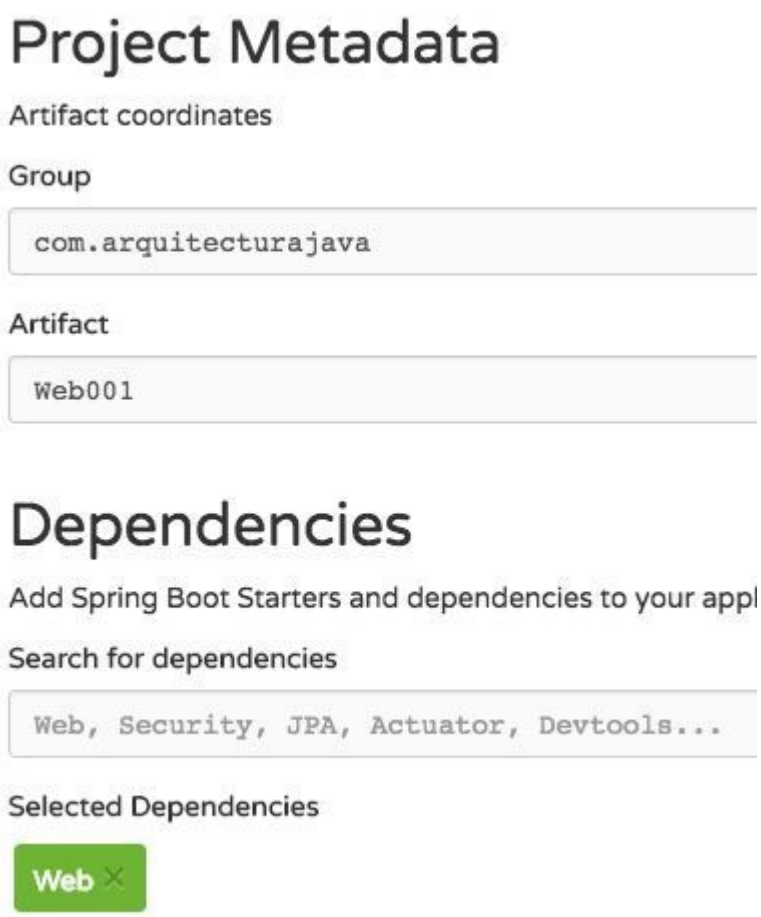
Spring Boot WAR es un concepto muy interesante . Si nosotros construimos una aplicación con Spring Boot habitualmente podemos usar Spring Initializer para generar un proyecto Maven o Gradle con unas dependencias u otras. Por ejemplo podemos utilizar un proyecto con Spring MVC.



La sencillez de Spring Boot es absoluta. Simplemente con poner que queremos un proyecto web es suficiente . No hay ni punto de comparación con un proyecto plano de Maven en el cual tenemos que dar de alta las diferentes dependencias.

Spring Boot WAR

En principio es facil preferir Spring Boot sobre un proyecto maven clásico ya que simplifica mucho . El problema es que los proyectos de Spring Boot llevan integrado un Tomcat y se despliegan como MicroServicios por defecto.

Así pues necesitaríamos usar Docker etc. La realidad es que la mayoría de la gente no esta desplegando arquitecturas de MicroServicios sino aplicaciones clásicas y algún Microservicio. ¿Podemos hacer uso de Spring Boot sin desplegar Microservicios?. Si que se puede pero nos veremos obligados a realizar algunos cambios en el proyecto . Vamos a verlo:

```
package com.arquitecturajava.web001;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;

@SpringBootApplication
public class Web001Application {
```

```
	public static void main(String[] args) {
		SpringApplication.run(Web001Application.class, args);
	}
}
```

