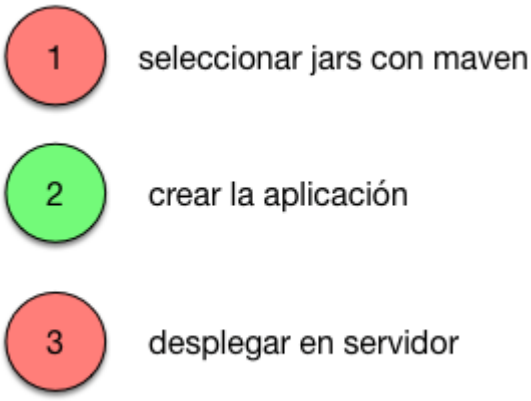
Tabla de Contenidos

- Primeros pasos sin Spring Boot
  - Problemas
- Spring Boot
- String Boot Starter y simplificaciones
  - Starters y ventajas
- Construyendo la aplicación
- Spring Boot Controller
  - Spring Boot Tomcat y Docker
- Spring Boot WAR
- Spring Boot vs. Quarkus: ¿Cuál elegir?
  - 1. Propósito y enfoque
  - 2. Rendimiento
  - 3. Ecosistema y compatibilidad
  - 4. Compatibilidad con GraalVM
  - 5. Cuándo usar cada uno
  - Conclusión
- Otros artículos relacionados

Spring Boot es una de las tecnologías dentro del mundo de Spring de las que más se usa actualmente .¿Qué es y cómo funciona Spring Boot? . Para entender el concepto primero debemos reflexionar sobre cómo construiamos aplicaciones con Spring Framework antiguamente.

