2.1 - Tecnologías de Servicios de Web

Tema 5 – Construcción y despliegue

Universidad Rey Juan Carlos

Micael Gallego
micael.gallego@urjc.es
@micael_gallego

code URJC

CloudApps

MASTER

Tema 5 – Construcción y despliegue

Tema 5.1 – Construcción y despliegue en Spring Boot

Universidad Rey Juan Carlos

Micael Gallego
micael.gallego@urjc.es
@micael_gallego

code URJC

Construcción en Spring Boot

- **Introducción**
- Empaquetado en .jar
- Empaquetado en native image

Introducción

- Los servicios web Java se **desarrollan en un IDE** (VSCode, IntelliJ, Eclipse, Netbeans…)
- Para que pueda ponerse a disposición de los usuarios en **producción** pasa por las siguientes fases:
  - **Construcción**
  - **Empaquetado y distribución**
  - **Despliegue y ejecución**

Introducción

- Construcción:
  - **Fases:**
    - Descarga de librerías
    - Generación de código
    - Compilación
    - Ejecución de tests
  - En Java las herramientas de construcción son **Maven** o **Gradle**

https://maven.apache.org/

https://gradle.org/

Introducción

- **Empaquetado y distribución**
  - Los servicios web Java tienen diferentes formatos de empaquetado y distribución
    - Fichero .war
    - Fichero .jar
    - Imagen nativa GraalVM
    - Imagen Docker

https://www.baeldung.com/java-jar-war-packaging

Introducción

- **Empaquetado y distribución**
  - **Fichero .war**
    - Necesita un servidor de aplicaciones **Java EE** para ejecutarse

http://docs.spring.io/spring-boot/docs/current/reference/html/howto-traditional-deployment.html

Introducción

- **Empaquetado y distribución**
  - **Fichero .war**
    - Era el habitual **hace 10/15 años** porque, pero ahora **no es la opción preferida**
    - El inconveniente es que la app está limitada por la versión de Java del servidor y las versiones de sus librerías (es **muy costoso de actualizar**)
    - No lo vamos a ver en detalle

Introducción

- **Empaquetado y distribución**
  - **Fichero .jar**
    - Incluye un **servidor web** integrado y todas las **librerías** necesarias
    - Necesita una **JVM instalada** en el sistema
    - Facilita la evolución de la aplicación

Introducción

code URJC

- **Empaquetado y distribución**
  - **Imagen nativa GraalVM**

    - Disponible desde **2019**
    - Fichero optimizado y nativo del Sistema Operativo
    - Incluye todo lo necesario para ejecutar la aplicación (**servidor web, librerías y JVM**)
    - Arranca **mucho más rápido y consume menos memoria** que las otras alternativas

https://www.graalvm.org/latest/reference-manual/native-image/

Introducción

- **Empaquetado y distribución**
  - **Imagen nativa GraalVM**
    - La generación del paquete optimizado tarda **varios minutos**
    - Los servicios web tienen que **adaparse** para poder usar este formato
    - **Spring Boot** lo soporta desde Nov del 2022
    - **Quarkus** lo soporta desde 2019

Introducción

code URJC

- **Empaquetado y distribución**

  - **Imagen Docker**
    - Disponible desde el **2013** y muy popular
    - Incluye todo lo necesario para ejecutar la aplicación (**servidor web, librerías y JVM**)
    - Necesita Docker instalado
    - Puede empaquetar .war+servidor, .jar+JVM o native images

https://www.docker.com/

Introducción

- **Empaquetado y distribución**

  - **Imagen Docker**
    - Docker permite ejecutar apps implementadas con **cualquier tecnología de desarrollo**
    - Varias aplicaciones pueden ejecutarse de forma **aislada** en el mismo servidor
    - Ofrece mecanismo estandarizado de distribuir aplicaciones (**Registros de imágenes**)
    - Lo estudiaremos más adelante en detalle

Introducción

- **Despligue y ejecución**
  - El despligue y la ejecución dependen mucho de la tecnología de **empaquetado** y **distribución**
  - **Docker** es el único formado de empaquetado que hemos visto que ofrece un sistema completo de despliegue y ejecución
  - **Kubernetes** se basa en Docker para desplegar y ejecutar servicios en un cluster de servidores
  - Se estudiará más adelante

Construcción en Spring Boot

- Introducción
- **Empaquetado en .jar**
- Empaquetado en native image

Empaquetado en .jar

code URJC

ejem1

- **Un proyecto SpringBoot incluye el plugin Maven de Spring Boot encargado de empaquetar el .jar**
  - Crea un fichero .jar con todas las librerías de la aplicación y el servidor web en la carpeta **target**

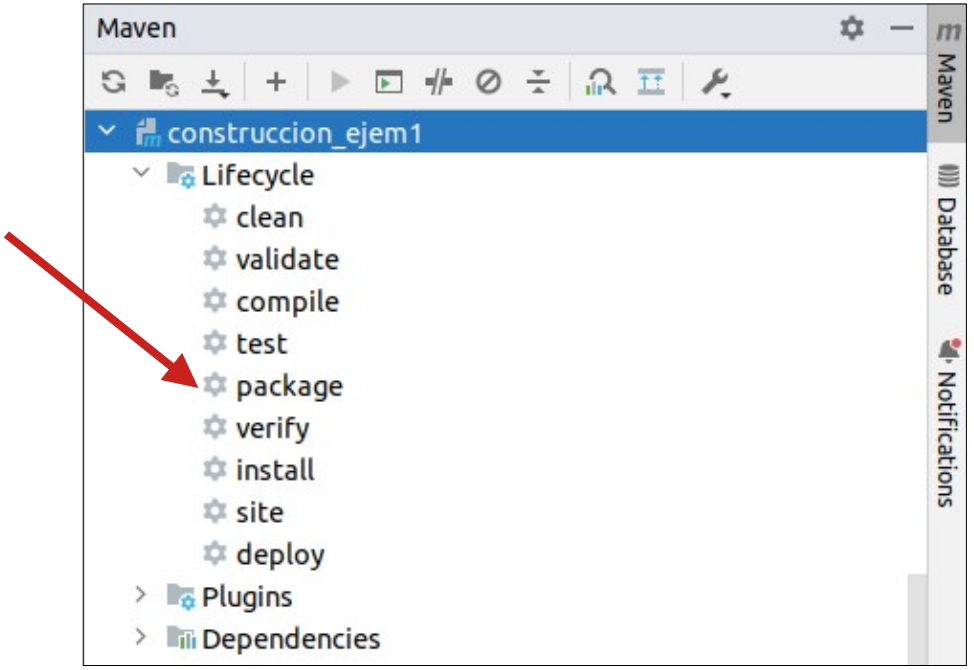
```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

Empaquetado en .jar

code URJC

ejem1

- **¿Cómo generar el .jar?**
  - Desde la línea de comandos (donde está el pom.xml)
    - $ mvn package
  - Desde IntelliJ



Empaquetado en .jar

code URJC

ejem1

- **Resultado**
  - Fichero <nombreproyecto>_<version>.jar en la carpeta target

```
construccion_ejem1-1.0.0.jar
```

Empaquetado en .jar

code URJC

ejem1

- **Ejecutar la app web con el .jar**
  - Es requisito tener instalado un Java JRE
  - Ejecutamos el comando

```
$ java -jar construccion_ejem1-1.0.0.jar
```

  - Para finalizar la aplicación ejecutar **Ctrl+C** en la consola (se envía una señal de apagado **SIGTERM**)
  - También se puede habilitar una **URL REST** para **apagar en remoto** (protegida por contraseña)

Empaquetado en .jar

- **Configuración de la aplicación**
  - Al ejecutar la aplicación se pueden sobreescribir las propiedades de configuración del fichero **application.properties** con:
    - 1) Parámetros de la línea de comandos
    - 2) Variables de entorno
    - 3) Fichero application.properties junto al fichero .jar

http://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-external-config.html

Empaquetado en .jar

code URJC

ejem1

- **Configurar la app web con el .jar**
  - Configuración de puerto

```
$ java -jar construccion_ejem1-1.0.0.jar --server.port=8081
```

  - Configuración de base de datos

```
$ java -jar construccion_ejem1-0.0.1.jar \
  --spring.datasource.url=jdbc:mysql://localhost/test \
  --spring.datasource.username=root \
  --spring.datasource.password=pass \
  --spring.jpa.hibernate.ddl-auto=create-drop
```

Empaquetado en .jar

code URJC

ejem1

- **Perfiles**
  - Es habitual que tengamos valores de propiedades diferentes en varios entornos (desarrollo y producción)
  - Podemos tener diferentes ficheros de configuración
    - application-prod.properties
    - application-dev.properties
  - Al ejecutar la aplicación se selecciona el perfil

```
$ java -jar -Dspring.profiles.active=prod construccion_ejem1-1.0.0.jar
```

http://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-profiles.html

Construcción en Spring Boot

- Introducción
- Empaquetado en .jar
- **Empaquetado en native image**

Empaquetado en native image

code URJC

https://www.graalvm.org/latest/reference-manual/native-image/

Empaquetado en native image

code URJC

- Las imágenes nativas de GraalVM son **ejecutables específicos del sistema operativo**
- **No requieren una JVM** para ejecutarse (la llevan integrada)
- Inician su ejecución mucho **más rápido** que las aplicaciones con JVM y consumen **menos memoria**
- Es una tecnología **muy novedosa** (primera versión en 2019) y está **evolucionando** muy rápido

https://docs.spring.io/spring-boot/docs/current/reference/html/native-image.html

Empaquetado en native image

code URJC

- Las **aplicaciones** y las **librerías** tienen que **adaptarse** para poder empaquetarse en native images
- Cada vez hay **más librerías** que lo soportan, pero todavía hay muchas que no están preparadas
- La adaptación es necesaria porque las imágenes nativas tienen que conocer **qué clases y métodos se usan** en la aplicación, para eliminar los que no se usan
- Si la app usa **reflexión** de Java, tiene que especificar en qué clases, para no eliminarlas (**reachability metadata**)

https://docs.spring.io/spring-boot/docs/current/reference/html/native-image.html

Empaquetado en native image

code URJC

- Las aplicaciones Spring Boot ejecutadas en JVM son bastante **dinámicas**
- Pueden **adaptarse muy bien al entorno** y cargar las librerías necesarias
- Cuando se empaquetan en native image son **específicas para un entorno concreto**, eso reduce el tamaño del ejecutable
- Ciertas opciones de Spring Boot no están disponibles en native images

https://docs.spring.io/spring-boot/docs/current/reference/html/native-image.html

Empaquetado en native image

code URJC

- **Instalación de GraalVM de Bellsoft (recomendado por Spring)**
  - Instalación de SDKMan en linux y mac

```
$ curl -s "https://get.sdkman.io" | bash
```

  - Instalación de Liberica GraalVM con SDKMan

```
$ sdk install java 22.3.r17-nik
$ sdk use java 22.3.r17-nik
```

https://docs.spring.io/spring-boot/docs/current/reference/html/native-image.html#native-image.developing-your-first-application.native-build-tools.prerequisites

Empaquetado en native image

- Instalación de herramientas de compilación nativa en linux

```
$ sudo apt-get install build-essential libz-dev zlib1g-dev
```

https://www.graalvm.org/22.1/reference-manual/native-image/#prerequisites

Empaquetado en native image

code URJC

ejem2

- Al generar el esqueleto de la aplicación SpringBoot hay que añadir la dependencia "**GraalVM Native Support**"
- Esto configura el plugin de SpringBoot para soporte de imágenes nativas

https://start.spring.io/

Empaquetado en native image

code URJC

ejem2

- **Empaquetado de app SpringBoot**
  - Ejecuta el comando maven

