code
URJC

Ampliación de Ingeniería del Software

Anexos

Universidad Rey Juan Carlos

Micael Gallego
Correo: micael.gallego@urjc.es
Twitter: @micael_gallego

Francisco Gortázar
Correo: francisco.gortazar@urjc.es
Twitter: @fgortazar

Michel Maes
michel.maes@urjc.es
Óscar Soto
oscar.soto@urjc.es

code
URJC

Micael Gallego, Francisco Gortázar, Michel Maes, Óscar Soto

Anexos

Anexos proporcionados en la asignatura

- Los siguientes Anexos tratan de completar el conocimiento de los alumnos con tecnologías y servicios:
  - Anexo: Uso de Maven por terminal
  - Anexo: Introducción a Docker
  - Anexo: Despliegue de aplicaciones en Heroku

code URJC

Ampliación de Ingeniería del Software

Anexo – Uso de Maven por terminal

Universidad Rey Juan Carlos

Micael Gallego
Correo: micael.gallego@urjc.es
Twitter: @micael_gallego

Francisco Gortázar
Correo: francisco.gortazar@urjc.es
Twitter: @fgortazar

Michel Maes
michel.maes@urjc.es

Óscar Soto
oscar.soto@urjc.es

code
URJC

Micael Gallego, Francisco Gortázar, Michel Maes, Óscar Soto

Maven

code URJC

¿Qué es Maven?

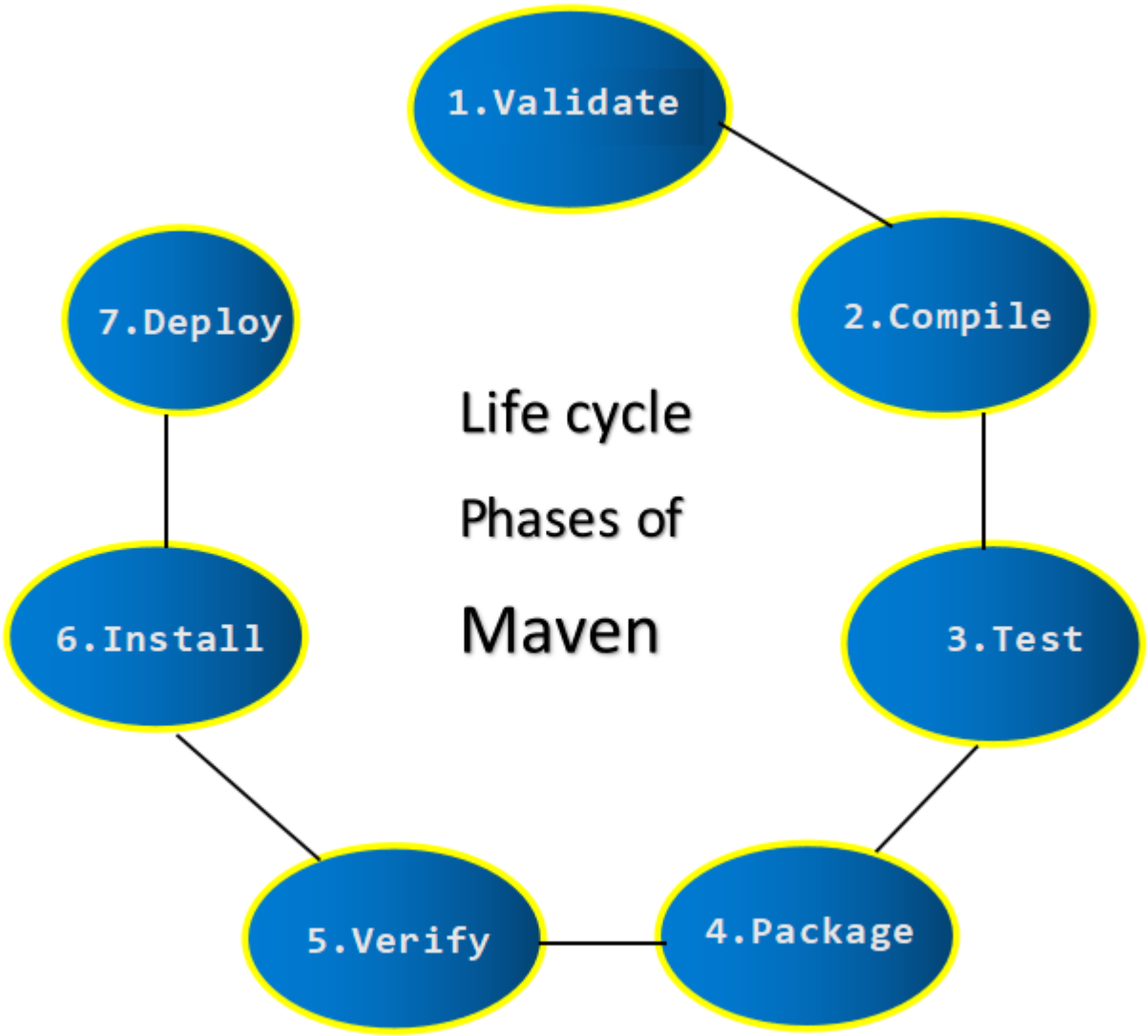
- Sistema de **gestión de dependencias (librerías)** con descarga automática
- Sistema de **construcción de proyectos** (de código a paquete entregable y ejecutable)
- Estructura única de proyecto compatible con todos los **entornos de desarrollo** y sistemas de **CI**

Maven

¿Qué es Maven?