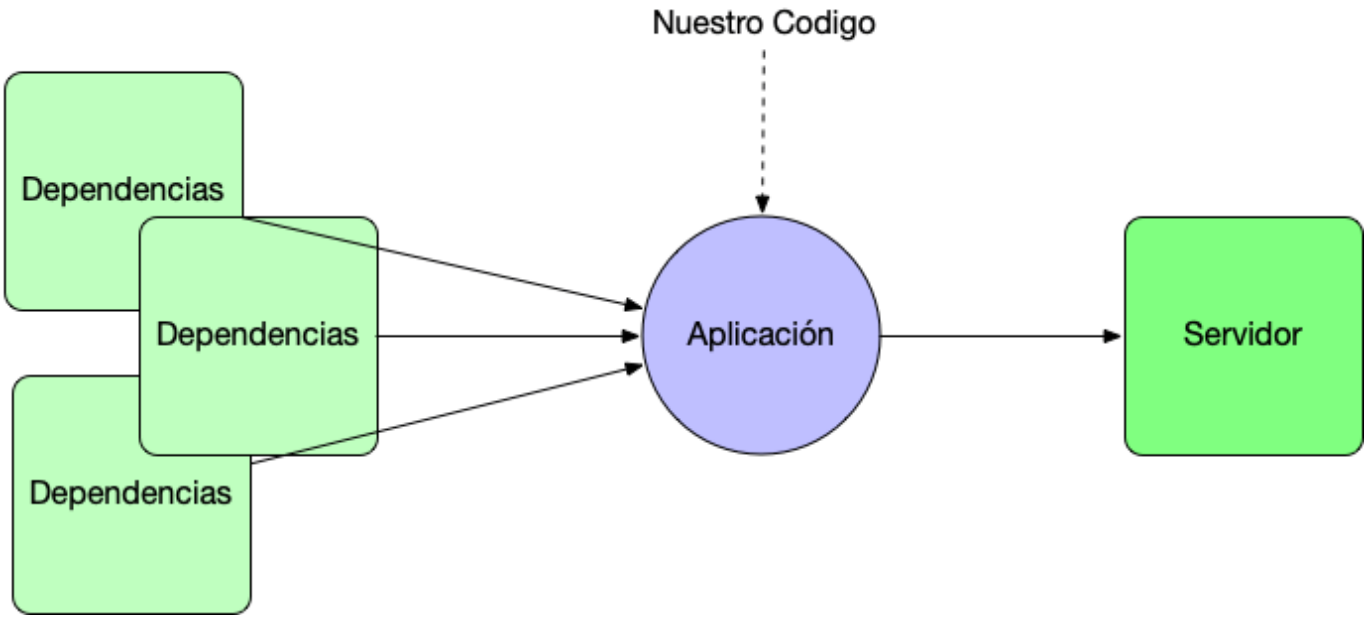


Primeros pasos sin Spring Boot

Fundamentalmente existian tres pasos a realizar . El primero es crear un proyecto Maven/Gradle y descargar las dependencias necesarias. En segundo lugar esta el proceso de crear la aplicación , pero para ello nos guste o no debemos abordar un proceso amplio de configuración de la aplicación , con ficheros XML o anotaciones y configuraciones muy especificas que muchas veces solo un experto era capaz de abordar con garantias. Por último debiamos desplegarla en un servidor.

Problemas

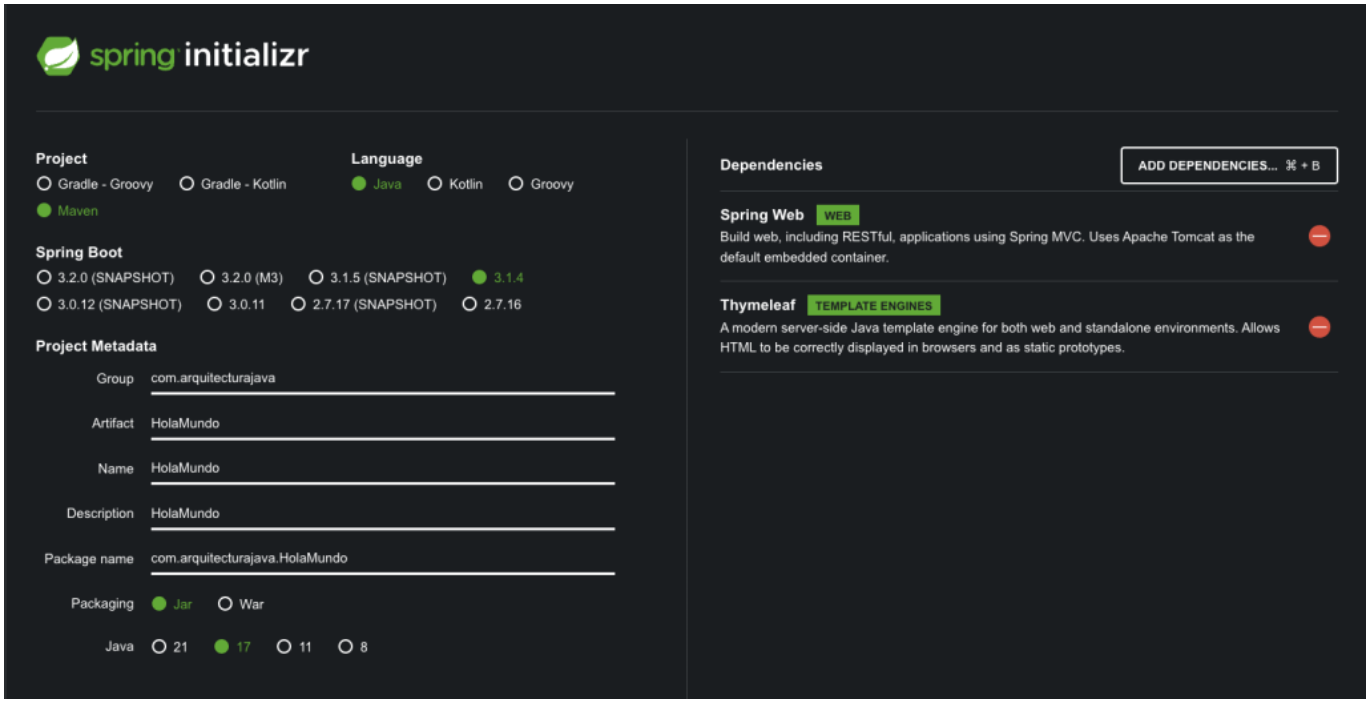
Si nos ponemos a pensar un poco a detalle en el tema , únicamente el paso dos es una tarea de desarrollo y dentro de esa tarea 2 incluso la parte de configuración no esta claro que sea desarrollo en sí. Son cosas que están más orientados a infraestructura que al desarrollo en sí mismo.



