
M6.UF4.A6.P3

**CREACION E IMPORTACIÓN DE
PROYECTOS GRADLE Y MAVEN
EN ECLIPSE**

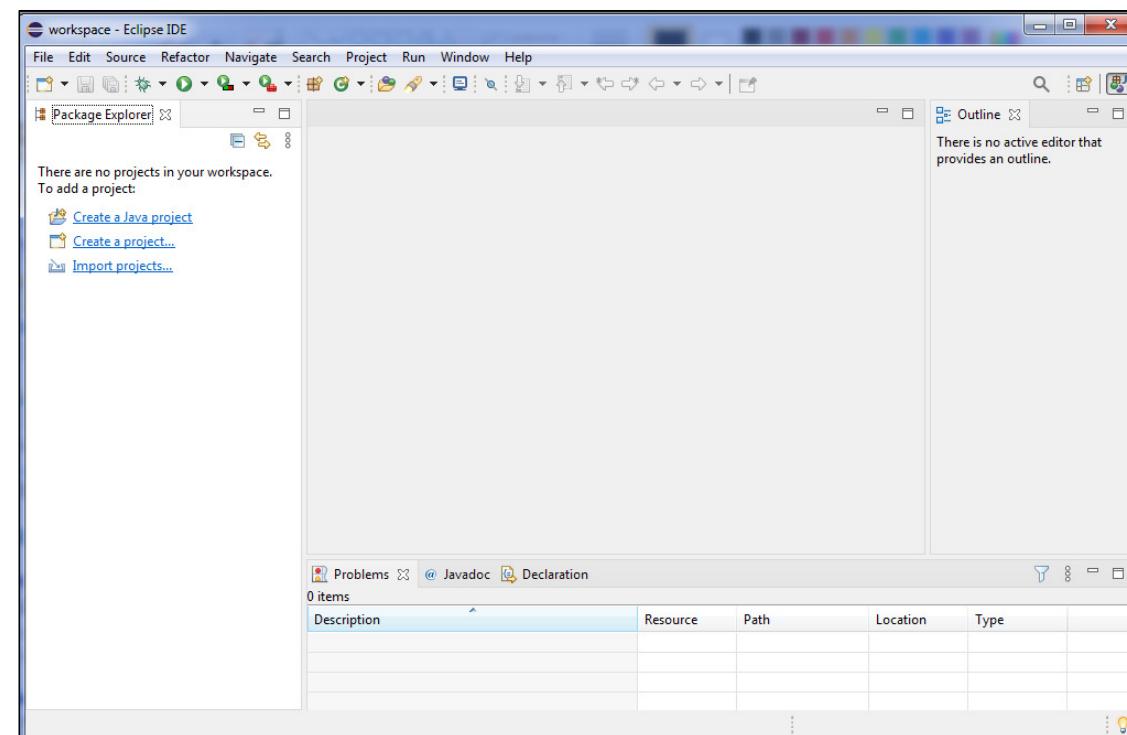
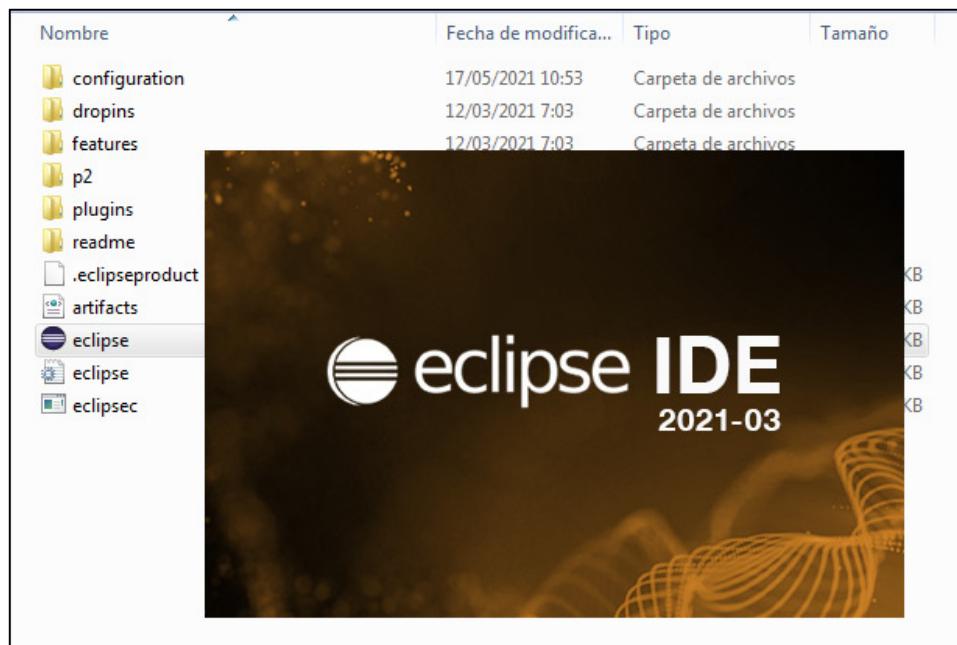
Eduard Lara

INDICE

1. Instalación Plugin Spring Eclipse
2. Eclipse SpringToolSuite
3. Creación proyecto Spring
4. Importar desde spring.io
5. Agregar dependencias una vez creado el proyecto

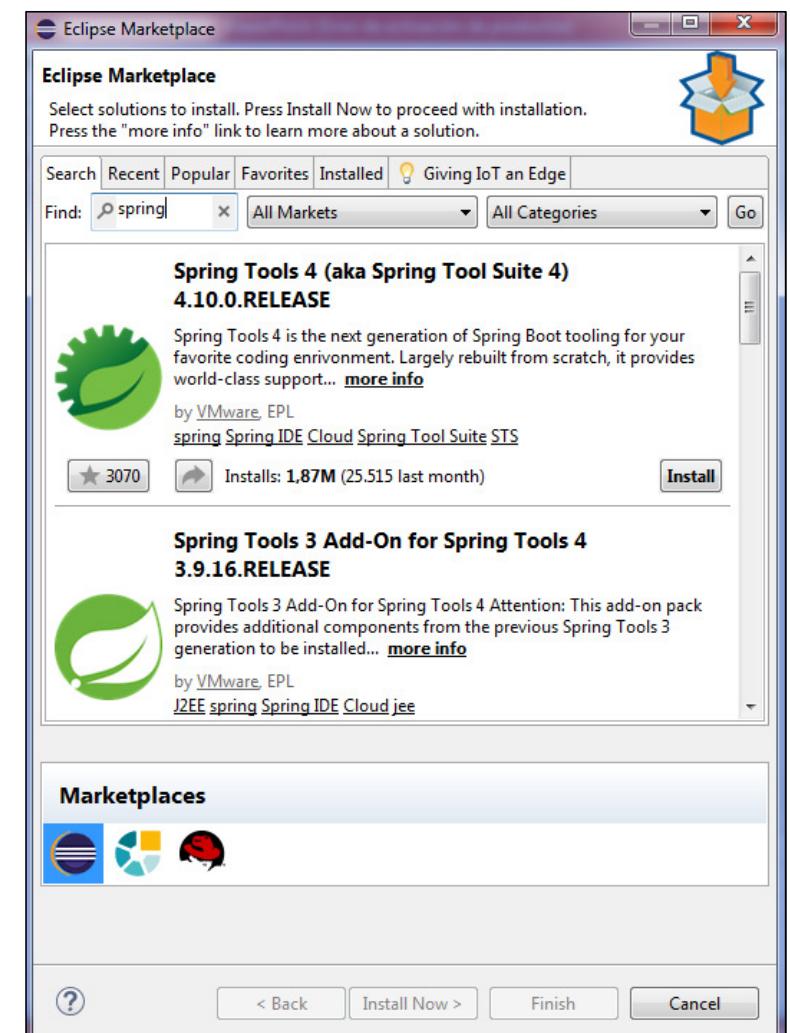
1. PLUGIN SPRING ECLIPSE

Paso 1) Para desarrollar con el Framework Spring, inicialmente en Eclipse debemos instalar el plugin de Spring, puesto que por defecto no viene instalado. Podemos arrancar nuestra versión de eclipse habitual, con la que hemos ido desarrollando, o bajarnos otra versión de <https://www.eclipse.org/downloads/packages/>:



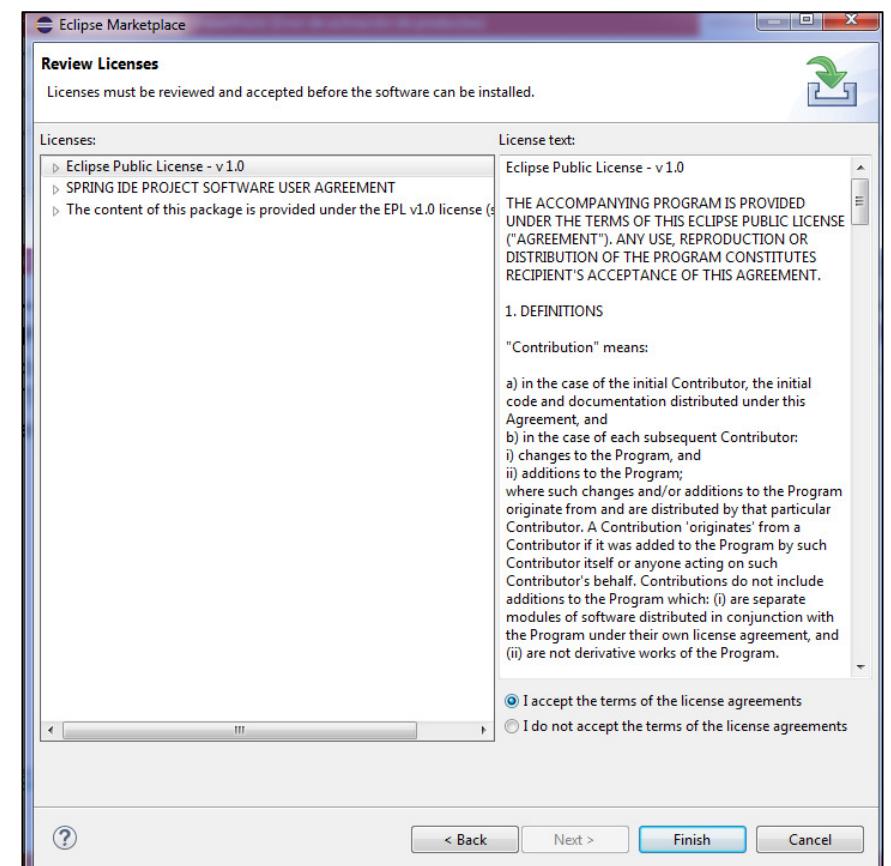
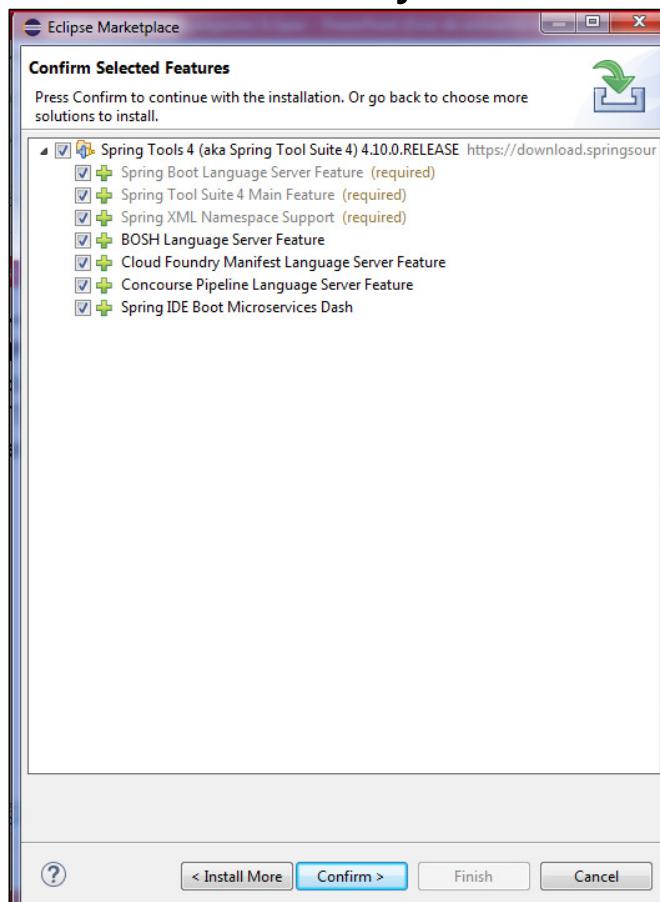
1. PLUGIN SPRING ECLIPSE

Paso 2) Vamos a Help/Eclipse MarketPlace y buscamos el término Spring. Vemos que el plugin Spring Tools 4 no se encuentra instalado. Hacemos click en Install:



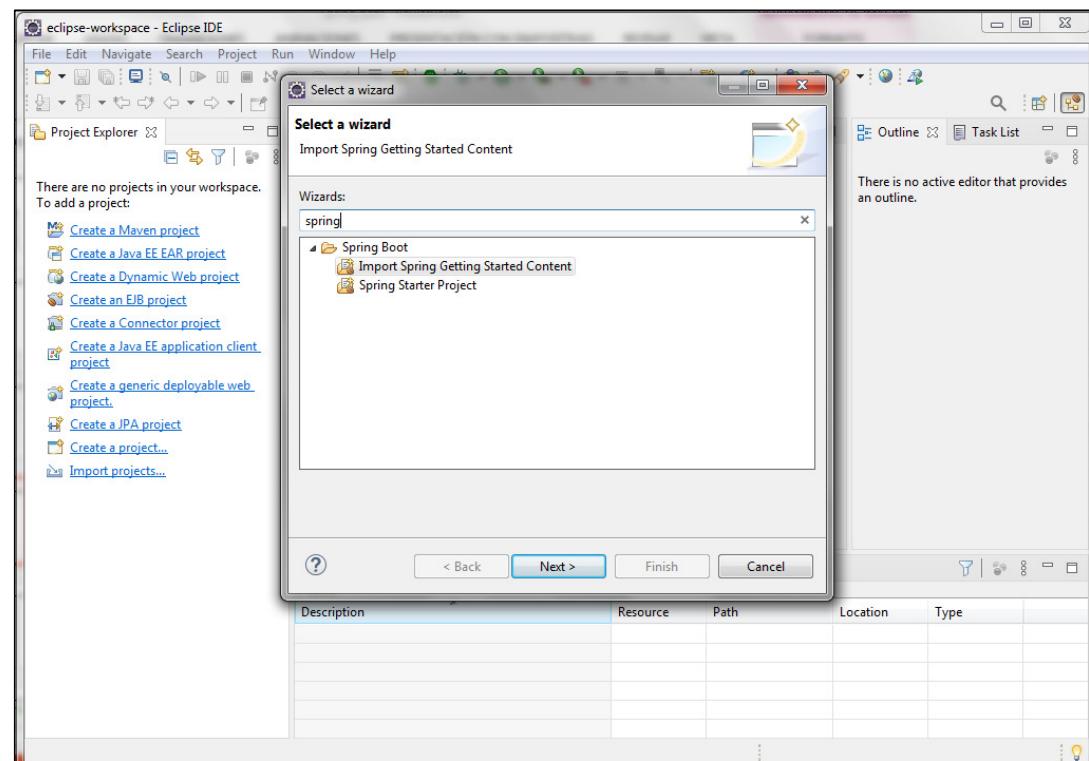
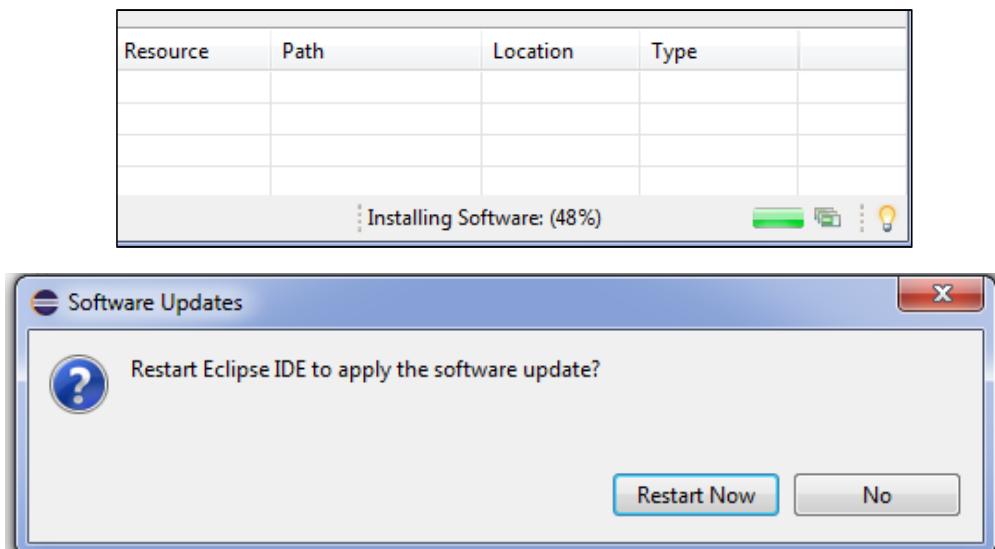
1. PLUGIN SPRING ECLIPSE

Paso 3) Confirmamos todo el paquete y en la siguiente pantalla aceptamos los términos de la licencia y hacemos click en Finish:



1. PLUGIN SPRING ECLIPSE

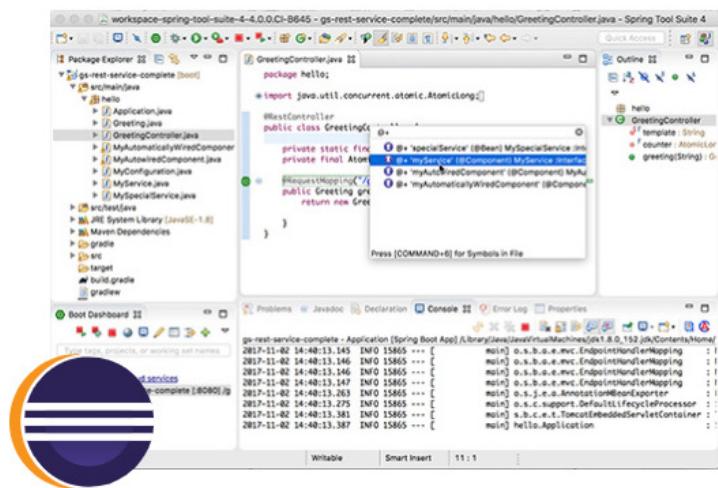
Paso 4) Empieza la instalación del plugin, que puede demorar un poco. Abajo a la derecha aparece la barra de proceso. Finalmente eclipse nos pide reiniciar y en el arranque vemos que ya podemos crear proyectos Spring Boot:



2. SPRINGTOOLSUIT

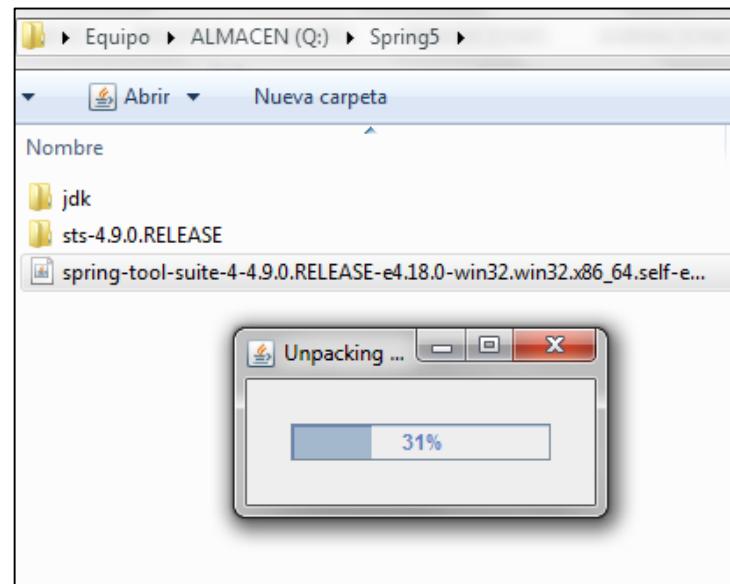
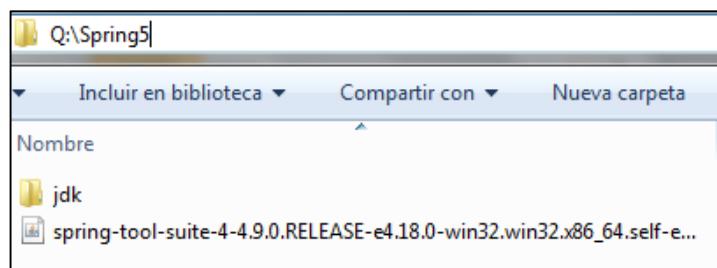
Paso 1) SpringToolSuite es una segunda opción para trabajar con Spring. Se trata de un eclipse preconfigurado con Spring, que se puede descargar desde la página spring.io/tools, previa elección de la versión idónea para nuestro sistema operativo

The screenshot shows the official website for Spring Tools 4. At the top, there's a navigation bar with links like 'Why Spring', 'Learn', 'Projects', 'Training', 'Support', and 'Community'. Below the navigation, the 'spring' logo is displayed. The main heading is 'Spring Tools 4 for Eclipse'. A subtext explains that it's the next generation of Spring tooling for your favorite environment. It highlights that it's largely rebuilt from scratch, providing world-class support for developing Spring-based enterprise applications across various IDEs. Three download links are provided at the bottom: '4.9.0 - LINUX 64-BIT', '4.9.0 - MACOS 64-BIT', and '4.9.0 - WINDOWS 64-BIT'.



2. SPRINGTOOLSUIT

Paso 2) Una vez descargado SpringToolSuite, lo descomprimimos en una carpeta:



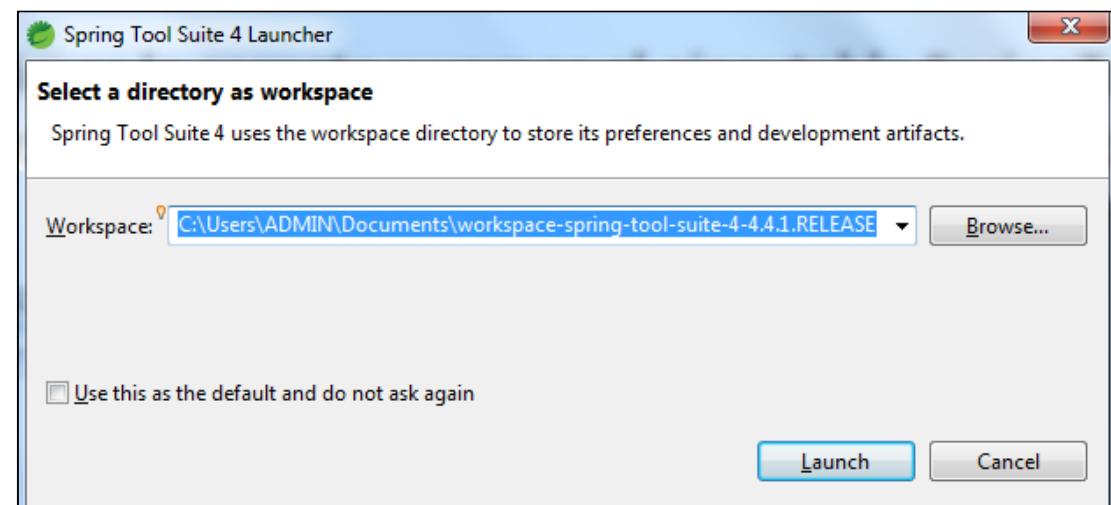
2. SPRINGTOOLSUIT

Paso 3) Abrimos la carpeta y vemos el ejecutable SpringToolSuite4. Hacemos dobleclick para iniciarla y lo primero que nos pregunta es por nuestro espacio de trabajo o workspace, donde vamos a guardar y a crear nuestros proyectos.

The screenshot shows a Windows File Explorer window with the following details:

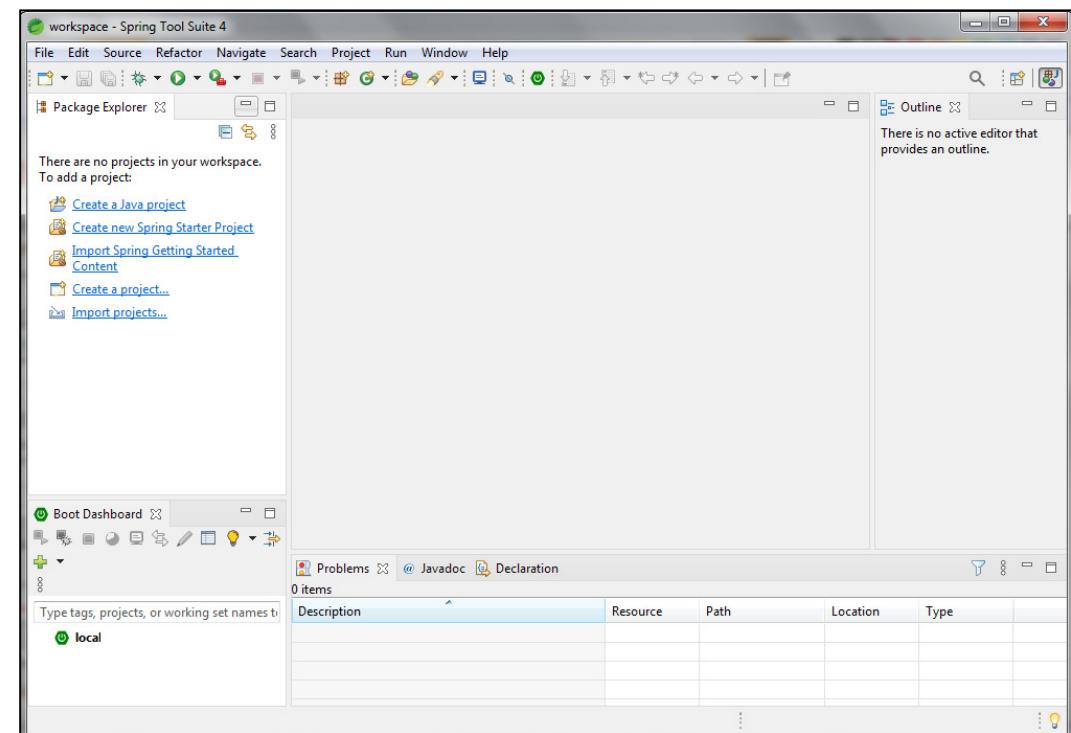
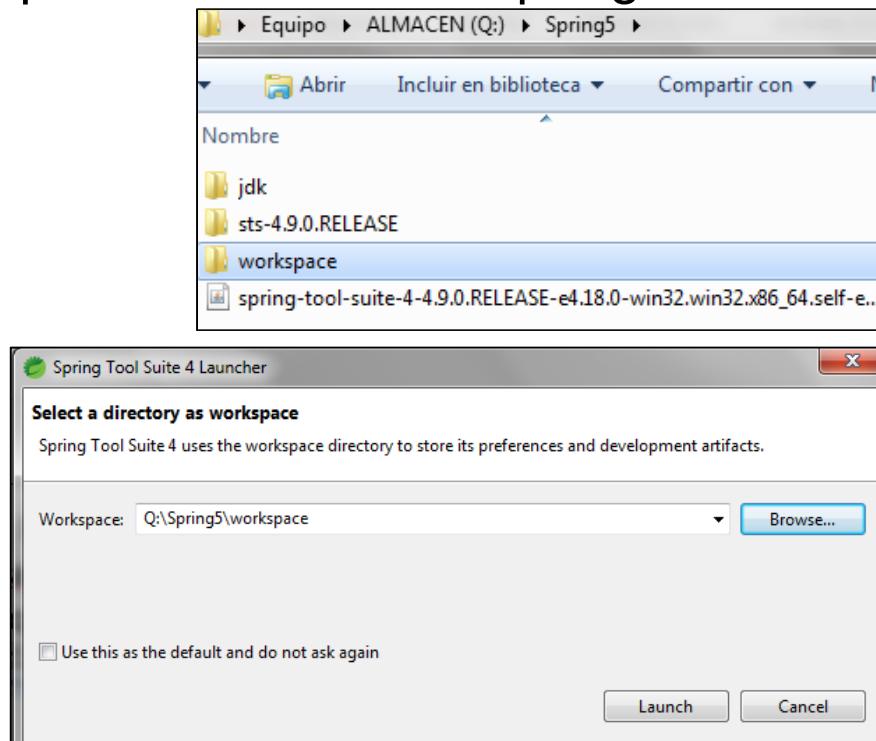
- Path: Equipo > ALMACEN (Q:) > Spring5 > sts-4.9.0.RELEASE
- Toolbar buttons: Incluir en biblioteca, Compartir con, Nueva carpeta.
- Table headers: Nombre, Fecha de modifica..., Tipo, Tamaño.
- File list:

Nombre	Fecha de modifica...	Tipo	Tamaño
configuration	27/12/2020 21:24	Carpeta de archivos	
dropins	27/12/2020 21:24	Carpeta de archivos	
features	27/12/2020 21:24	Carpeta de archivos	
META-INF	27/12/2020 21:24	Carpeta de archivos	
p2	27/12/2020 21:24	Carpeta de archivos	
plugins	27/12/2020 21:24	Carpeta de archivos	
readme	27/12/2020 21:24	Carpeta de archivos	
.eclipseproduct	27/12/2020 21:24	Archivo ECLIPSE...	1 KB
artifacts.xml	27/12/2020 21:24	Documento XML	163 KB
eclipsec.exe	27/12/2020 21:24	Aplicación	129 KB
license.txt	27/12/2020 21:24	Archivo TXT	12 KB
open-source-licenses.txt	27/12/2020 21:24	Archivo TXT	861 KB
SpringToolSuite4.exe	27/12/2020 21:24	Aplicación	417 KB
SpringToolSuite4.ini	27/12/2020 21:24	Opciones de confi...	1 KB



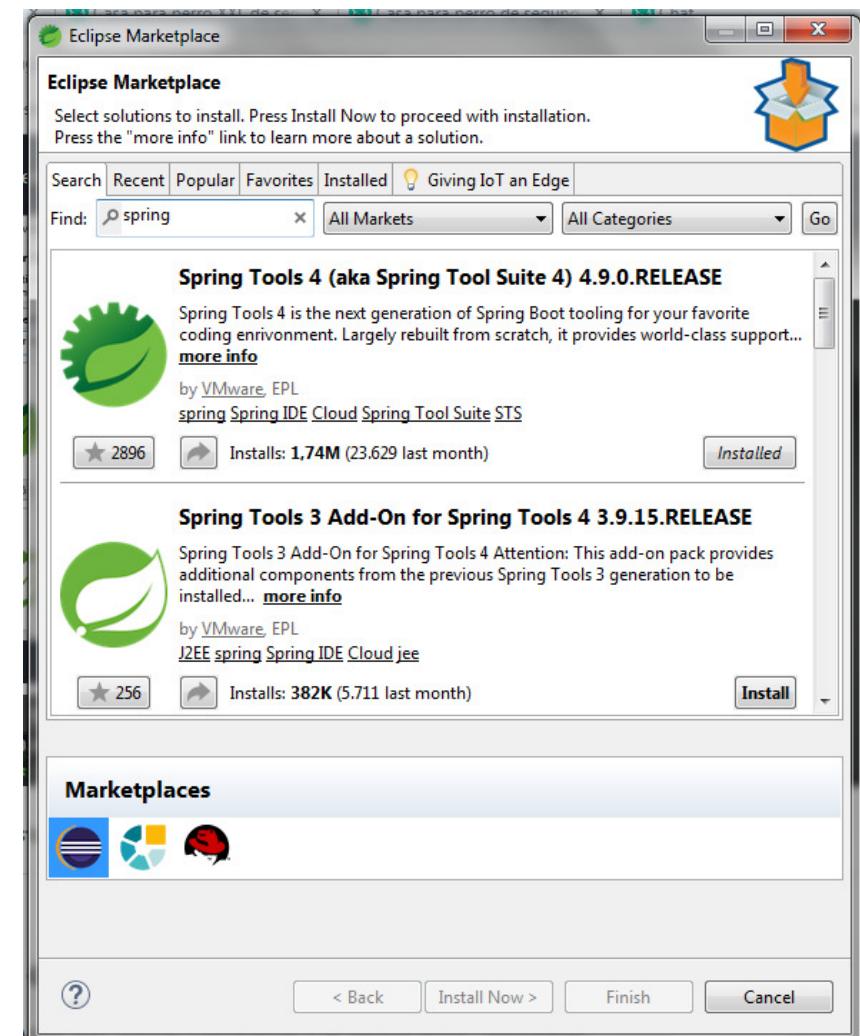
2. SPRINGTOOLSUITE

Paso 4) En el mismo directorio del SpringToolSuite, creamos la carpeta workspace, y así se la indicamos al SpringToolSuite. Cuando arranca vemos que se trata de un eclipse disfrazado de Spring.



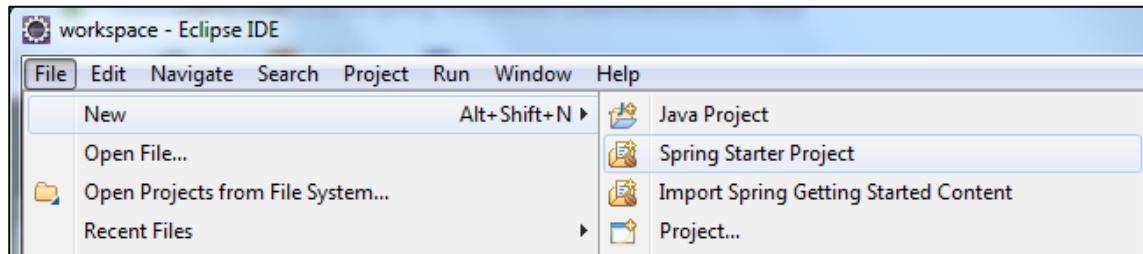
2. SPRINGTOOLSUITE

Paso 5) Vamos a Help/Eclipse MarketPlace y buscamos Spring. Vemos que el plugin Spring Tools 4 se encuentra instalado

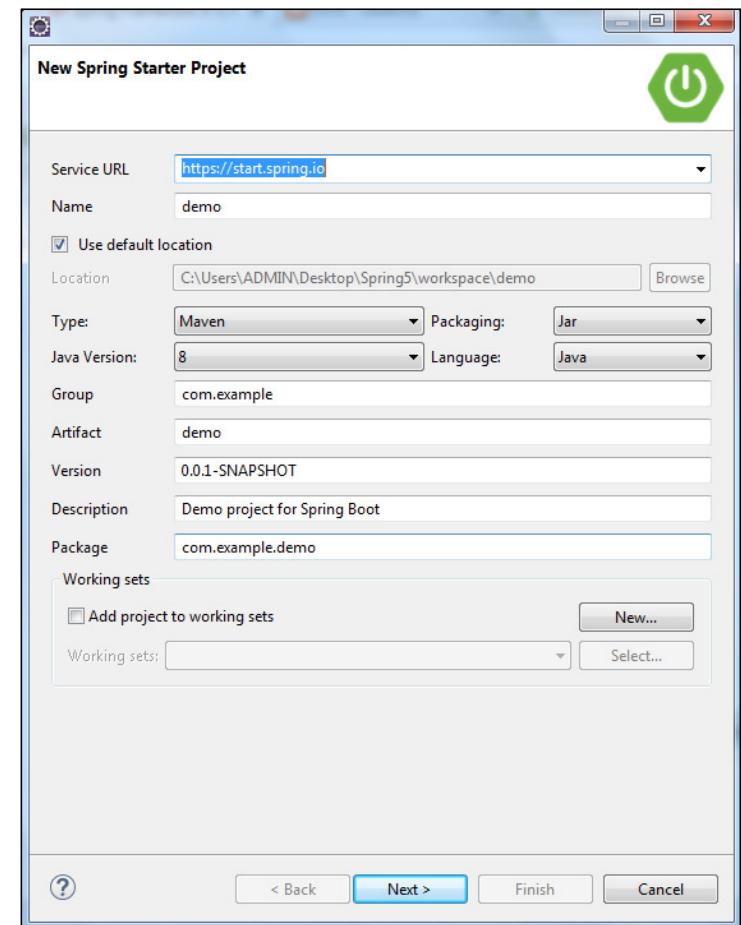
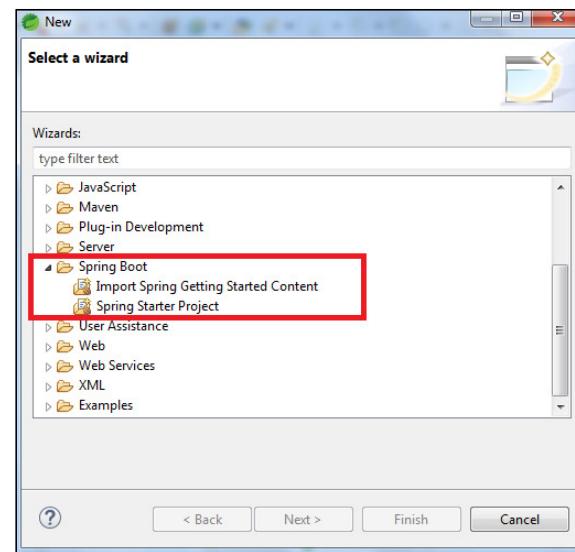


3. CREAR PROYECTO SPRING

Paso 1) Seleccionamos File/New/Spring Starter Project:



En caso de que no
aparezca debemos ir
por el camino largo.
File/New/Otros. Ir a
Spring boot



3. CREAR PROYECTO SPRING

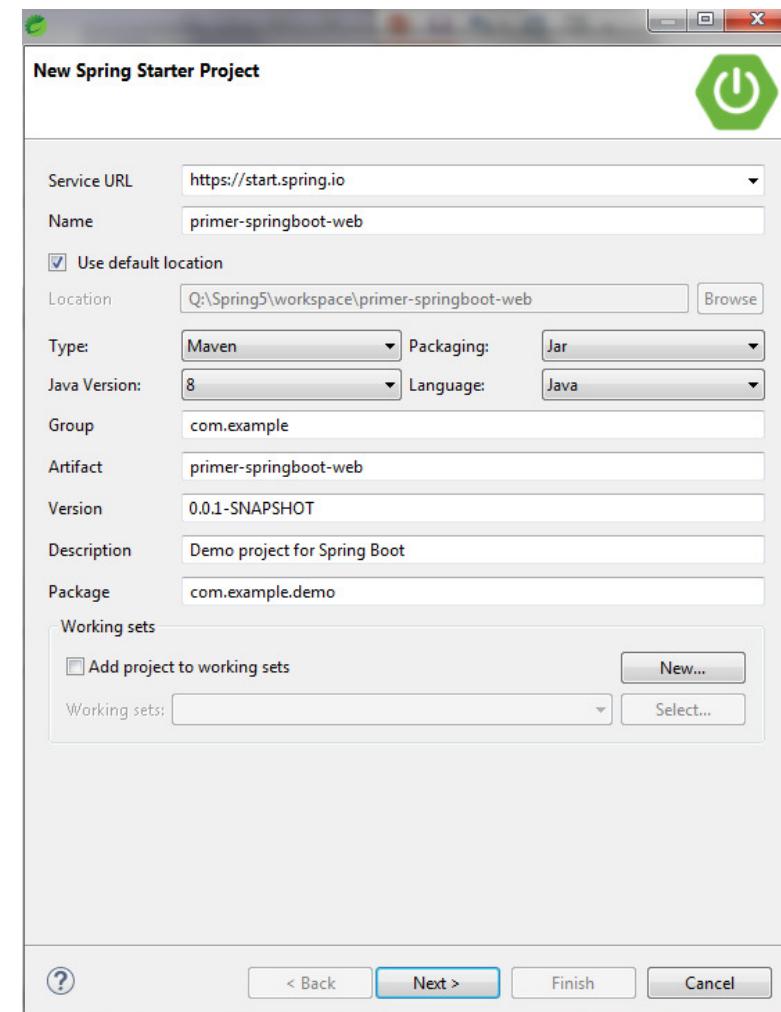
Paso 2) Parámetros de creación del proyecto:

Name: Nombre del proyecto, sin espacios en blanco
primer-springboot-web

Type: Maven/Gradle. Administrador de dependencias para generar el proyecto. Permiten crear nuestro proyecto, compilar o construir nuestro jar o war final que vamos a desplegar o publicar en producción.