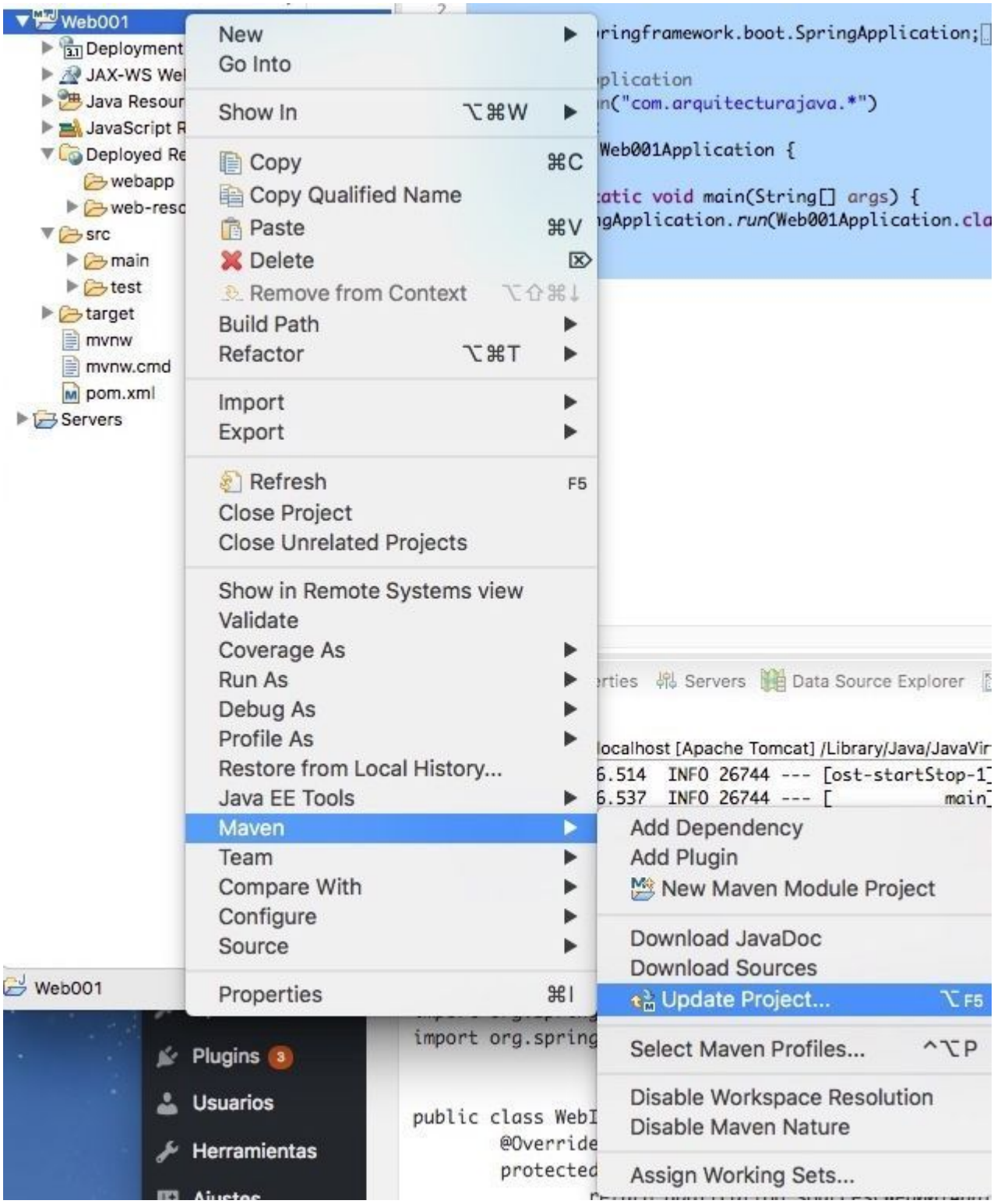
Si nos fijamos el proyecto en principio es un proyecto de Maven de tipo JAR con un único fichero y una serie de dependencias de Spring Boot a nivel del POM. El primer paso es modificar las dependencias y añadir que la de Tomcat es provided. De esta forma estamos diciendo que el servidor sobre el que desplegamos ya incluye todo lo necesario y no es necesario incluir Tomcat.

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-tomcat</artifactId>
<scope>provided</scope>
</dependency
</dependencies>
[/xml]
```

El segundo paso es evidentemente modificar el proyecto que es de tipo jar a tipo war

```
&lt;groupId>com.arquitecturajava&lt;/groupId>
	&lt;artifactId>Web001&lt;/artifactId>&lt;version>0.0.1-
SNAPSHOT&lt;/version>
	&lt;packaging>WAR&lt;/packaging>
```

Decirle a eclipse que refresque a nivel de maven:


Web001
Deployment
JAX-WS Wel
Java Resour
JavaScript R
Deployed Re
webapp
web-resc
src
main
test
target
mvnw
mvnw.cmd
pom.xml
Servers
New
Go Into
Show In ⌥⌘W
Copy ⌘C
Copy Qualified Name
Paste ⌘V
Delete
Remove from Context ⌥⇧⌘↓
Build Path
Refactor ⌥⌘T
Import
Export
Refresh F5
Close Project
Close Unrelated Projects
Show in Remote Systems view
Validate
Coverage As
Run As
Debug As
Profile As
Restore from Local History...
Java EE Tools
Maven
Team
Compare With
Configure
Source
Properties ⌘I
Add Dependency
Add Plugin
New Maven Module Project
Download JavaDoc
Download Sources
Update Project... ⌥F5
Select Maven Profiles... ^⌥P
Disable Workspace Resolution
Disable Maven Nature
Assign Working Sets...
Servers
Data Source Explorer
Web001
Plugins 3
Usuarios
Herramientas
import org.spring
public class WebI
@Override
protected

El tercer paso es añadir una clase de WebInitializer a nivel de nuestro proyecto que se encargue de registrar la clase por defecto construida.

```
package com.arquitecturajava.web001;

import org.springframework.boot.builder.SpringApplicationBuilder;
import
org.springframework.boot.web.support.SpringBootServletInitializer;

public class WebInicializador extends SpringBootServletInitializer {
        @Override
        protected SpringApplicationBuilder
configure(SpringApplicationBuilder application) {
                return application.sources(Web001Application.class);
        }

}
```

Ya solo nos queda construir un controlador con el que probar y referenciarle adecuadamente:

```
package com.arquitecturajava.web001;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
@RestController
public class ControladorHola {

	@RequestMapping("/hola")
	public String hola() {

		return "hola spring boot";
	}
}
```

Le referenciamos :

```
package com.arquitecturajava.web001;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;

@SpringBootApplication
@ComponentScan("com.arquitecturajava.*")
@EnableWebMvc
public class Web001Application {

	public static void main(String[] args) {
		SpringApplication.run(Web001Application.class, args);
	}
}
```

Es momento de ejecutar nuestra aplicacion con run on server:

Acabamos de utilizar el concepto de Spring Boot WAR para construir nuestra aplicación sin MicroServicios usando Spring Boot.

Otros artículos relacionados:

1. Spring Boot AOP y rendimiento
2. ¿Qué es Spring Boot?
3. El porqué de los MicroServicios
4. Java EE MicroServices con Payara
5. Spring Boot