No deberíamos tener que estar eligiendo continuamente las dependencias y el servidor de despliegue así como realizar una configuración inicial .Esto es un tema solo para expertos y que aporta realmente poco debería ser automatico.

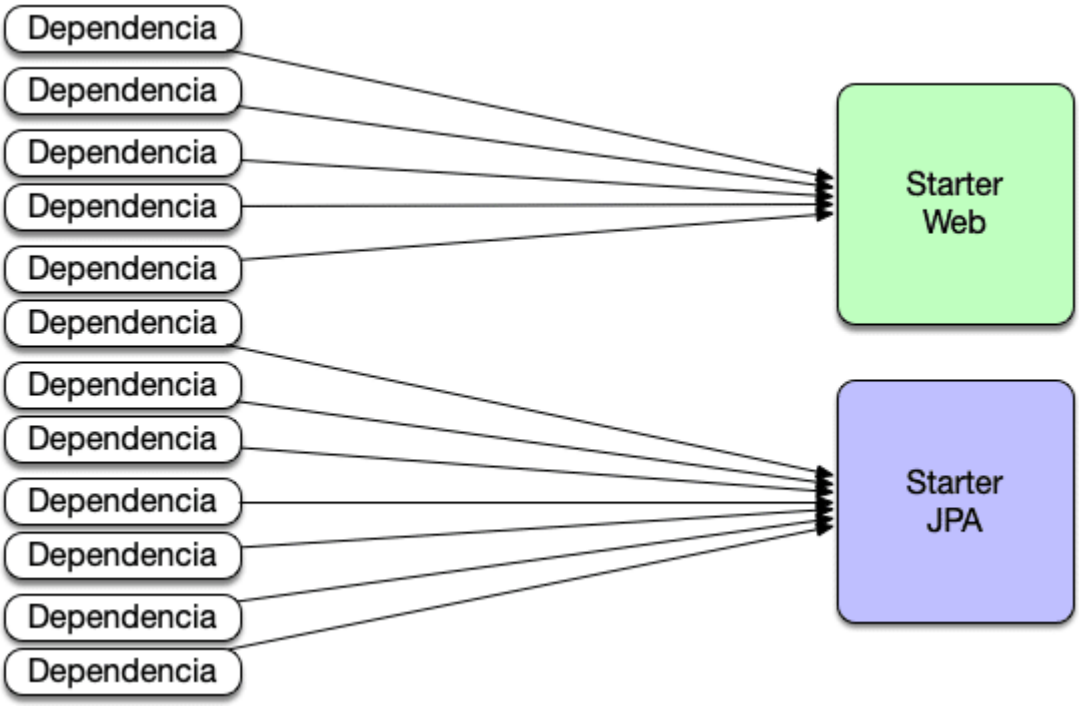
Spring Boot

SpringBoot nace con la intención de simplificar los pasos 1 y 3 . Simplificar la configuración , que nos podamos centrar en el desarrollo de nuestra aplicación. ¿Cómo funciona?. El enfoque es sencillo y lo entenderemos realizando un ejemplo. Para ello nos vamos a conectarnos al asistente de Boot que se denomina Spring Initializer.



String Boot Starter y simplificaciones

El asistente es intuitivo , elegimos el package al que queremos que nuestras clases pertenezcan , elegimos el nombre del proyecto y por último las dependencias. Eso sí ya no se trata de elegir JAR por JAR sino por tipo de aplicación que necesitamos a este concepto se le denomina Spring Starter. Por lo tanto en vez de tener que elegir 10 o 20 dependencias es mucho más cómodo elegir 2 starters y Spring Boot se encarga del resto.