Java Version: 16, 11, 8. Usaremos la 8

Group: Muy similar a los packages de java. Estructura de package y ahí podemos agrupar varias aplicaciones de Maven dentro del mismo Group Id



3. CREAR PROYECTO SPRING

Paso 2) Parámetros de creación del proyecto:

Packaging: Tipo de empaquetado del proyecto: jar/war. Jar se recomienda cuando creamos aplicaciones con SpringBoot. Inicialmente uno podría pensar que war es para proyectos web y jar para aplicaciones estándar, pero con jar también podemos crear proyectos web.

Spring boot incluye un servidor embebido Tomcat que permite desplegar aplicaciones en jar. Es mucho mas portable y mucho mas fácil desplegar aplicaciones con jar. Sólo se necesita el JDK, y el comando java -jar nom_proyecto. No requiere instalar ningún servidor como Tomcat externo, JBOSS O Glassfish.

Utilizamos War cuando queremos publicar en un servidor externo que no sea el que viene dentro de SpringBoot: JBOSS, Glasfish, Tomcat externo, o trabajamos con jsp
Cuando tenemos vistas por ejemplo con Thymeleaf o aplicaciones con API Rest Backend utilizamos jar.

3. CREAR PROYECTO SPRING

Paso 2) Parámetros de creación del proyecto:

Artifact: Nombre del proyecto

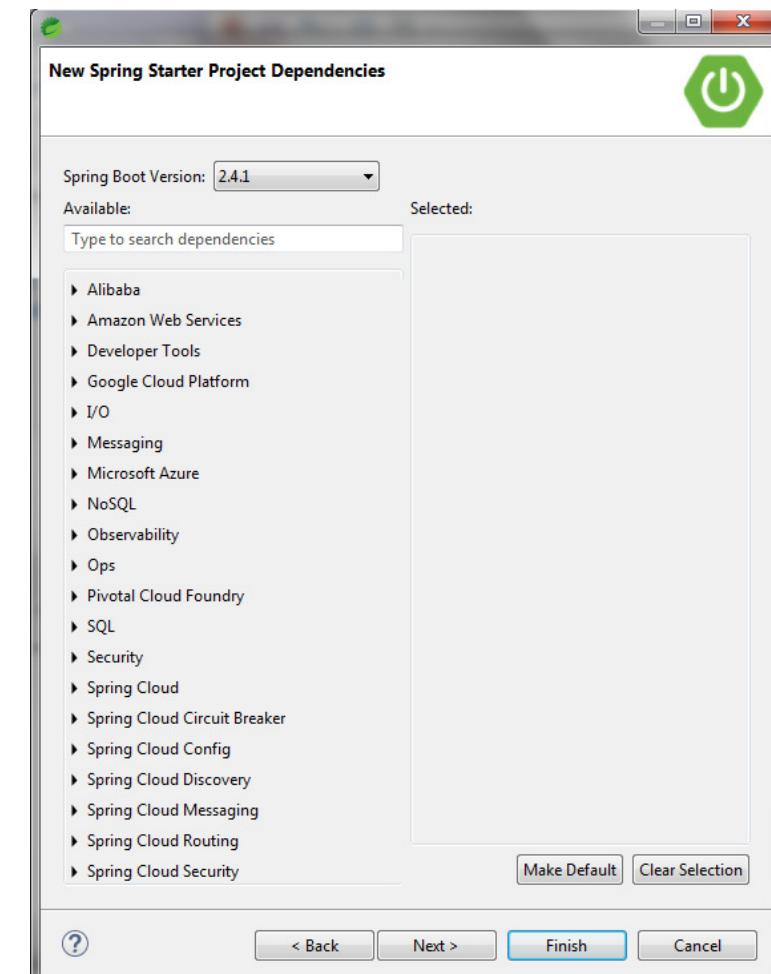
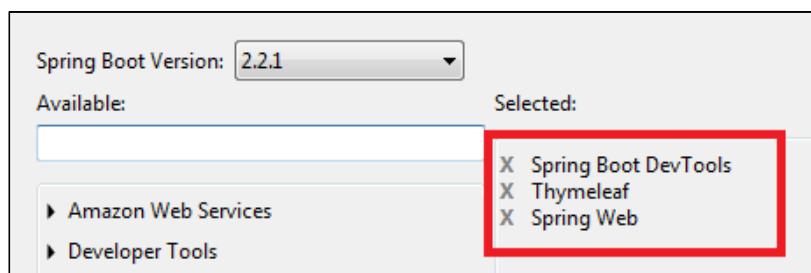
Version: La versión que le vamos a dar

Package: Package de java. Package base principal o raíz de nuestra aplicación springboot. Un package se compone del nombre_del_dominio_de_la_empresa + nombre_área_trabajo o nombre_del_proyecto que estamos desarrollando. No hay una regla tan estricta para esto. Podemos poner el mismo package en Group ID.

3. CREAR PROYECTO SPRING

Paso 3) Spring Boot Version: Poner la última versión estable, que no sea SNAPSHOT. Estas versiones no se recomiendan porque son como versiones beta en desarrollo, solo se debe marcar la última final estable

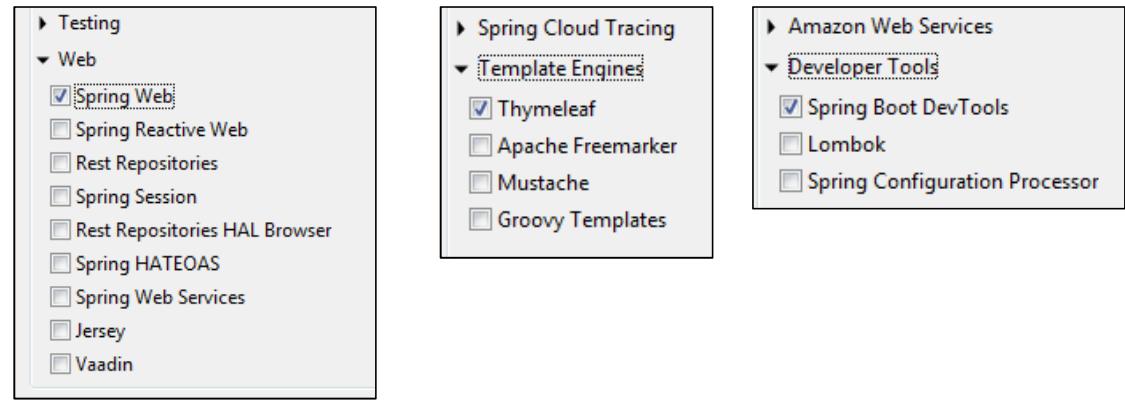
Dependencias: Aquí configuramos las dependencias del proyecto SpringBoot. Para trabajar con una aplicación web, hemos de seleccionar los siguientes componentes: Spring web, SpringBoot Devtools y Thymeleaf.



3. CREAR PROYECTO SPRING

Paso 3) Componente **Spring web**, dentro de Web. Es la base que incluye otras dependencias como:

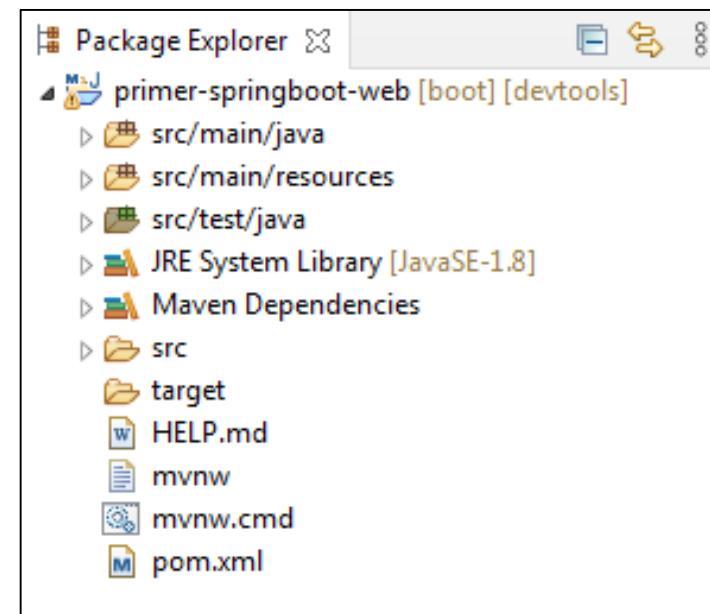
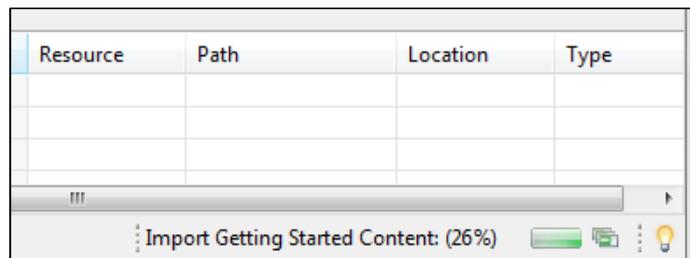
- Spring-core, spring-web
- Spring webmvc, Servlet-api
- Spring-boot-starter-tomcat
- Spring-boot-autoconfigure



- Componente **Thymeleaf** dentro de Templates Engines. Es el motor de plantillas que mas se usa y recomienda. Integracion total con Spring boot
- Componente **Spring Boot DevTools** dentro de Developer Tools. Es importante ya que cualquier cambio que hagamos en nuestro código java, se reinicia el servidor automáticamente, actualizando el despliegue de forma optimizada y rápida, sin tener que hacerlo de forma manual.

3. CREAR PROYECTO SPRING

Paso 4) Hacemos click en finish y puede tardar un rato la creación del proyecto por la descarga de las dependencias:



No ha marcado error pero lo podría haber hecho por la descarga mal de un fichero que marcará el fichero como corrupto por una marca en rojo

3. CREAR PROYECTO SPRING

Paso 5) En el fichero pom.xml podemos ver la versión de java, que se puede cambiar, las dependencias anteriormente agregadas (thymeleaf, web, devtools), todo configurado automáticamente con SpringBoot.



The screenshot shows the pom.xml file for a Spring Boot project named 'primer-springboot-web'. The file is structured as follows:

```
<groupId>com.example</groupId>
<artifactId>primer-springboot-web</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>primer-springboot-web</name>
<description>Demo project for Spring Boot</description>

<properties>
    <java.version>1.8</java.version>
</properties>

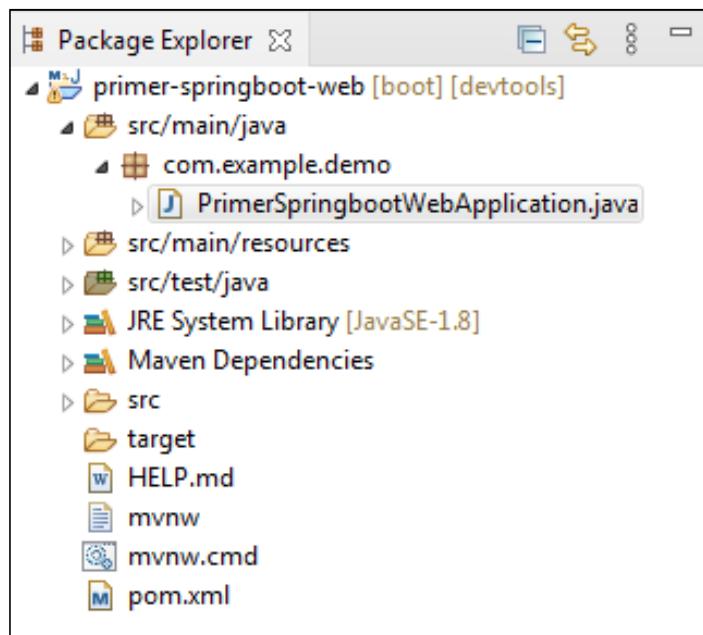
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

3. CREAR PROYECTO SPRING

Paso 6) Debajo de nuestro package com.example.demo están todas las clases del proyecto: controlador, models, service, etc.

Las clases con la notación component o service, permite registrar estos componentes Spring, pero para eso deben de estar dentro de este contexto.



```
PrimerSpringbootWebApplication.java
1 package com.example.demo;
2
3 import org.springframework.boot.SpringApplication;...
5
6 @SpringBootApplication
7 public class PrimerSpringbootWebApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(PrimerSpringbootWebApplication.class, args);
11     }
12
13 }
14
```

Es el inicio de la aplicación PrimerSpringbootWebApplication. En el método estatico run de SpringApplication se le pasa como argumentos el class de nuestro fichero de inicio, junto con los argumentos del main

3. CREAR PROYECTO SPRING

Paso 7) En PrimerSpringbootWebApplication encontramos la anotación @SpringBootApplication. Haciendo Ctrl click vemos la configuración de spring

Podemos resaltar los componentes:

@SpringBootConfiguration

@EnableAutoConfiguration habilita la configuración automática

@ComponentScan que escanea y busca los componentes que están dentro de este package con anotaciones controller, service, repository. Busca todos los beans, y los registra.

