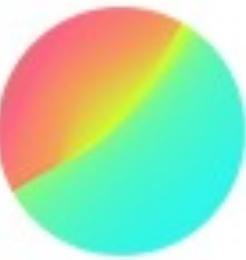


¿Qué versión Z estamos usando?

Nuestro proyecto X utiliza un mecanismo predeterminado de Maven llamado mecanismo de dependencia para resolver las dependencias y saber qué biblioteca usar.

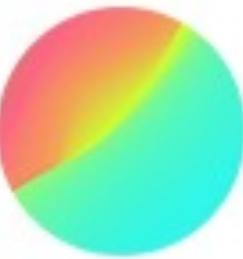
Veamos cómo funciona el mecanismo de dependencia:

En primer lugar, se utilizará la versión de la biblioteca cuyo nodo esté más cercano a la raíz (proyecto X) en el árbol de dependencia. Sin embargo, ¿qué sucede si hay varias versiones de la misma biblioteca cuyos nodos están en el mismo nivel en el árbol? En tal caso, se utiliza la primera versión de la biblioteca encontrada. Esto significa que la elección de las versiones de la biblioteca depende del orden de las dependencias dentro del archivo POM, donde se elegirán primero las dependencias declaradas primero.



Herramienta para detectar problemas

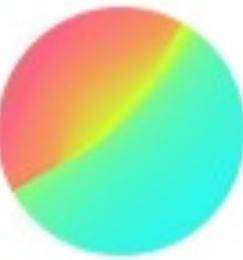
```
1 <plugins>
2   <plugin>
3     <groupId>org.apache.maven.plugins</groupId>
4     <artifactId>maven-enforcer-plugin</artifactId>
5     <version>1.4.1</version>
6     <configuration>
7       <rules><dependencyConvergence/></rules>
8     </configuration>
9   </plugin>
10 </plugins>
```



Resultado de usar la herramienta

mvn enforcer:enforce

```
1 Dependency convergence error for log4j:log4j:1.2.17 paths to dependency are:  
2  
3 +-com.ricston.conflict:conflict-info:2.1.3-SNAPSHOT  
4   +-org.slf4j:slf4j-log4j12:1.7.6  
5     +-log4j:log4j:1.2.17  
6  
7 and  
8  
9 +-com.ricston.conflict:conflict-info:2.1.3-SNAPSHOT  
10  +-log4j:log4j:1.2.16
```



Para evitar choques podemos usar un dependency manager

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>foo</groupId>
      <artifactId>bar</artifactId>
      <version>1.2.3</version>
    </dependency>
  </dependencies>
</dependencyManagement>
```