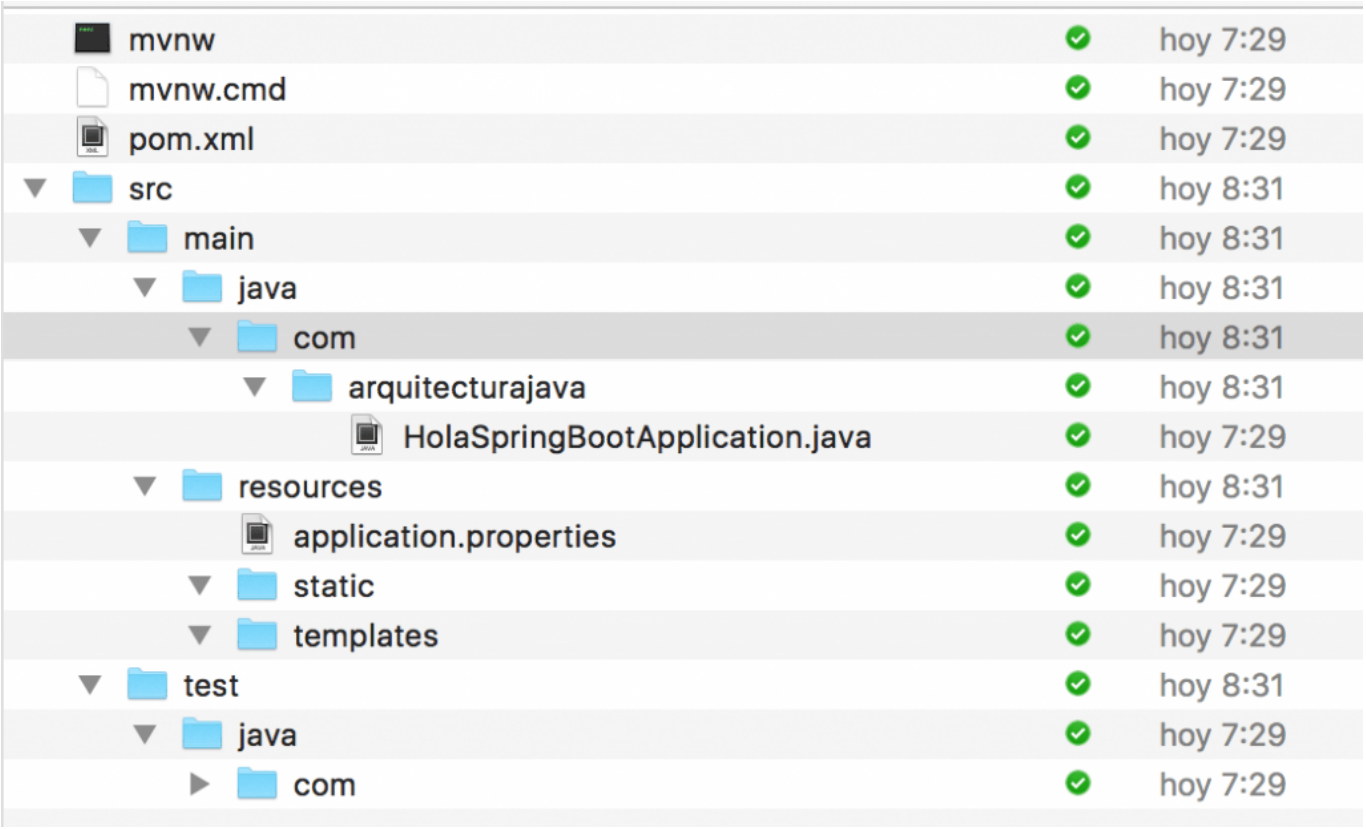


Starters y ventajas

La ventaja es abismal ya que no solo reduzco las dependencias sino también los temas a configurar ya que los starter pueden encargarse de añadir carpetas etc.

Construyendo la aplicación

En este caso voy a construir una aplicación Spring MVC y elijo la dependencia web o Starter Web. Pulsamos generar proyecto y nos descargará un proyecto Maven en formato zip . Descomprimimos el proyecto y este es su contenido.