```
SpringBootApplication.class X
39 /**
40 * Indicates a {@link Configuration configuration} class that declares one or more
41 * {@link Bean @Bean} methods and also triggers {@link EnableAutoConfiguration
42 * auto-configuration} and {@link ComponentScan component scanning}. This is
43 * annotation that is equivalent to declaring {@code @Configuration},
44 * {@code @EnableAutoConfiguration} and {@code @ComponentScan}.
45 *
46 * @author Phillip Webb
47 * @author Stephane Nicoll
48 * @author Andy Wilkinson
49 * @since 1.2.0
50 */
51 @Target(ElementType.TYPE)
52 @Retention(RetentionPolicy.RUNTIME)
53 @Documented
54 @Inherited
55 @SpringBootConfiguration
56 @EnableAutoConfiguration
57 @ComponentScan(excludeFilters = { @Filter(type = FilterType.CUSTOM, classes =
58                                     @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExclude
59 public @interface SpringBootApplication {
60
61 /**
62 * Exclude specific auto-configuration classes such that they will never
63 * @return the classes to exclude
64 */
65 @AliasFor(annotation = EnableAutoConfiguration.class)
66 Class<?>[] exclude() default {};
67
```

3. CREAR PROYECTO SPRING

Paso 8) Dependencias de Maven. Se descargan de forma automática en el proyecto indicadas por el fichero pom.xml.

Spring boot starter thymeleaf

Spring web

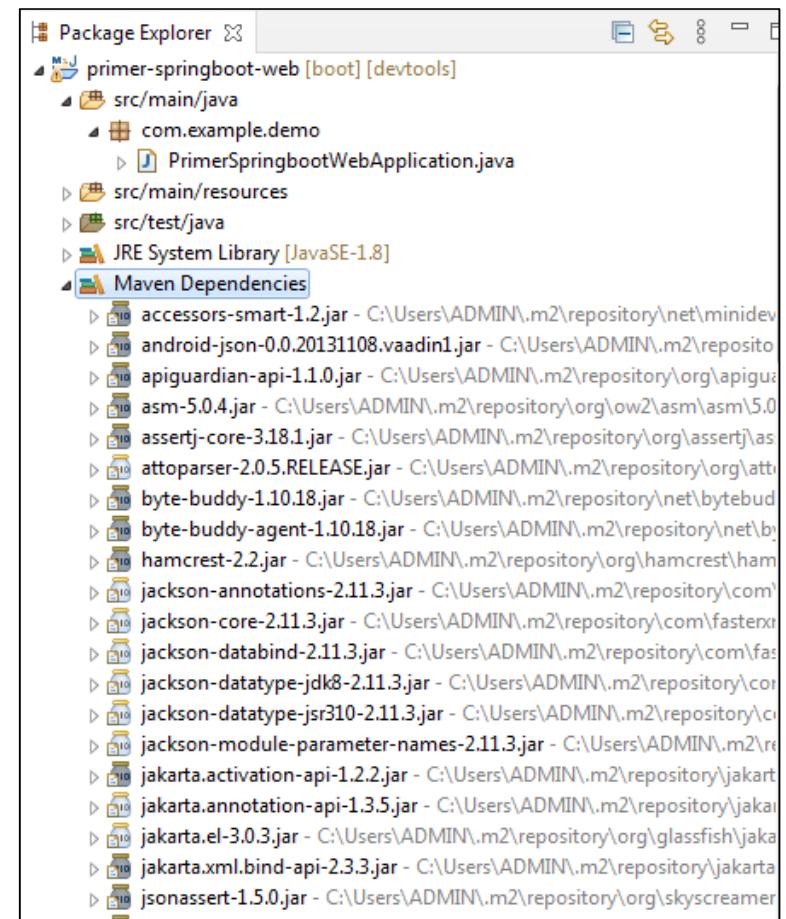
Spring boot devtools

Spring core

Tomcat embeded core

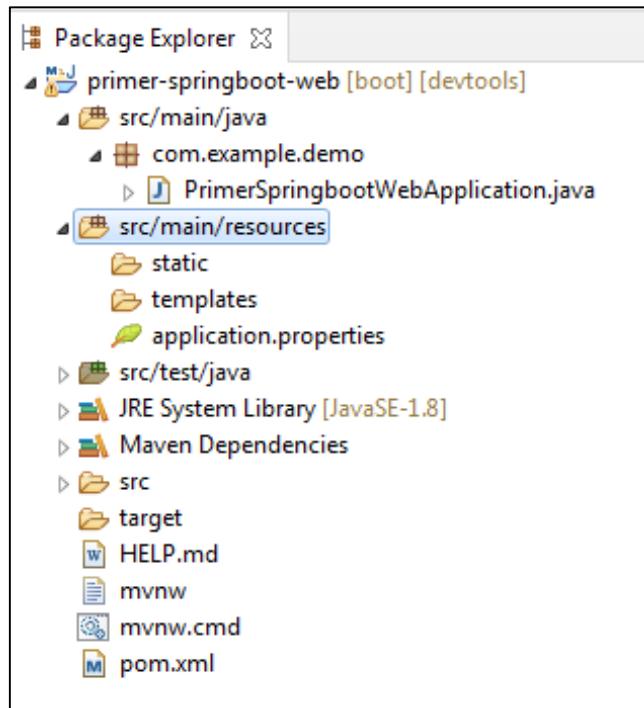
Spring web mvc

Spring aop



3. CREAR PROYECTO SPRING

Paso 9) Directorio `src/main/resources`. Tenemos el fichero **`application.properties`** con la configuración principal de springboot. Esta vacío porque toda la configuración viene por debajo.



Se puede utilizar para cambiar el puerto del servidor de 8080 a 8090, por ejemplo.

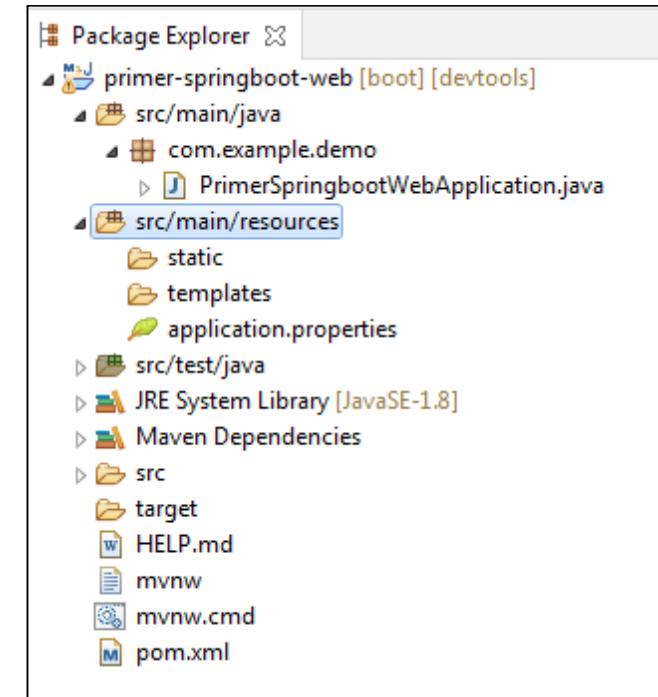
Importante: No dejar después un espacio en blanco. Aquí se pueden configurar parámetros de acceso a base de datos Mysql, el connectString, usuario, password, la clase jdbcDriver, JPA o hibernate, el dialecto del motor de base de datos, etc