```
$ mvn -Pnative native:compile
```

  - Y esperar varios minutos…

https://docs.spring.io/spring-boot/docs/current/reference/html/native-image.html#native-image.developing-your-first-application.native-build-tools

Empaquetado en native image

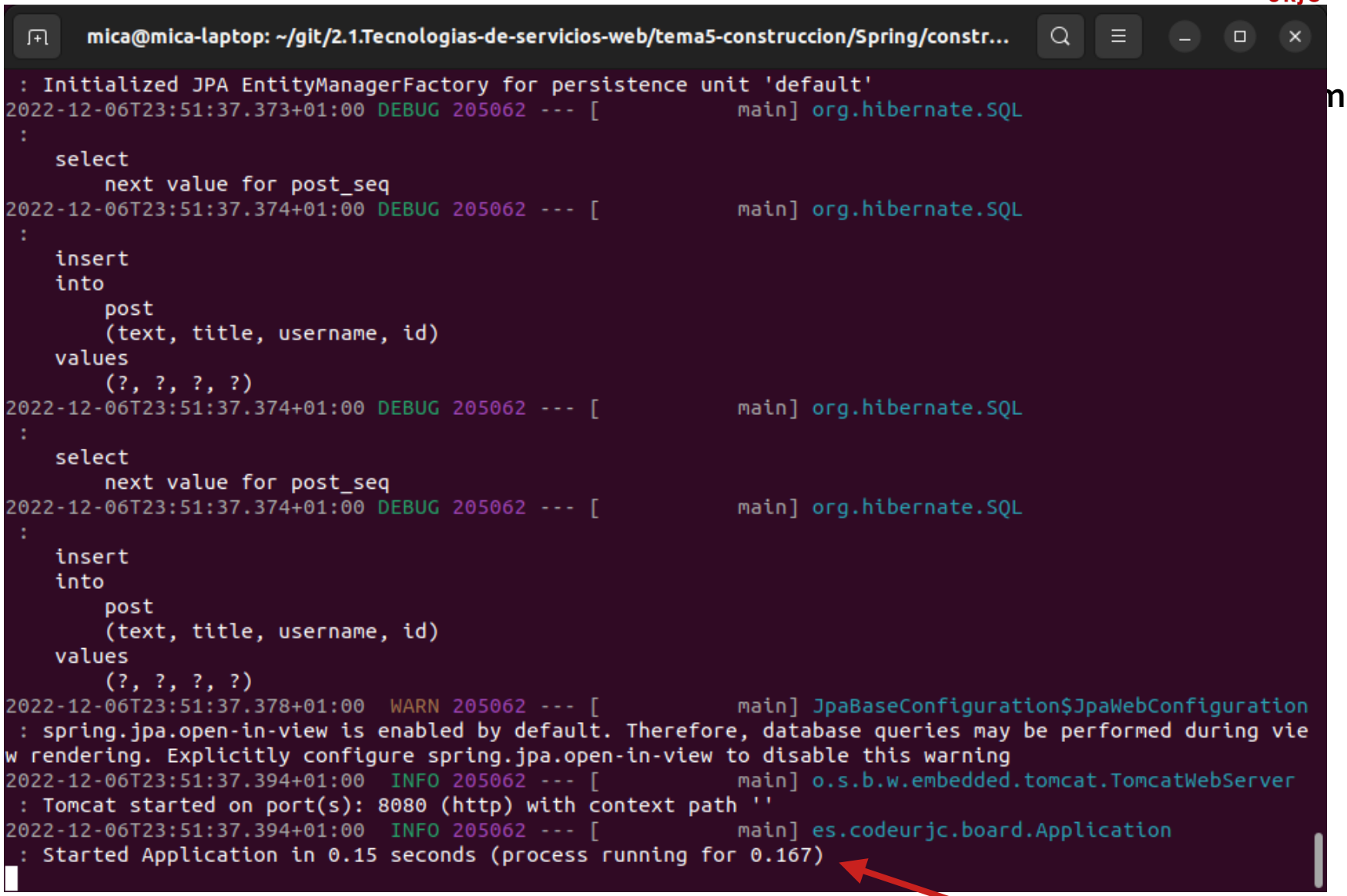
code URJC

ejem2

- **Ejecución de app SpringBoot**
  - Un binario nativo con el nombre del proyecto Maven se genera en la carpeta **target**
  - Se ejecuta con

```
$ ./target/construccion_ejem2
```

m2


mica@mica-laptop: ~/git/2.1.Tecnologias-de-servicios-web/tema5-construccion/Spring/constr...
: Initialized JPA EntityManagerFactory for persistence unit 'default'
2022-12-06T23:51:37.373+01:00 DEBUG 205062 --- [main] org.hibernate.SQL
:
select
next value for post_seq
2022-12-06T23:51:37.374+01:00 DEBUG 205062 --- [main] org.hibernate.SQL
:
insert
into
post
(text, title, username, id)
values
(?, ?, ?, ?)
2022-12-06T23:51:37.374+01:00 DEBUG 205062 --- [main] org.hibernate.SQL
:
select
next value for post_seq
2022-12-06T23:51:37.374+01:00 DEBUG 205062 --- [main] org.hibernate.SQL
:
insert
into
post
(text, title, username, id)
values
(?, ?, ?, ?)
2022-12-06T23:51:37.378+01:00 WARN 205062 --- [main] JpaBaseConfiguration$JpaWebConfiguration
: spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during vie
w rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2022-12-06T23:51:37.394+01:00 INFO 205062 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer
: Tomcat started on port(s): 8080 (http) with context path ''
2022-12-06T23:51:37.394+01:00 INFO 205062 --- [main] es.codeurjc.board.Application
: Started Application in 0.15 seconds (process running for 0.167)