Una aplicación de Spring con estructura Maven totalmente configurada y carpetas adicionales para cubrir nuestras necesidades . El siguiente paso importar esta aplicación a nuestro Eclipse. Vamos a ver el contenido de la clase HolaSpringBootApplication

```
package com.arquitecturajava;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class HolaSpringBootApplication {

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

```
}
```

Esta clase es la encargada de arrancar nuestra aplicación de Spring a diferencia de un enfoque clásico no hace falta desplegarla en un servidor web ya que Spring Boot provee de uno (contiene un Tomcat embebido)

Spring Boot Controller

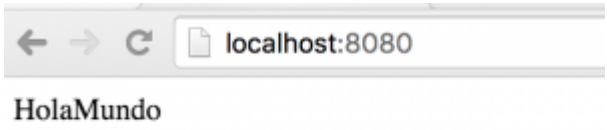
Vamos a construir un controlador de HolaMundo sencillo:

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@RestController
public class ControladorHola {

    @RequestMapping("/")
    String hola() {
        return "HolaMundo";
    }
}
```

Este controlador registra la url de / para que nos devuelva “HolaMundo” . Es momento de ejecutar nuestra aplicación como una aplicación de consola utilizando botón derecho run as Java Application en el fichero de HolaSpringBootApplication. Esto abrirá un servidor web y accederemos a la url.