- Maven gestiona muchos aspectos de la **construcción** de la aplicación
  - Descarga automática de dependencias
  - Compilación
  - Ejecución de tests
  - Publicación de la aplicación construida (binarios)
  - Documentación

Maven


1.Validate
2.Compile
3.Test
4.Package
5.Verify
6.Install
7.Deploy
Life cycle
Phases of
Maven

Maven

code URJC

Life-cycle de Maven

- Es un proceso de construcción y distribución de artefactos software (proyectos)
- Cuenta con una serie de pasos secuenciales
- Si uno de los pasos falla, no se continúa con el siguiente
- Maven cuenta con diferentes plugins que permiten llevar a cabo estas pasos de manera automática
- Podemos ejecutar solo hasta cierta fase (p.e. Maven Test)

Maven

Life-cycle de Maven

- **Validate:** Comprueba que el proyecto es correcto y tiene toda la información necesaria
- **Compile:** Compila el código fuente de la aplicación
- **Test:** Compila el código de los test y los ejecuta
- **Package:** Empaqueta el código fuente en un formato distribuible (p.e. JAR)
- **Verify:**Comprueba checks de la aplicación
- **Install:** Instala el paquete en el repositorio local (.m2)
- **Deploy:** Copia el paquete final a un repositorio remoto (p.e. Maven Central)

Maven

¿Cómo se usa Maven?

- **Desde el entorno de desarrollo:** Maven está integrado en los IDEs Java más importantes (Eclipse, IntelliJ, Visual Studio Code…)
  - Instalado por defecto junto al IDE
- **Desde la línea de comandos:** Sin necesidad de usar un IDE. Ideal para construcción de proyectos de forma automatizada
  - Debe instalarse en la máquina

Maven

Maven desde línea de comandos

- Es necesario instalar el binario en nuestra máquina o en la máquina de CI
- Para **Java 17** es necesario instalar **Maven 3.8**
- Descarga e instalación
  - Oficial: https://maven.apache.org/download.cgi
  - Windows (Guía): https://mkyong.com/maven/how-to-install-maven-in-windows/
  - Linux (Guía): https://computingforgeeks.com/install-latest-apache-maven-on-ubuntu/

Maven

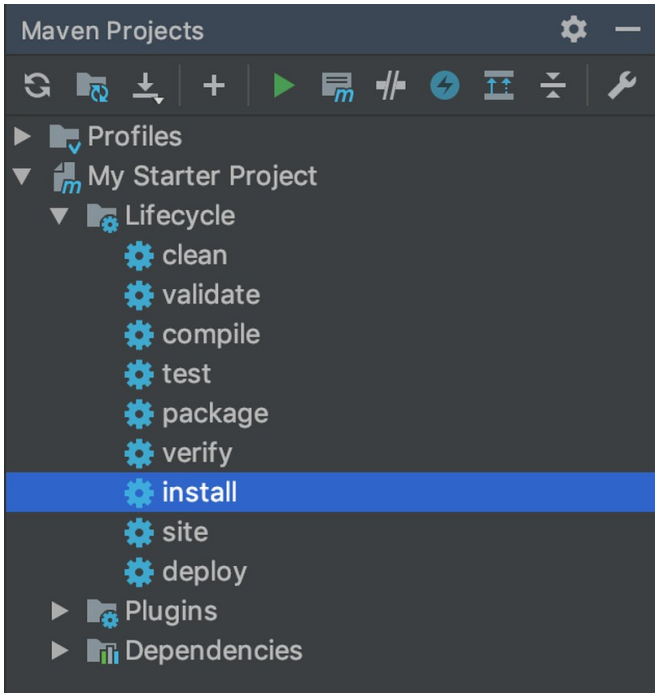
Maven desde línea de comandos

- Una vez instalado, podemos comprobar su versión

Maven

Maven desde línea de comandos

- Desde la herramienta de comandos podemos ejecutar todas las acciones que realizamos de manera interactiva desde el IDE
  - mvn clean
  - mvn validate
  - mvn compile
  - mvn test
  - mvn package
  - ...