```
server.port=8090
```

3. CREAR PROYECTO SPRING

Paso 10)

Carpeta templates: Para guardar las plantillas de las vistas de los controladores. Se recomienda utilizar plantillas thymeleaf como motor de plantillas por encima de JSP ya que es mucho mas robusto y esta mas optimizado para HTML5. Se trabaja a través de atributos html que son propios de thymeleaf. No se compila a un servlet ni tiene dependencias con api servlet. Tampoco utiliza etiquetas ni taglib como jsp. El código es mucho mas limpio y puro.

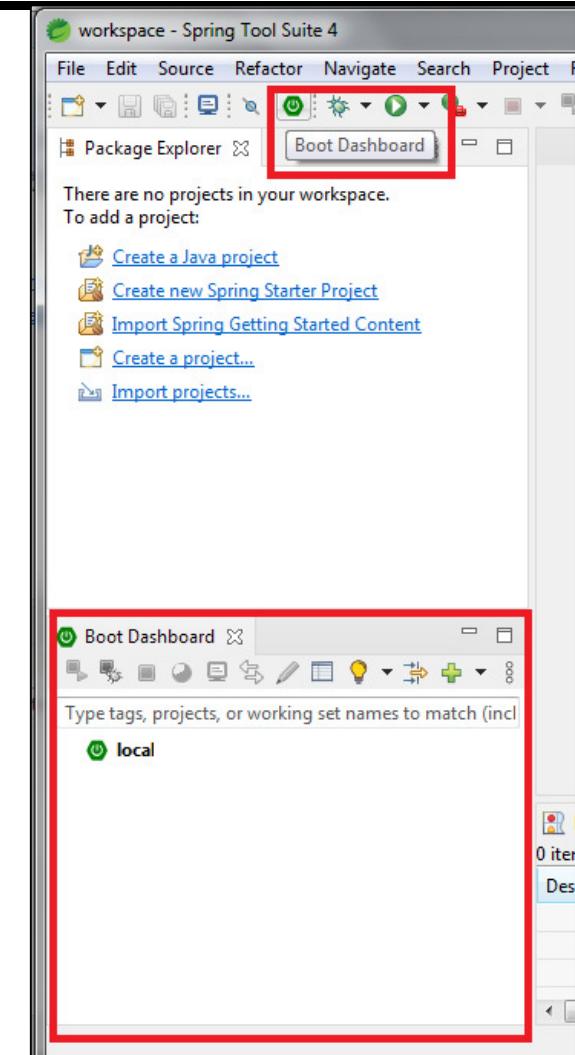
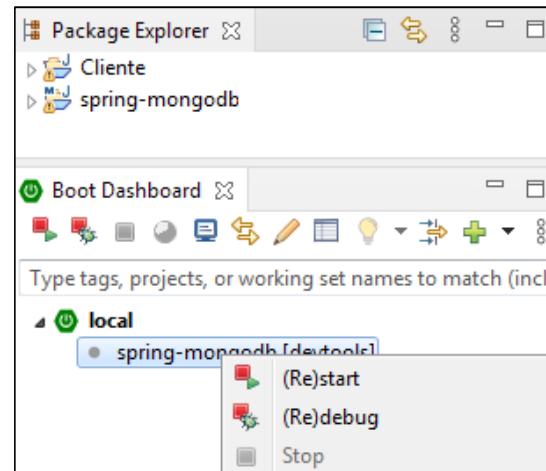


Carpeta static: Para guardar los recursos estáticos: css, javascript, imágenes.

Carpeta target: Donde aparecerá el jar, el archivo ejecutable que podemos copiar y ejecutar con java -jar.

3. CREAR PROYECTO SPRING

Paso 11) Pestaña Boot Dashboard. Cada vez que creamos o abrimos un proyecto SpringBoot, éste aparece desplegado en la pestaña Dashboard bajo la etiqueta local. Es el lugar ideal para detenerlo, iniciar o reiniciarlo. La primera vez que se levanta un proyecto se puede hacer mediante **Run as/Spring boot**. Si se tiene que volver a reiniciar la aplicación, no se debe volver a hacer de esta forma porque da error al estar tomado el puerto. Se tiene que hacer desde la pestaña Boot Dashboard.



4. IMPORTACIÓN DESDE SPRING.IO

Paso 1. Vamos a la pagina <https://start.spring.io> y creamos un proyecto gradle, que llamaremos prueba_gradle. No agregamos ninguna dependencia:

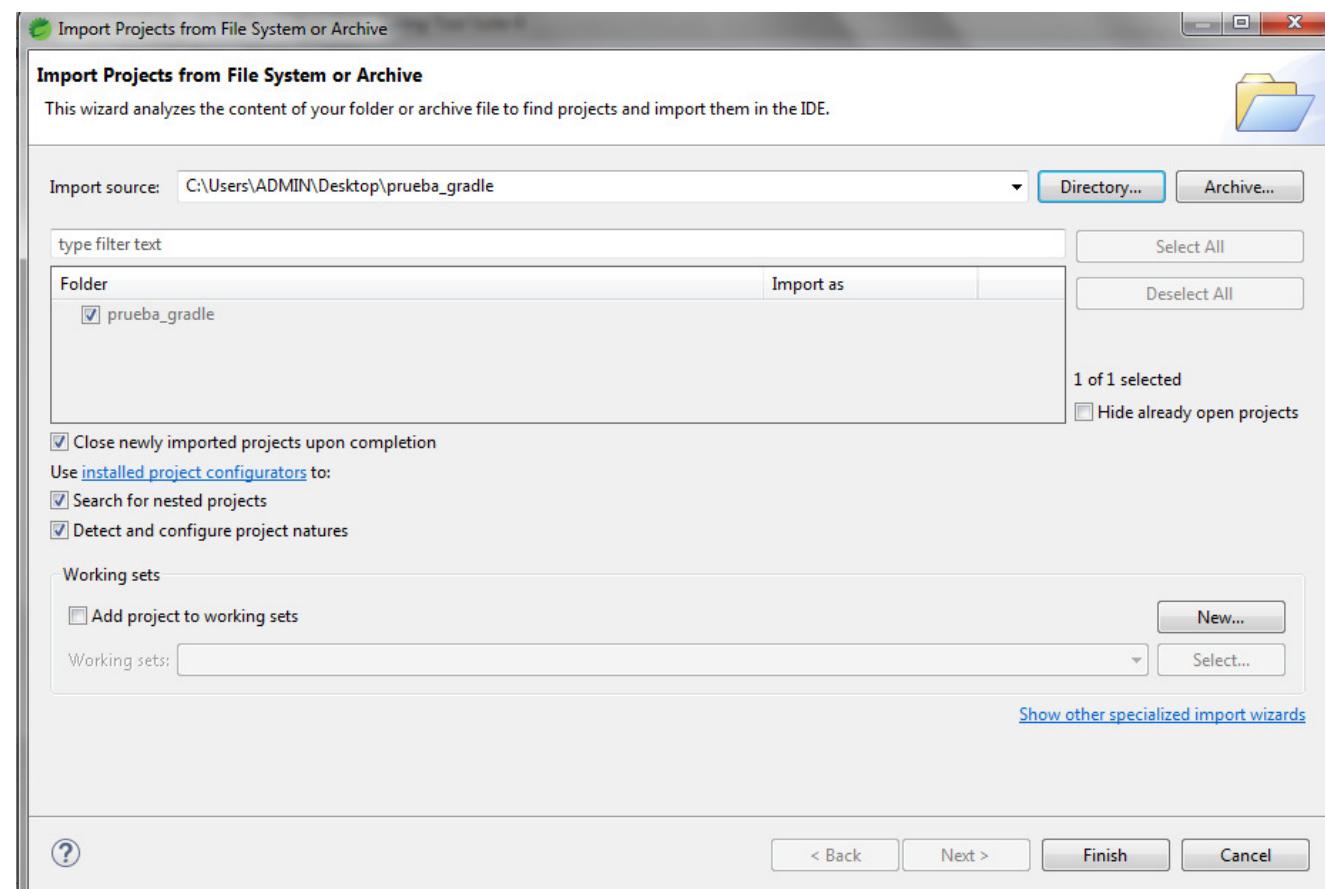
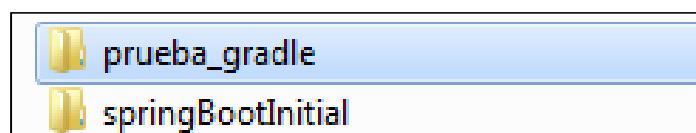
The screenshot shows the Spring Initializr web interface at <https://start.spring.io>. The configuration is as follows:

- Project:** Gradle Project (selected)
- Language:** Java (selected)
- Spring Boot:** 2.4.5 (selected)
- Dependencies:** No dependency selected
- Project Metadata:**
 - Group: com.example
 - Artifact: prueba_gradle
 - Name: prueba_gradle

At the bottom, there are three buttons: GENERATE (CTRL + D), EXPLORE (CTRL + SPACE), and SHARE... .

4. IMPORTACIÓN DESDE SPRING.IO

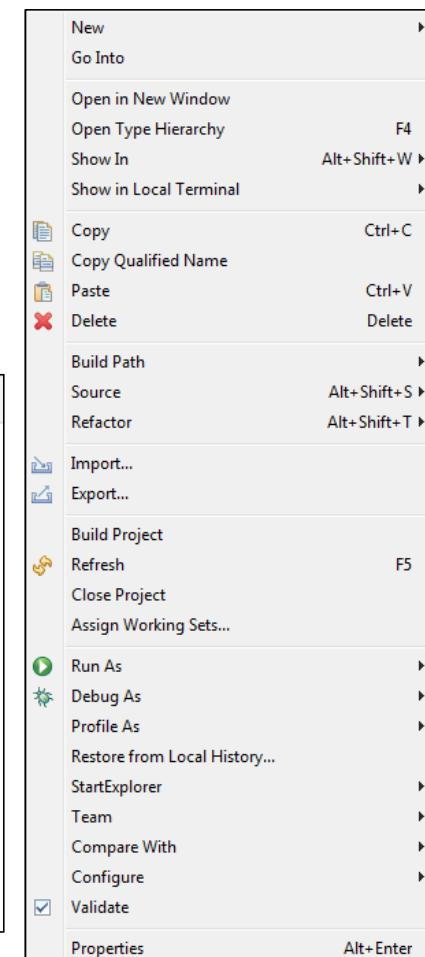
Paso 2. Descomprimimos el zip y lo importamos a Eclipse, en File/Open Projects from File System...



4. IMPORTACIÓN DESDE SPRING.IO

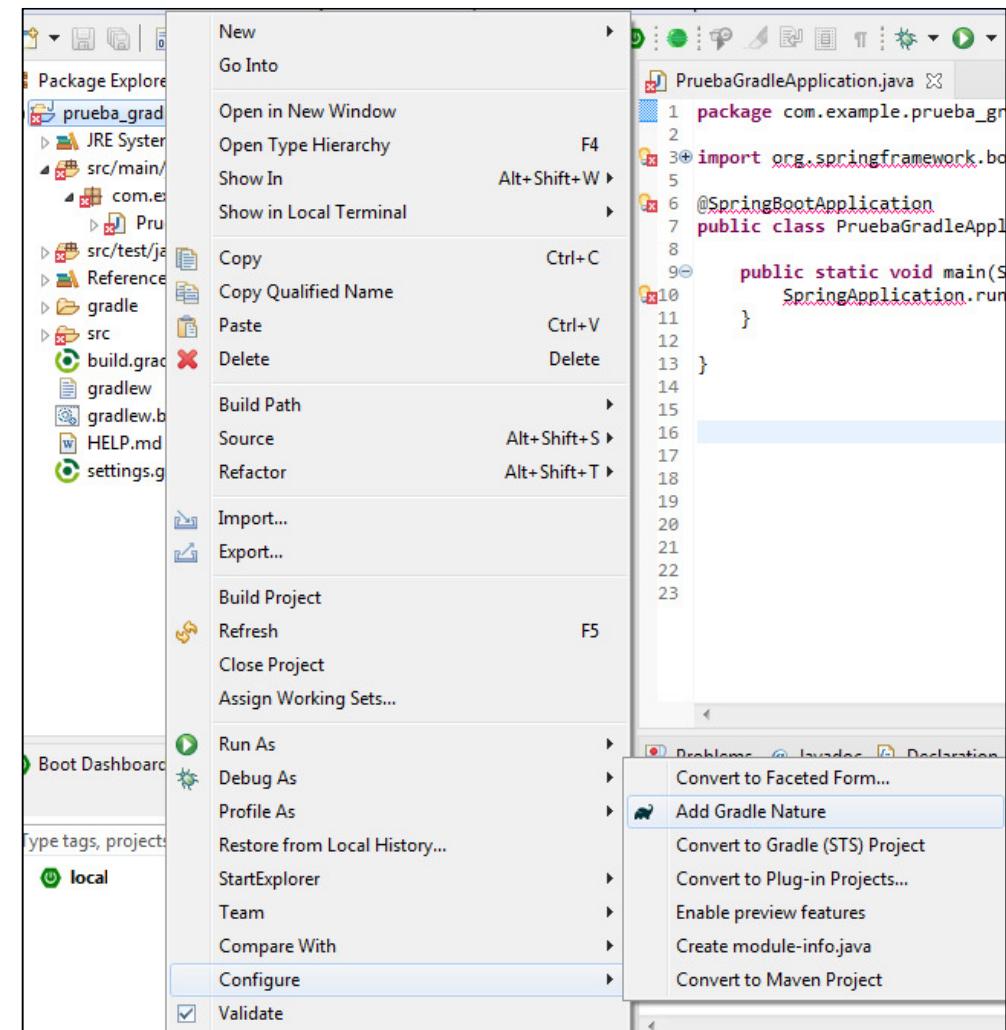
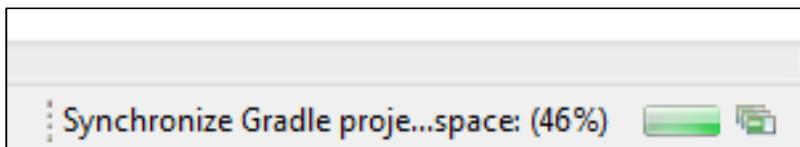
Paso 3. Vemos que el proyecto presenta errores y no reconoce las dependencias, ni como proyecto Gradle, ni como proyecto Spring. Hacemos click botón derecho encima del proyecto y vemos que no hay ninguna opción de **Gradle** ni de **Spring**, opciones que saldrían si el proyecto se reconociese como tal.

```
1 package com.example.prueba_gradle;
2
3 import org.springframework.boot.SpringApplication;
4
5 @SpringBootApplication
6 public class PruebaGradleApplication {
7
8     public static void main(String[] args) {
9         SpringApplication.run(PruebaGradleApplication.class, args);
10    }
11
12 }
13
14
15
16
17
18
19 }
```



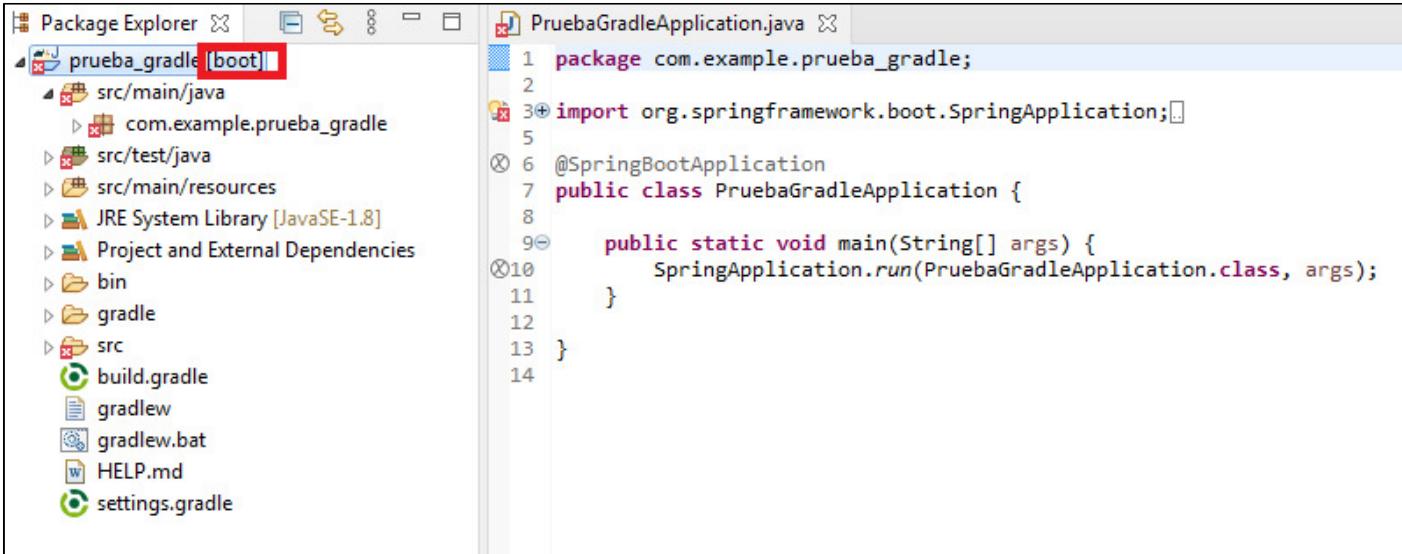
4. IMPORTACIÓN DESDE SPRING.IO

Paso 4. Para solventar este error debemos volver a configurar el proyecto como un proyecto Gradle. Hacemos click botón derecho encima del proyecto, y vamos a la opción **Configure/Add Gradle Nature**. Eclipse inicia un proceso de sincronización en el repositorio Gradle que dura unos segundos:



4. IMPORTACIÓN DESDE SPRING.IO

Paso 5. Después de este proceso, Eclipse por lo menos ya reconoce el proyecto como Spring. Solo falta un proceso mas para que todos los errores desaparezcan:

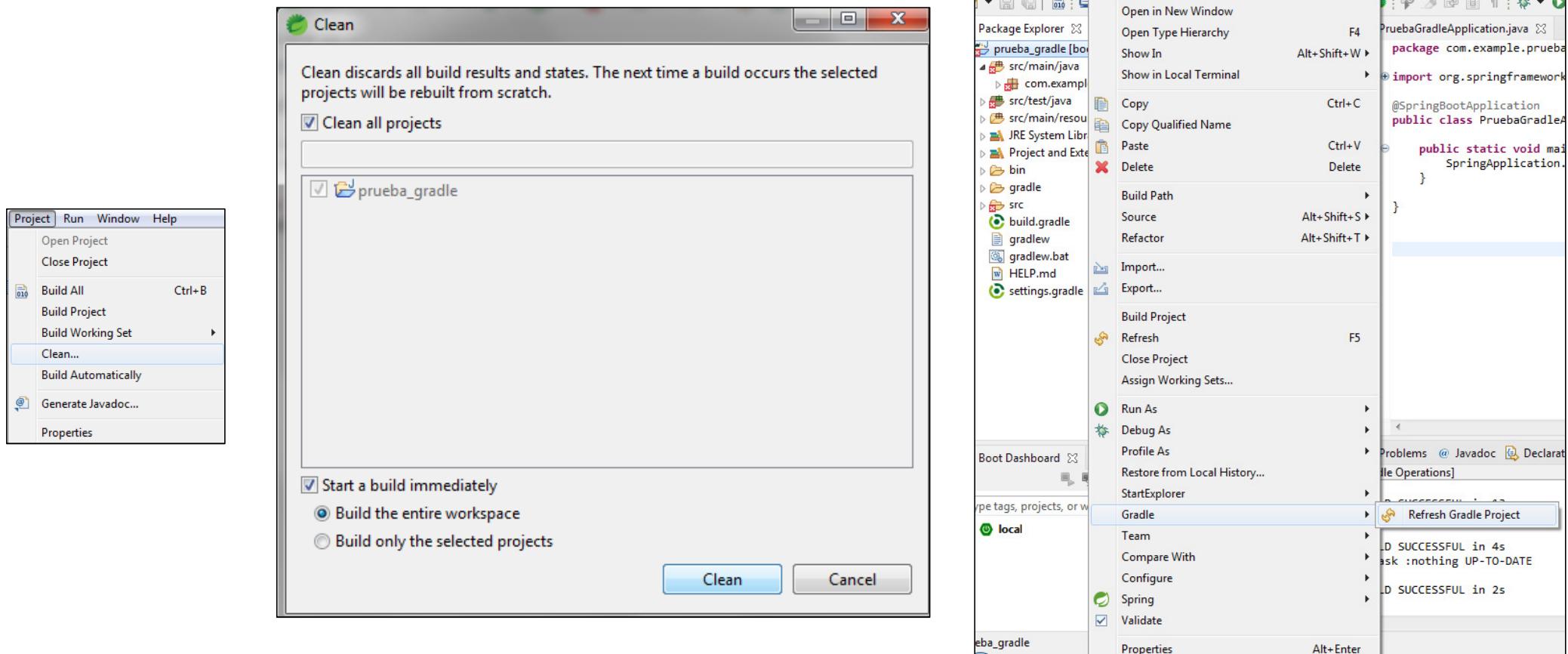


The screenshot shows the Eclipse IDE interface with the 'Package Explorer' view on the left and the 'PruebaGradleApplication.java' editor on the right. The 'Package Explorer' shows a project named 'prueba_gradle [boot]' containing several source and build files. The 'PruebaGradleApplication.java' editor displays the following Java code:

```
1 package com.example.prueba_gradle;
2
3 import org.springframework.boot.SpringApplication;
4
5 @SpringBootApplication
6 public class PruebaGradleApplication {
7
8     public static void main(String[] args) {
9         SpringApplication.run(PruebaGradleApplication.class, args);
10    }
11
12 }
13
14 }
```

4. IMPORTACIÓN DESDE SPRING.IO

Paso 6. Podemos hacer un Clean del proyecto para actualizar las dependencias de Gradle.



4. IMPORTACIÓN DESDE SPRING.IO

Paso 7. Con cualquiera de los pasos anteriores, desaparecen todos los errores de la aplicación y podemos arrancar el servidor Tomcat SpringBoot sin problemas:

The screenshot shows the Eclipse IDE interface with the following details:

- Package Explorer:** Shows the project structure for "prueba_gradle [boot]". It includes a "src/main/java" folder containing "com.example.prueba_gradle" with "PruebaGradleApplication.java" and "PruebaGradleApplication" (a configuration file). There are also "src/test/java", "src/main/resources", "JRE System Library [JavaSE-1.8]", "Project and External Dependencies", "bin", "gradle", "src", "build.gradle", "gradlew", "gradlew.bat", "HELP.md", and "settings.gradle".
- Code Editor:** Displays the content of "PruebaGradleApplication.java":

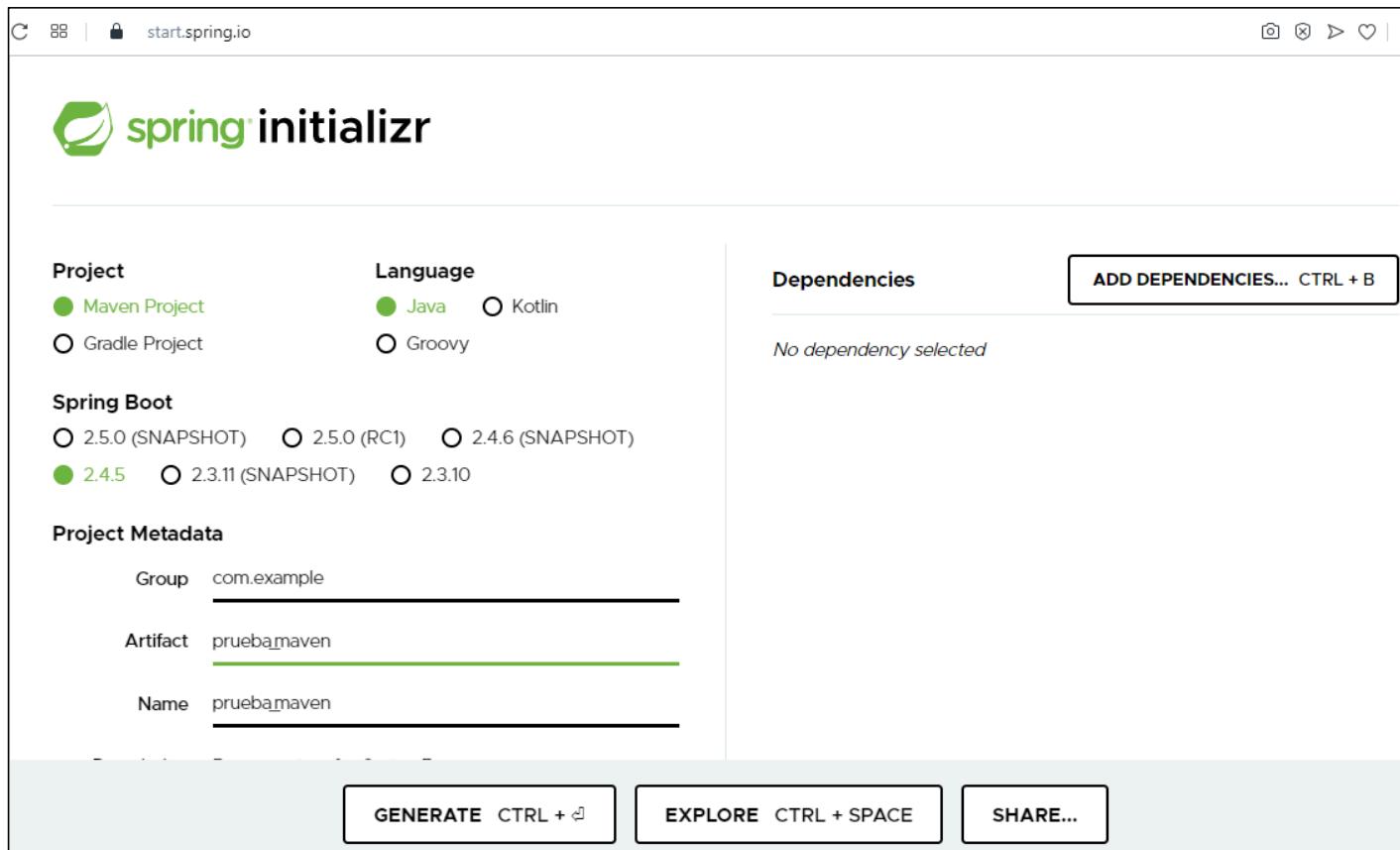
```
1 package com.example.prueba_gradle;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class PruebaGradleApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(PruebaGradleApplication.class, args);
11     }
12 }
13 }
```
- Console:** Shows the output of the application's execution:

```
<terminated> prueba_gradle - PruebaGradleApplication [Spring Boot App] Q:\Spring5\sts-4.9.0.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.1.v20201027-0507\jre\bin\javaw.exe (9)
.
.
.
:: Spring Boot ::          (v2.4.2)

2021-02-09 10:55:03.513  INFO 16312 --- [           main] c.e.p.PruebaGradleApplication
2021-02-09 10:55:03.519  INFO 16312 --- [           main] c.e.p.PruebaGradleApplication
2021-02-09 10:55:04.074  INFO 16312 --- [           main] c.e.p.PruebaGradleApplication
: Starting PruebaGradleApplication using Java 15.0.1 on HP-
: No active profile set, falling back to default profiles:
: Started PruebaGradleApplication in 0.949 seconds (JVM run
```

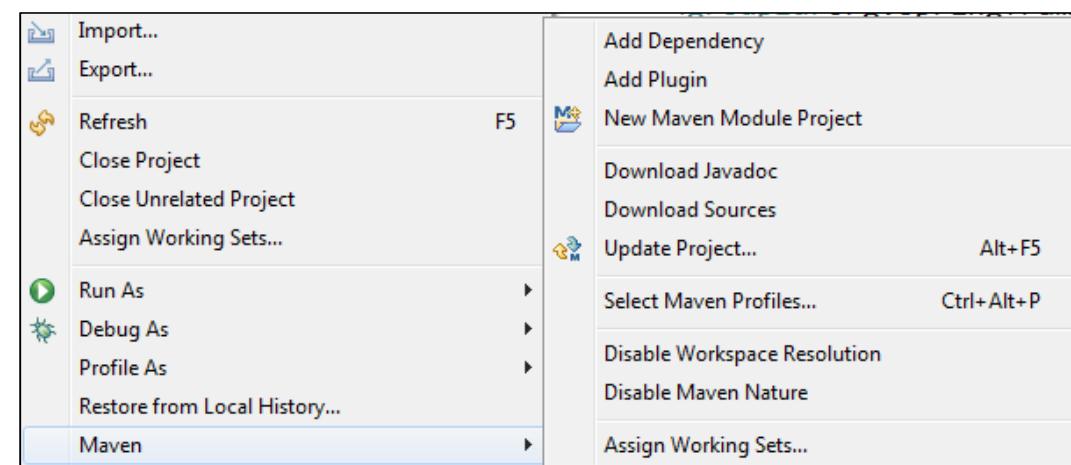
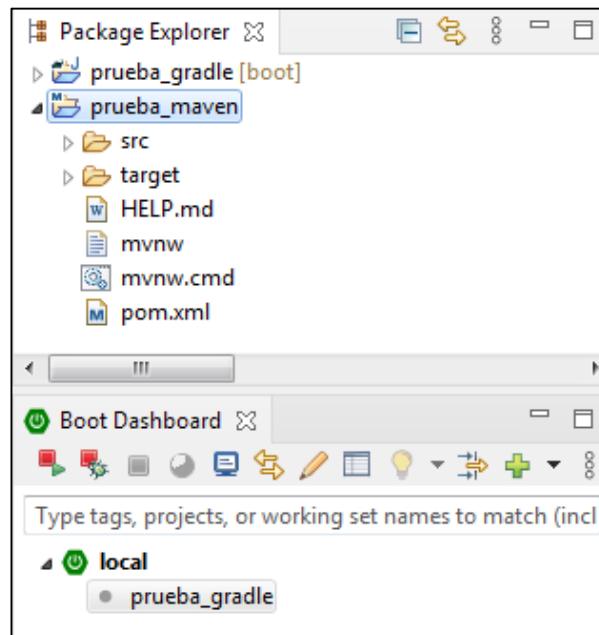
4. IMPORTACIÓN DESDE SPRING.IO

Paso 1. Vamos a la pagina <https://start.spring.io> y creamos un proyecto maven, que llamaremos prueba_maven. No agregamos ninguna dependencia:



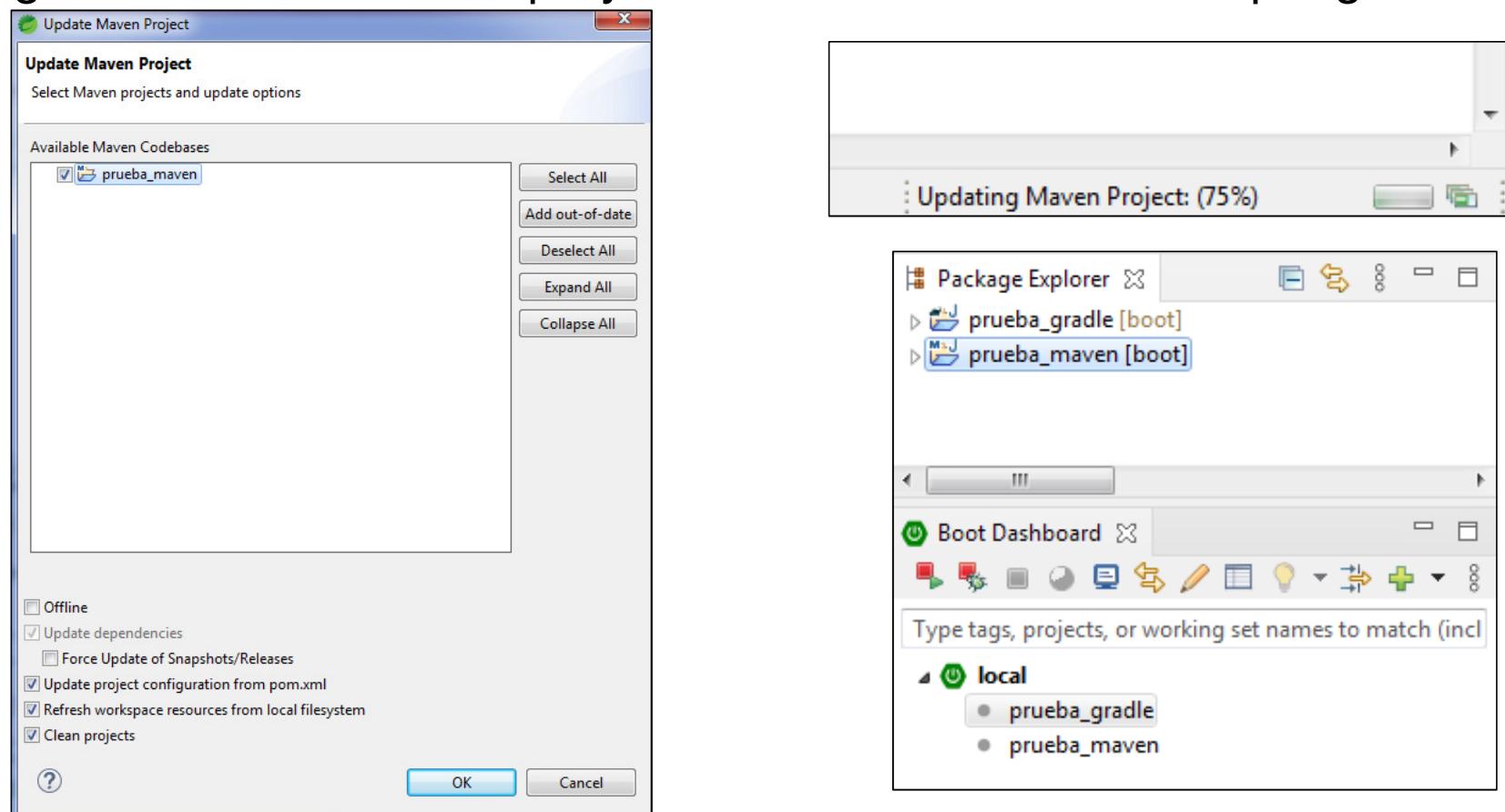
4. IMPORTACIÓN DESDE SPRING.IO

Paso 2. Descomprimimos prueba_maven.zip y lo importamos a Eclipse. Ahora el proyecto maven no presenta errores, se reconoce como un proyecto maven, pero no así como SpringBoot y por lo tanto no está desplegado en la pestana DashBoard. Debemos volver a actualizar el proyecto como maven. Hacemos click botón derecho encima del proyecto, y vamos a la opción **Maven/UpdateProject**.



4. IMPORTACIÓN DESDE SPRING.IO

Paso 3. Iniciamos el proceso de actualización con el repositorio Maven que dura unos segundos. Finalmente el proyecto si se reconoce como SpringBoot:



5. AGREGAR DEPENDENCIAS

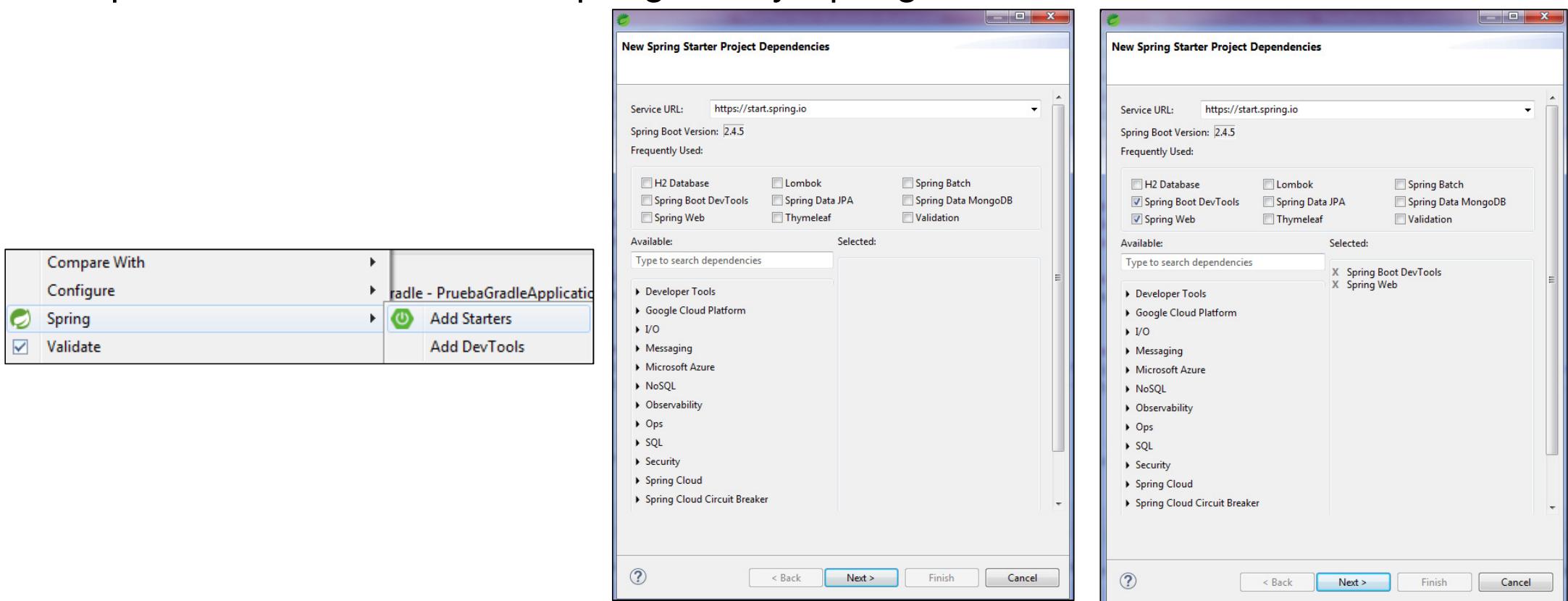
Paso 1. Nuestros proyectos de prueba solo contiene la dependencia por defecto que le distingue como proyecto Spring (spring-boot-starter y spring-boot-starter-test), pero no tiene ni la DevTools ni la Spring Web, necesarias para cualquier aplicación Spring