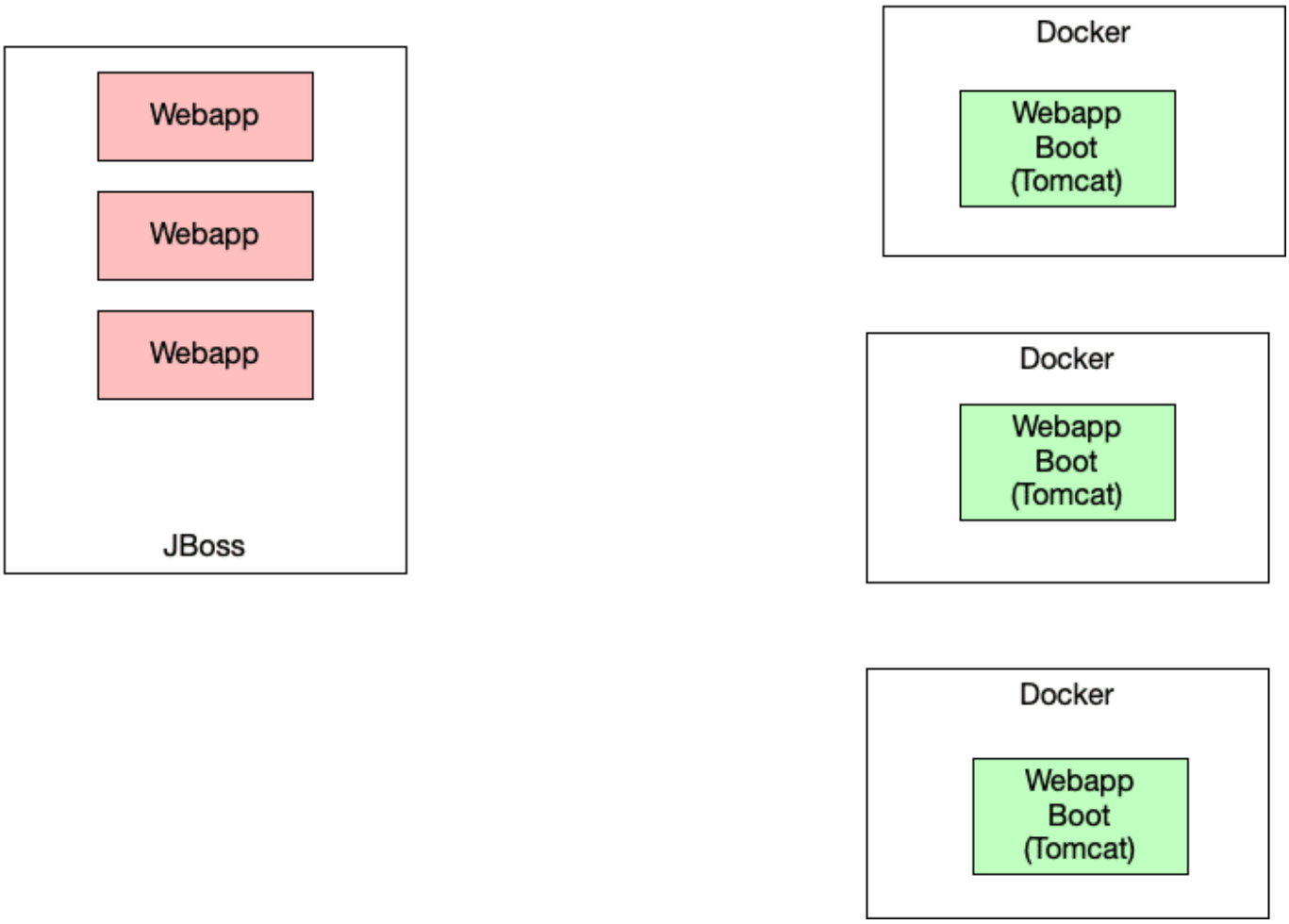


SpringBoot nos ha simplificado toda la operativa a la hora de construir la aplicación prácticamente no hemos tenido que seleccionar dependencias de Spring y no ha hecho falta definir ningun servidor Tomcat en nuestro entorno de desarrollo ya que Spring Boot trae uno integrado. Vamos a ver un ejemplo desde cero con un video del curso gratuito de Spring Boot ;).

Spring Boot Tomcat y Docker

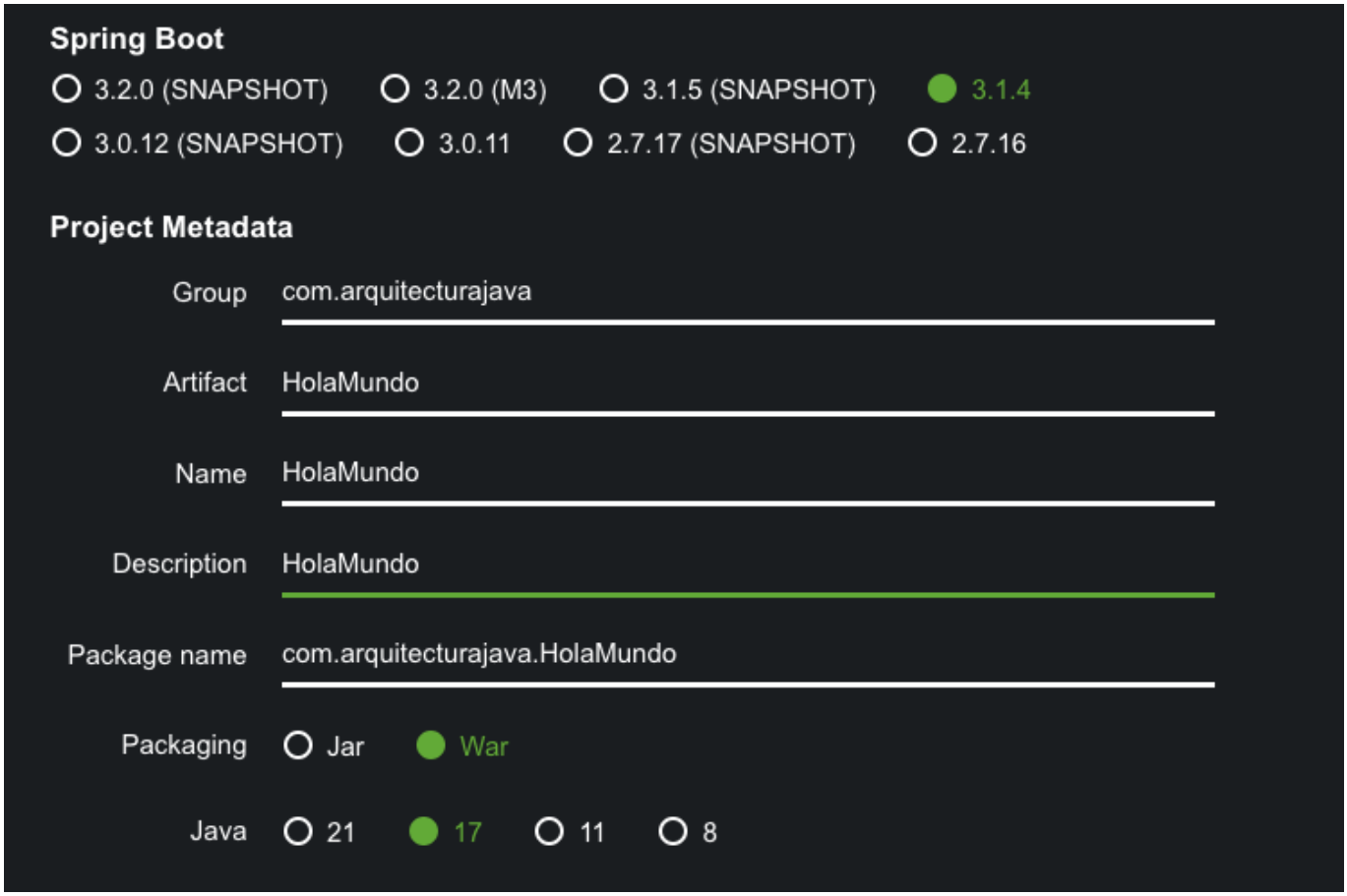
¿Porque Spring Boot trae integrado Tomcat? . Muy sencillo porque a partir de ahora los despliegues no se van a realizar en Servidores Web Standard que almacenan decenas de aplicaciones sino que cada una de las aplicaciones se va a desplegar en un contenedor Docker completamente aislada del resto e independiente . El contenedor necesita que la

aplicación sea completamente operativa por si sola.



Spring Boot WAR

Si tu problema es que hoy por hoy no tienes contenedores Docker ni tienes Kubernetes entre tus herramientas de despliegue , no hay problema . Puedes seguir usando Spring Boot y desplegarlo en un entorno de Tomcat o JBoss directamente simplemente modificando el arranque y haciendo que nuestra aplicación en vez de desplegarse como JAR se despliegue como WAR.



Esto no cambiara en gran manera la clase principal

```
package com.arquitecturajava.HolaMundo;

import org.springframework.boot.builder.SpringApplicationBuilder;
```

```
import
org.springframework.boot.web.servlet.support.SpringBootServletInitiali
zer;

public class ServletInitializer extends SpringBootServletInitializer {

        @Override
        protected SpringApplicationBuilder
configure(SpringApplicationBuilder application) {
                return
application.sources(HolaMundoApplication.class);
        }

}
```

Spring Boot vs. Quarkus: ¿Cuál elegir?

Tanto Spring Boot como Quarkus son frameworks populares para el desarrollo de aplicaciones Java, pero tienen enfoques diferentes. Aquí te presentamos una comparación clave entre ambos.

1. Propósito y enfoque

- Spring Boot: Facilita la creación de aplicaciones empresariales con mínima configuración. Es ideal para monolitos y microservicios, con un ecosistema robusto.
- Quarkus: Optimizado para microservicios y entornos cloud-native, se destaca por su rapidez y bajo consumo de memoria, ideal para contenedores y funciones serverless.

2. Rendimiento

- Spring Boot: Es flexible, pero su arranque puede ser más lento debido a su sistema de auto-configuración.
- Quarkus: Con GraalVM, Quarkus permite compilar imágenes nativas, ofreciendo tiempos de

arranque más rápidos y menor consumo de memoria.

3. Ecosistema y compatibilidad

- Spring Boot: Ofrece un ecosistema muy maduro y amplio, con soporte para muchas bibliotecas.
- Quarkus: Aunque es más nuevo, su integración con herramientas como Hibernate es sólida, aunque su ecosistema aún está en crecimiento.

4. Compatibilidad con GraalVM

- Spring Boot: Tiene soporte en desarrollo para GraalVM, pero requiere ajustes.
- Quarkus: Optimizado para GraalVM desde su creación, facilitando la generación de imágenes nativas.

5. Cuándo usar cada uno

- Usa Spring Boot si…: Necesitas flexibilidad y un ecosistema probado para aplicaciones empresariales.
- Usa Quarkus si…: Buscas rapidez en el arranque, menor consumo de recursos y desplegar microservicios en entornos cloud-native.

Conclusión

Elige Spring Boot para proyectos empresariales robustos y flexibles. Opta por Quarkus si tu prioridad es el rendimiento en la nube y los entornos serverless.

Otros artículos relacionados

- Spring Boot Starter ,un concepto fundamental
- Curso Spring Boot y MicroServicios
- Java Override y encapsulación
- JPA Join Fetch y su uso.