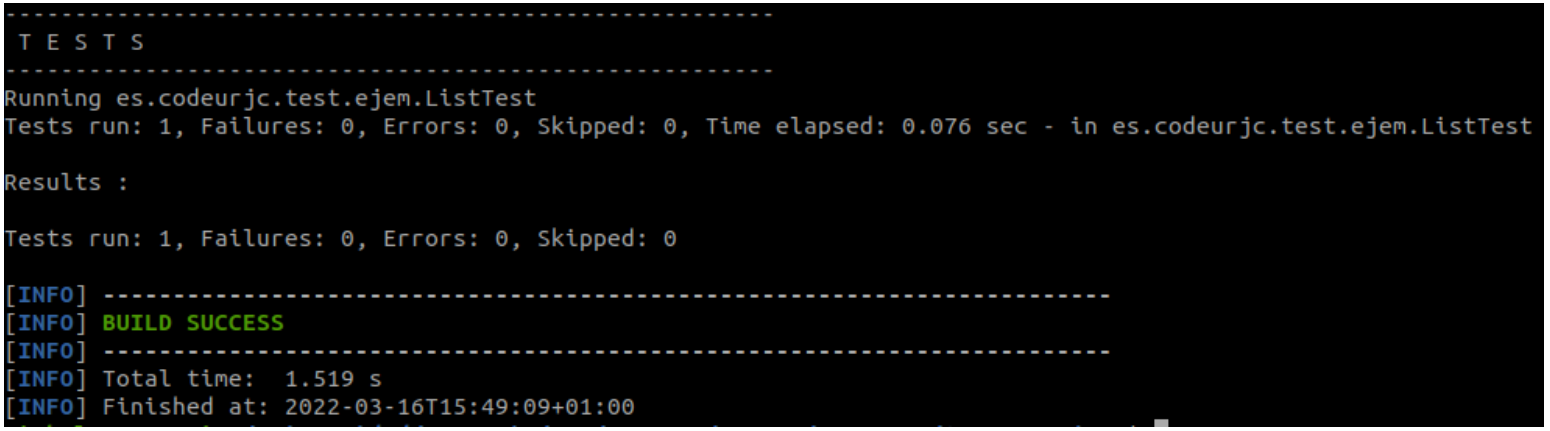


Maven

mvn test

- Ejecuta todos los test del framework de testing que hayamos configurado (p.e. JUnit)

mvn test



tema1_2_ejem0

Maven

mvn test

- Es posible filtrar los tests y ejecutar únicamente una clase o método

```
mvn test -Dtest=ClassName#Method
```

```
mvn test -Dtest=AppTest
```

```
mvn test -Dtest=Calculadora3Test#testResta
```

Maven

mvn package

- Empaqueta el proyecto para su distribución
- Es común generar un JAR (en la carpeta target)
  - Incluye la aplicación con todas sus dependencias
  - No incluye los test
  - Para aplicaciones web o interactivas, puede ser ejecutado directamente teniendo Java instalado
- Requiere de configuración adicional en el pom.xml en función del proyecto

Maven

mvn package con SpringBoot

- Usaremos el proyecto **selenium_ejem2**
- SpringBoot cuenta con un plugin para generar un JAR ejecutable de manera sencilla

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

Maven

mvn package con SpringBoot

- Usaremos el proyecto **selenium_ejem2**
- Construimos el JAR ejecutable
  - mvn package
- Ejecutamos el JAR como una aplicación
  - java -jar target/selenium_ejem2-0.0.1-SNAPSHOT.jar

TIP: En caso de no querer ejecutar los test, es posible “ignorarlos” con **mvn package -DskipTests**

code
URJC

Ampliación de Ingeniería del Software

Anexo – Introducción a Docker

Universidad Rey Juan Carlos

Micael Gallego
Correo: micael.gallego@urjc.es
Twitter: @micael_gallego

Francisco Gortázar
Correo: francisco.gortazar@urjc.es
Twitter: @fgortazar

Michel Maes
michel.maes@urjc.es
Óscar Soto
oscar.soto@urjc.es

code
URJC

Micael Gallego, Francisco Gortázar, Michel Maes, Óscar Soto

Introducción a Docker

code URJC

¿Qué es Docker?

- Es una nueva forma de **ejecutar** y **distribuir** aplicaciones
- Las aplicaciones se empaquetan con **todas sus dependencias**
- Se pueden ejecutar en **cualquier entorno** (linux, windows, mac)
- Sólo es necesario tener instalado **Docker**
- Una aplicación Docker en ejecución se llama **contenedor**
- Un contenedor se crea desde una **imagen** (plantilla)

Introducción a Docker

code URJC

¿Qué son las Imágenes Docker?

- Paquete que permite ejecutar aplicaciones en contenedores
- Contiene las herramientas del SO (**ubuntu, alpine**), runtime (**Java**) y la aplicación en sí (**webapp.jar**)
- Un contenedor siempre se inicia desde una **imagen**
- Si se quiere arrancar un contenedor partiendo de una imagen que no está disponible, se **descarga automáticamente** de Internet

Introducción a Docker

code URJC

¿Cómo instalo Docker en mi máquina?

- Ubuntu:
  - https://store.docker.com/editions/community/docker-ce-server-ubuntu
- Mac:
  - https://store.docker.com/editions/community/docker-ce-desktop-mac
- Windows:
  - https://store.docker.com/editions/community/docker-ce-desktop-windows

Introducción a Docker

Ejecución de una aplicación dockerizada

- Ejecución de "hello-world" en un contenedor

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
03f4658f8b78: Pull complete
a3ed95caeb02: Pull complete
Digest:
sha256:8be990ef2aeb16dbcb9271ddfe2610fa6658d13f6dfb8bc72074cc1ca369
66a7
Status: Downloaded newer image for hello-world:latest
Hello from Docker.
This message shows that your installation appears to be working
correctly.
...
```

Introducción a Docker

Ejecución de una aplicación dockerizada

- Ejecución de "hello-world" en un contenedor

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
03f4658f8b78: Pull complete
a3ed95caeb02: Pull complete
Digest:
sha256:8be990ef2aeb16dbcb9271ddfe2610fa6658d13f6dfb8bc72074cc1ca369
66a7
Status: Downloaded newer image for hello-world:latest
Hello from Docker.
This message shows that your installation appears to be working
correctly.
...
```

La primera vez la imagen se descarga

Introducción a Docker

Ejecución de una aplicación dockerizada

- Ejecución de "hello-world" en un contenedor

```
$ docker run hello-world
Hello from Docker.
This message shows that your installation appears to be working
correctly.
...
```

La segunda vez se usa la imagen descargada (no se vuelve a descargar)

Introducción a Docker

Ejecutar una aplicación Java en un contenedor

- Descargamos o clonamos la aplicación de ejemplo
  - https://github.com/URJC-AIS/docker-example
- Spring permite construir imágenes Docker de manera sencilla con el siguiente comando

```
$ mvn spring-boot:build-image
```

```
<groupId>es.codeurjc.rest</groupId>
<artifactId>items</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

Genera la imagen **items:0.0.1-SNAPSHOT**

Introducción a Docker

code URJC

Ejecutar una aplicación Java en un contenedor

- Podemos definir en nombre de la imagen en el comando

```
$ mvn spring-boot:build-image -Dspring-boot.build-image.imageName=items:v1
```

- También podemos definir en nombre de la imagen en el pom.xml

```
<plugin>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-maven-plugin</artifactId>
   <configuration>
       <image>
       <name>items:${project.version}</name>
       </image>
   </configuration>
</plugin>
```

Introducción a Docker

Ejecutar una aplicación Java en un contenedor

- Podemos ver las imágenes disponibles en nuestra máquina

$ docker images

```
michel@maes-msi:~/Universidad/Docencia/AIS/2021-22/docker-example$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
items        v1        ceca272c2b7d   42 years ago    273MB
```

Introducción a Docker

code URJC

Ejecutar una aplicación Java en un contenedor

- Podemos lanzar la aplicación como un contenedor partiendo de la imagen que acabamos de crear

```
docker run -p 8080:8080 items:0.0.1-SNAPSHOT
```

Introducción a Docker

code URJC

Ejecutar una aplicación Java en un contenedor

- Podemos lanzar la aplicación como un contenedor partiendo de la imagen que acabamos de crear

```
docker run -p 8080:8080 items:0.0.1-SNAPSHOT
```

Nombre de la imagen que hemos creado

Introducción a Docker

code URJC

Ejecutar una aplicación Java en un contenedor

- Podemos lanzar la aplicación como un contenedor partiendo de la imagen que acabamos de crear

```
docker run -p 8080:8080 items:0.0.1-SNAPSHOT
```

Puerto de la aplicación que queremos que sea accesible desde el navegador.

Se indica el puerto que usa la aplicación dentro de Docker y el puerto de nuestra máquina.

Introducción a Docker

code URJC

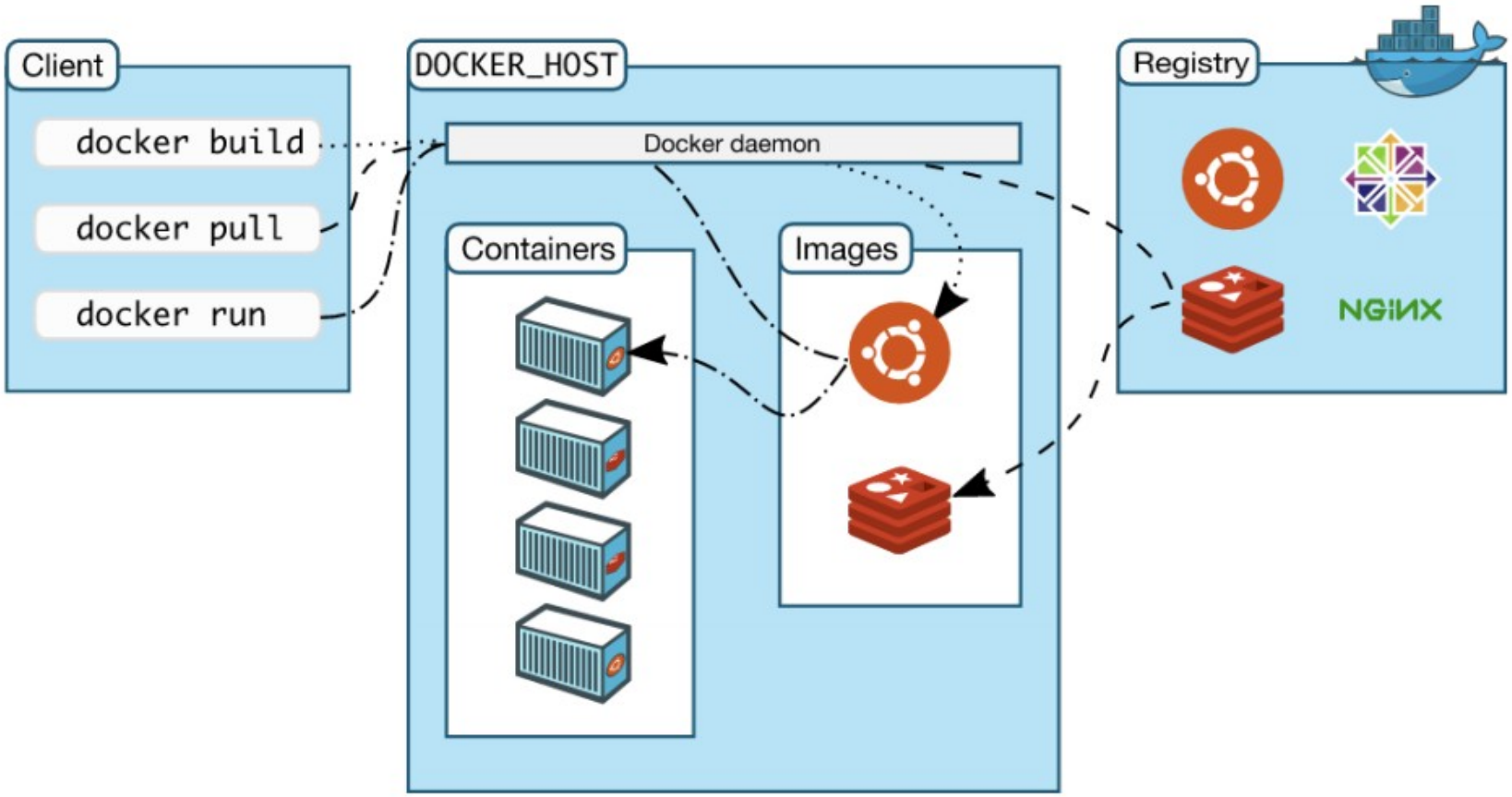
¿Qué es un Docker Registry?

- **Servicio remoto** para **subir y descargar** imágenes
- Puede guardar varias "versiones" (tags) de la misma imagen
- Las diferentes versiones de una misma imagen se almacenan en un repositorio (mysql, java, ubuntu ...)
- **DockerHub** es un registro público y gratuito gestionado por Docker Inc.
- Puedes instalar un registro privado

Introducción a Docker

code URJC

¿Qué es un Docker Registry?



Introducción a Docker

¿Como subo mi imagen a un Docker Registry?

- Utilizaremos el registry por defecto, DockerHub
- Utilizaremos la imagen que hemos creado anteriormente (items:v1)
- Debemos crear una cuenta gratuita en DockerHub
  - https://hub.docker.com/
- Por defecto, la imagen que subamos será pública

https://docs.docker.com/docker-hub/repos/

Introducción a Docker

code URJC

¿Como subo mi imagen a un Docker Registry?

- Necesitamos que la imagen cumpla el siguiente formato:

  <hub-user>/<repo-name>:<tag>

- Si ya tenemos una imagen creada, podemos renombrarla:

  docker tag items:v1 <hub-user>/items:v1

- También podemos crear una nueva imagen con este formato

```
$ mvn spring-boot:build-image
        -Dspring-boot.build-image.imageName=<hub-user>/items:v1
```

Introducción a Docker

¿Como subo mi imagen a un Docker Registry?

- Realizamos el login en DockerHub por terminal

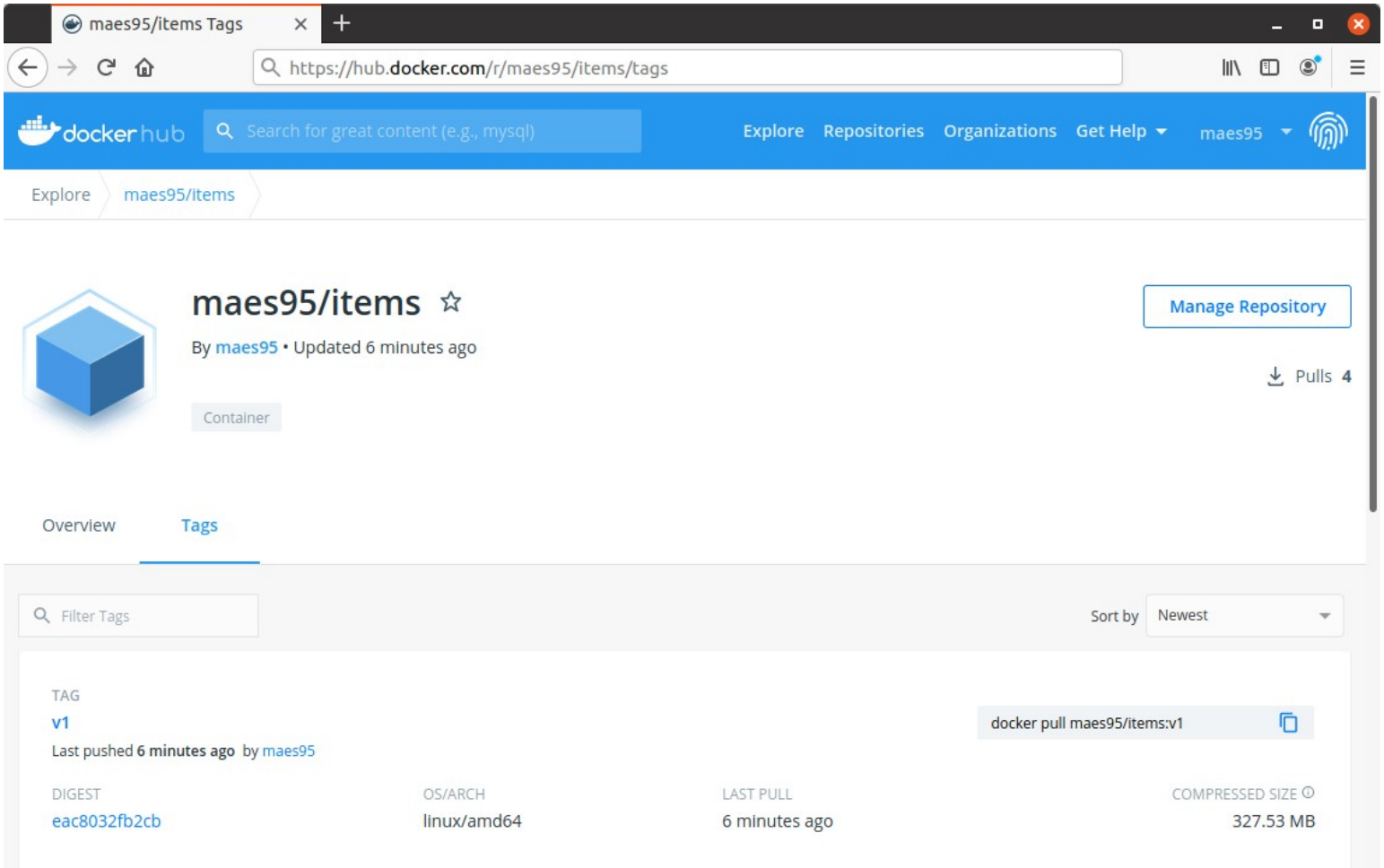
```
docker login
```

- Subimos la imagen a Docker Hub

```
docker push <hub-user>/items:v1 .
```

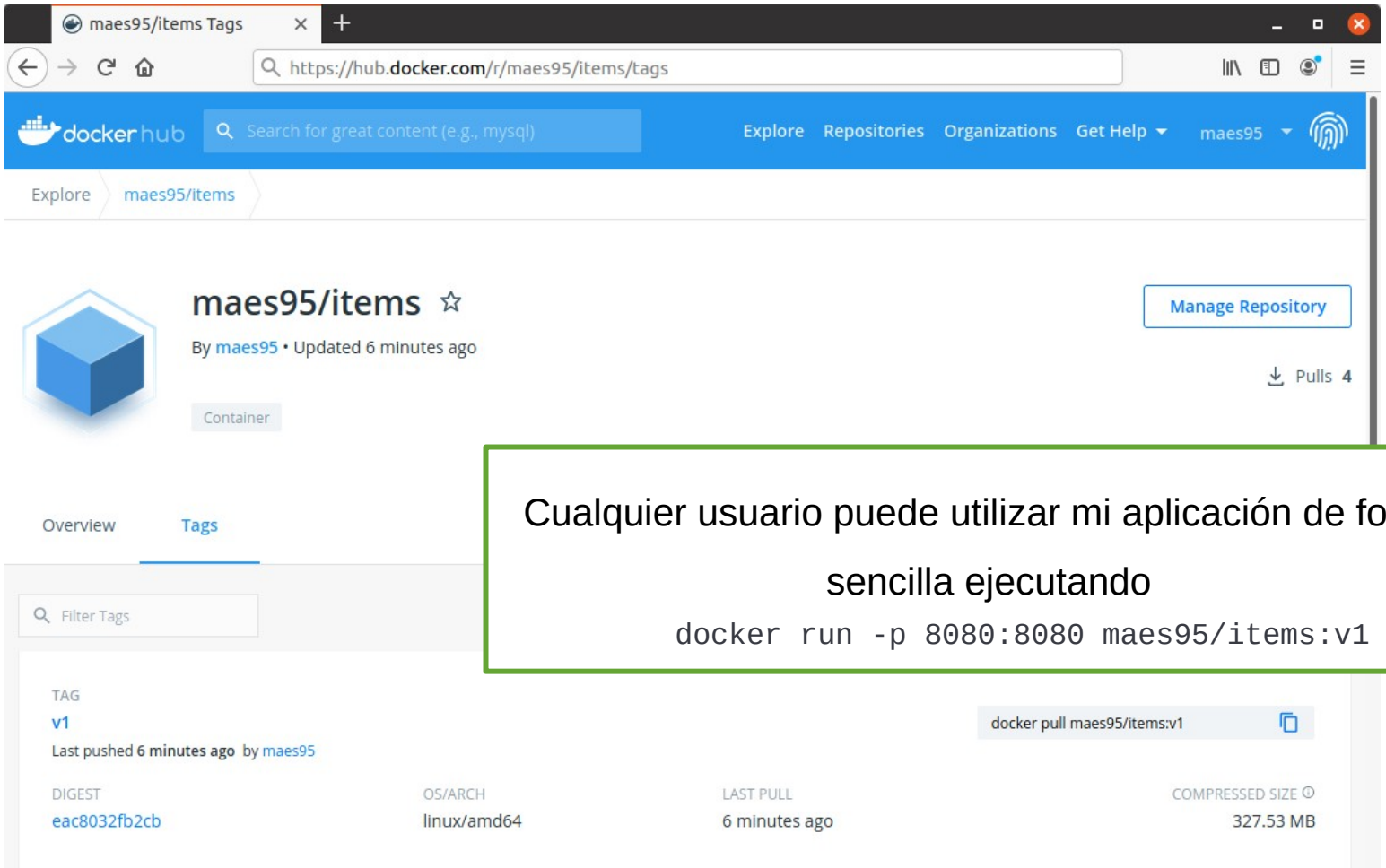
Introducción a Docker

¿Como subo mi imagen a un Docker Registry?



Introducción a Docker

¿Como subo mi imagen a un Docker Registry?



Cualquier usuario puede utilizar mi aplicación de forma sencilla ejecutando

```
docker run -p 8080:8080 maes95/items:v1
```

Introducción a Docker

code URJC

Otros comandos de Docker

- Lanzar un contenedor en segundo plano con un nombre

```
docker run -d -p 8080:8080 –name items-app items:v1
```

- Borrar un contenedor lanzado

```
docker rm -f items-app
```

- Borrar una imagen

```
docker rmi items:v1
```

code URJC

Ampliación de Ingeniería del Software

Anexo – Despliegue de aplicaciones en Heroku

Universidad Rey Juan Carlos

Micael Gallego
Correo: micael.gallego@urjc.es
Twitter: @micael_gallego

Francisco Gortázar
Correo: francisco.gortazar@urjc.es
Twitter: @fgortazar

Michel Maes
michel.maes@urjc.es

Óscar Soto
oscar.soto@urjc.es

code
URJC

Micael Gallego, Francisco Gortázar, Michel Maes, Óscar Soto

Introducción a Heroku

code URJC

¿Qué es Heroku?

- Una plataforma como servicio (PaaS)
- Nos permite **desplegar aplicaciones** en la nube de forma sencilla
  - Podemos lanzar las aplicaciones directamente desde terminal
  - Buena integración con sistemas de CI/CD
- No se administra ningún servidor
- Utiliza Docker para simplificar el despliegue

Introducción a Heroku

¿Qué necesito para utilizar Heroku?

- Nos registramos de forma gratuita
  - https://www.heroku.com/
- Descargamos el cliente para terminal
  - https://devcenter.heroku.com/articles/heroku-cli
- Descargamos o clonamos la aplicación de ejemplo
  - https://github.com/URJC-AIS/continuous-deployment-heroku

Introducción a Heroku

Pasos para desplegar en Heroku

- 0) Configurar mi aplicación para el despliegue
- 1) Hacer login en Heroku
- 2) Crear una aplicación en Heroku
- 3) Crear la imagen Docker de la aplicación
- 4) Publicar la imagen Docker en el registro de Heroku
- 5) Desplegar la imagen Docker en la aplicación Heroku

Introducción a Heroku

code URJC

0) Configurar mi aplicación para el despliegue

- La aplicación tiene que usar el puerto indicado por Heroku (no en 8080)
- Heroku proporcionará a la aplicación la variable de entorno **$PORT**, la cual podemos recoger en la configuración de la aplicación Spring. Podemos definir un puerto por defecto para el desarrollo en local.

application.properties

```
server.port=${PORT:8080}
```

Introducción a Heroku

code URJC

0) Configurar mi aplicación para el despliegue

- Las aplicaciones de Heroku gratuitas están limitadas a **512Mb** de RAM
- Por defecto una aplicación SpringBoot en un contenedor Docker necesita más de 512Mb
- Configuramos la memoria adecuada en el pom.xml al crear la imagen Docker

Introducción a Heroku

0) Configurar mi aplicación para el despliegue

- Debemos cambiar la configuración en el pom.xml

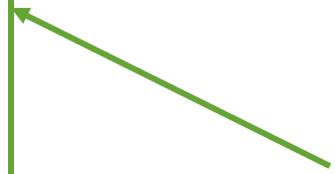
```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
        <image>
            <env>
                <BPE_OVERRIDE_JAVA_TOOL_OPTIONS>
                    -Xss256K
                    -XX:ReservedCodeCacheSize=64M
                    -XX:MaxMetaspaceSize=100000K
                    -Xmx64M
                </BPE_OVERRIDE_JAVA_TOOL_OPTIONS>
            </env>
        </image>
    </configuration>
</plugin>
```

Introducción a Heroku

0) Configurar mi aplicación para el despliegue

- Debemos cambiar la configuración en el pom.xml

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
        <image>
            <env>
                <BPE_OVERRIDE_JAVA_TOOL_OPTIONS>
                    -Xss256K
                    -XX:ReservedCodeCacheSize=64M
                    -XX:MaxMetaspaceSize=100000K
                    -Xmx64M
                </BPE_OVERRIDE_JAVA_TOOL_OPTIONS>
            </env>
        </image>
    </configuration>
</plugin>
```



Introducción a Heroku

code URJC

1) Hacer login en Heroku

```
$ heroku login
```

- Nos loguea a través del navegador

2) Crear una aplicación en Heroku

```
$ heroku create <app-name>
```

- El <app-name> debe ser único para todo Heroku
- Utilizaremos el siguiente formato: **ais-<usuario_urjc>**

Introducción a Heroku

code URJC

3) Crear la imagen de la aplicación

- El nombre de las imágenes Docker sigue el siguiente formato

<registry>/<user>/<image-name>:<version>

- Las imágenes que hemos creado ahora en realidad tienen el registry de DockerHub por defecto

docker.io/maes95/items:v1

- Para subir una imagen al registry de Heroku es necesario seguir un formato específico:
  - El registro es registry.heroku.com
  - El usuario es `<app-name>` (el nombre de la aplicación)
  - El nombre de la imagen **debe** ser web
  - No es necesario especificar la versión (usará latest)

Introducción a Heroku

code URJC

3) Crear la imagen de la aplicación

- Clonamos el proyecto y nos movemos a la raíz del proyecto

```
$ git clone https://github.com/URJC-AIS/continuous-deployment-heroku
```

- Construimos la imagen Docker siguiendo el formato definido por Heroku

```
$ mvn spring-boot:build-image \
    -Dspring-boot.build-image.imageName=registry.heroku.com/<app-name>/web
```

Introducción a Heroku

code URJC

4) Publicar la imagen Docker en el registro de Heroku

- Se configura docker para publicar en el registro de imágenes de Heroku (solo la primera vez)

```
$ heroku container:login
```

- Subimos la imagen Docker

```
$ docker push registry.heroku.com/<app-name>/web
```

Introducción a Heroku

code URJC

5) Desplegar la imagen Docker en la aplicación Heroku

- Con el siguiente comando, se lanza un contenedor a partir de la imagen (**web**) subida a la aplicación **<app-name>**

```
$ heroku container:release web --app <app-name>
```

- Nuestra aplicación estará disponible en la siguiente url

```
https://<app-name>.herokuapp.com/
```

Introducción a Heroku

5) Desplegar la imagen Docker en la aplicación Heroku

Introducción a Heroku

Monitorización

- Podemos ver los logs de la aplicación desde la terminal

```
$ heroku logs --tail --app <app-name>
```