```
pom.xml
17     <java.version>11</java.version>
18     </properties>
19     <dependencies>
20         <dependency>
21             <groupId>org.springframework.boot</groupId>
22             <artifactId>spring-boot-starter</artifactId>
23         </dependency>
24
25         <dependency>
26             <groupId>org.springframework.boot</groupId>
27             <artifactId>spring-boot-starter-test</artifactId>
28             <scope>test</scope>
29         </dependency>
30     </dependencies>
```

```
build.gradle
1 plugins {
2     id 'org.springframework.boot' version '2.4.5'
3     id 'io.spring.dependency-management' version '1.0.11.RELEASE'
4     id 'java'
5 }
6
7 group = 'com.example'
8 version = '0.0.1-SNAPSHOT'
9 sourceCompatibility = '11'
10
11 repositories {
12     mavenCentral()
13 }
14
15 dependencies {
16     implementation 'org.springframework.boot:spring-boot-starter'
17     testImplementation 'org.springframework.boot:spring-boot-starter-test'
18 }
19
20 test {
21     useJUnitPlatform()
22 }
```

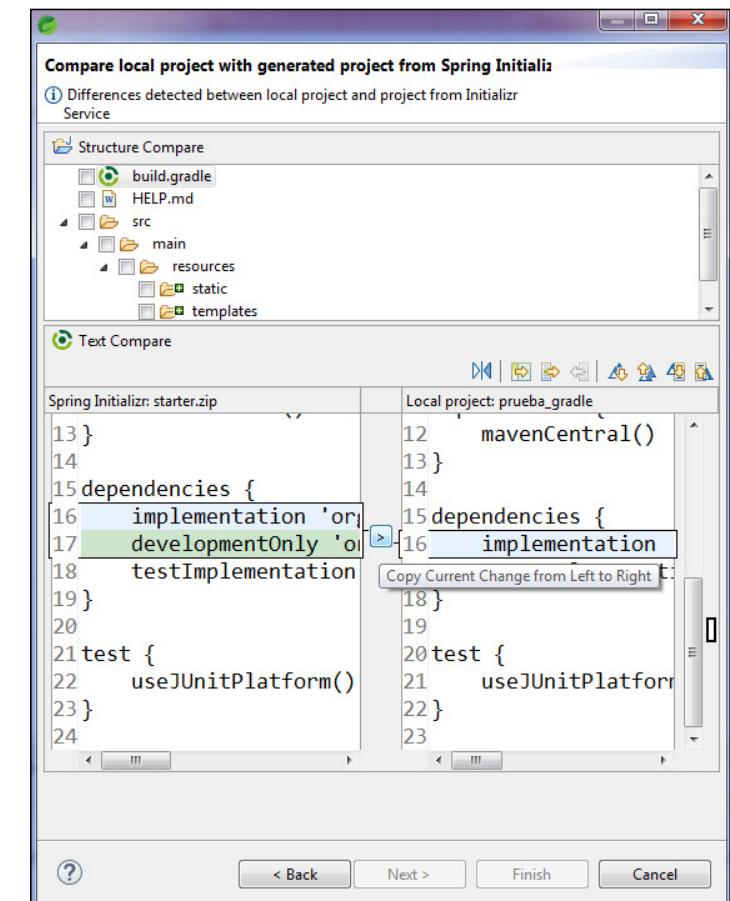
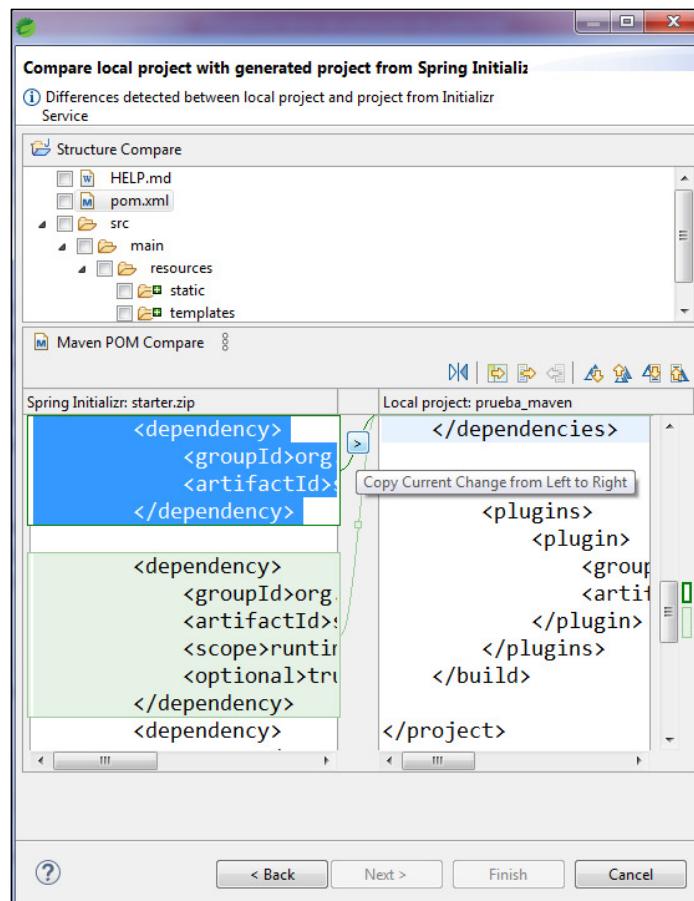
5. AGREGAR DEPENDENCIAS

Paso 2. Para agregar dependencias una vez ya creado el proyecto, debemos hacer click botón derecho y seleccionar la opción Spring/Add Starters. Seleccionamos las dependencias mínimas de Spring Web y Spring DevTools:



5. AGREGAR DEPENDENCIAS

Paso 3. En ambos tipos de proyectos (maven y gradle) se deben de agregar las dependencias haciendo click en un botón que las agrega gráficamente:



5. AGREGAR DEPENDENCIAS

Paso 4. Una vez hemos hecho click en el botón, se nos muestra como van a quedar modificados los ficheros pom.xml y build.gradle con las nuevas dependencias. Hacemos click en Finish:

The screenshot shows two side-by-side comparison windows for Maven and Gradle projects.

Maven POM Compare (Left):

- Structure Compare pane shows files: HELP.md, pom.xml, src/main/resources/static/templates.
- Maven POM Compare pane shows the difference between "Spring Initializr: starter.zip" and "Local project: prueba_maven".
 - The "Spring Initializr: starter.zip" dependency block is highlighted in blue.
 - The "Local project: prueba_maven" dependency block is highlighted in grey.

Text Compare (Right):

- Structure Compare pane shows files: build.gradle, HELP.md, src/main/resources/static/templates.
- Text Compare pane shows the difference between "Spring Initializr: starter.zip" and "Local project: prueba_gradle".
 - Line 16: "implementation 'org.springframework:spring-web:5.3.12'" is highlighted in green in the left pane.
 - Line 17: "developmentOnly 'org.springframework:spring-test:5.3.12'" is highlighted in green in the left pane.
 - Line 18: "testImplementation 'org.junit.jupiter:junit-jupiter-api:5.8.1'" is highlighted in green in the right pane.

5. AGREGAR DEPENDENCIAS

Paso 5. Vemos las dependencias agregadas en los archivos pom.xml y build.gradle:

```
prueba_maven/pom.xml ✘
24
25    <dependency>
26        <groupId>org.springframework.boot</groupId>
27        <artifactId>spring-boot-starter-test</artifactId>
28        <scope>test</scope>
29    </dependency>
30    <dependency>
31        <groupId>org.springframework.boot</groupId>
32        <artifactId>spring-boot-starter-web</artifactId>
33    </dependency>
34    <dependency>
35        <groupId>org.springframework.boot</groupId>
36        <artifactId>spring-boot-devtools</artifactId>
37        <scope>runtime</scope>
38        <optional>true</optional>
39    </dependency>
40 </dependencies>
```

```
build.gradle ✘
1 plugins {
2     id 'org.springframework.boot' version '2.4.5'
3     id 'io.spring.dependency-management' version '1.0.11.RELEASE'
4     id 'java'
5 }
6
7 group = 'com.example'
8 version = '0.0.1-SNAPSHOT'
9 sourceCompatibility = '11'
10
11 repositories {
12     mavenCentral()
13 }
14
15 dependencies {
16     implementation 'org.springframework.boot:spring-boot-starter-web'
17     developmentOnly 'org.springframework.boot:spring-boot-devtools'
18     testImplementation 'org.springframework.boot:spring-boot-starter-test'
19 }
20
21 test {
22     useJUnitPlatform()
23 }
```

5. AGREGAR DEPENDENCIAS

Paso 6. Podemos observar que ambos proyectos ya han agregado en su descripción [devtools] y ya podemos reiniciar nuestro servidor:

