

# Práctica 1: Primera aplicación con Spring Boot

En esta práctica tendremos un primer contacto con Spring Boot, Git y Docker.

Los objetivos principales son:

- Empezar a conocer Spring Boot, ejecutando una sencilla aplicación *hola mundo* en Spring Boot.
- Empezar a conocer el *framework* de plantillas Thymeleaf, realizando pequeñas modificaciones en la aplicación que usen un formulario.
- Trabajar de forma regular, realizando pequeños commits que se deben subir al repositorio personal de la asignatura en GitHub.
- Crear una aplicación desplegable usando Docker
- Desplegar la aplicación en el servidor de la asignatura.

## 1. Instalación de software

Vamos a trabajar bastante con el terminal. En Linux o macOS podemos usar el terminal que viene con el sistema. En Windows se puede usar el terminal Git Bash que se instala en la instalación de Git para Windows.

Es posible desarrollar la práctica en cualquier sistema operativo. Debemos instalar el siguiente software:

- Git
- Java JDK 8 o posterior
- IntelliJ Ultimate



### Nota del profesor sobre en el sistema operativo en el que realizar la práctica

Aunque en los apuntes aparezca información sobre cómo trabajar desde Windows, no puedo garantizar que las instrucciones funcionen correctamente en todas las posibles configuraciones, ni te podré ayudar con posibles problemas, porque no es un sistema operativo que maneje habitualmente. Por tanto, si tienes Windows, te recomiendo que instales una máquina virtual Linux y la uses para la práctica.

Recomendamos hacer el desarrollo usando el IDE [IntelliJ Ultimate](#). Aunque es de pago, es posible [obtener una licencia de estudiante](#) usando la dirección de correo de la UA.

## Instalación básica

### Linux

Para instalar el software en Linux.

- Instalar Git y Java:

```
$ sudo apt install git  
$ sudo apt install default-jdk
```

- Instalar [IntelliJ Ultimate](#)

### macOS

- Git y Java vienen instalados con el sistema operativo.
- Instalar [IntelliJ Ultimate](#)

### Windows

Es recomendable instalar [git for Windows](#), que además de Git instala Git BASH, un terminal Bash integrado en Windows.

Además, hay que instalar Java e [IntelliJ Ultimate](#).

## Después de la instalación básica

Es fácil probar que funciona el software instalado. Basta con ejecutar desde el terminal:

```
$ git --version  
$ java -version (imprime la versión de Java)
```

## Configuración del prompt para que aparezca la rama de Git

### Bash

Es también bastante útil configurar el prompt para que aparezca la rama del repositorio Git en que nos encontramos. Para ello se debe añadir en el fichero `$HOME/.bashrc` (linux y Git Bash Windows) o `$HOME/.bash_profile` (macOS con shell bash):

```
parse_git_branch() {  
    git branch 2> /dev/null | sed -e '/^[\*]/d' -e 's/* \(.*/)/ (\1)/'  
}  
export PS1="\[\e[37m\]\A \[\e[m\]\[\e[33;32m\]\W\[\e[33;31m\]\$(parse_git_branch)\[\e[33;00m\] $ "
```

Podemos encontrar más opciones de configuración del prompt en muchas páginas en Internet. Por ejemplo [aquí](#).

## Zsh

Si trabajas con el shell `zsh` que viene por defecto en MacOS, debes añadir en el fichero `.zshrc` lo siguiente:

```
parse_git_branch() {
    git branch 2> /dev/null | sed -n -e 's/^.* \(.*\)/ [\1]/p'
}
setopt PROMPT_SUBST
export PROMPT='%1~%F{green}$(parse_git_branch)%f %% '
```

## 2. Creación del repositorio GitHub con la práctica

Para inicializar el repositorio de GitHub en el que vas a trabajar en esta práctica debes seguir los siguientes pasos:

1. Inicializa tu nombre de usuario y tu correo en Git. El nombre de usuario será el nombre que aparecerá en los *commits*. Pon tu nombre y apellido.

```
$ git config --global user.name "Pepe Perez"
$ git config --global user.email pepe.perez@example.com
```

2. Crea una cuenta en GitHub. Puedes usar el nombre de usuario que quieras (o usar el que ya tienes), pero **escribe correctamente tu nombre y apellidos en el perfil** usando la opción *Settings > Profile* y actualizando el campo *Name*.
3. Una vez logeado en GitHub, pincha en el enlace con una invitación que compartiremos en el foro de Moodle. Deberás aceptar las peticiones de GitHub Classroom y podrás aceptar la práctica *Spring Boot Demo App*.

MADS 2020-21

**Accept the assignment —**

**SpringBoot Demo App**

Once you accept this assignment, you will be granted access to the `springboot-demo-app-domingogallardo2` repository in the `mads-ua-20-21` organization on GitHub.

**Accept this assignment**

Se creará automáticamente el repositorio `springboot-demo-app-<usuario>` en la organización `mads-ua-22-23`. Es un repositorio privado al que tienes acceso tú y el

profesor. Contiene el código inicial del proyecto demostración de Spring Boot (es una copia del repositorio [domingogallardo/spring-boot-demoapp](https://github.com/domingogallardo/spring-boot-demoapp)).

The screenshot shows a GitHub repository page. At the top, it displays the repository name 'mads-ua-20-21/springboot-demo-app-domingogallardo2' (Private), with options to 'Watch' (1), 'Star' (0), and 'Fork' (0). Below this are tabs for 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', and 'Insights'. The 'Code' tab is selected, showing a list of files: '.mvn/wrapper' (Initial commit, 13 minutes ago), 'src' (Initial commit, 13 minutes ago), '.gitignore' (Initial commit, 13 minutes ago), 'README.md' (Initial commit, 13 minutes ago), 'mvnw' (Initial commit, 13 minutes ago), 'mvnw.cmd' (Initial commit, 13 minutes ago), and 'pom.xml' (Initial commit, 13 minutes ago). To the right, there's an 'About' section with details about the repository, including its creation by 'GitHub Classroom'. It also lists 'Readme', 'Releases' (none published), 'Packages' (none published), and 'Languages' (Java 98.4%, HTML 1.6%). The README.md content is titled 'Aplicación inicial Spring Boot' and describes it as an 'Aplicación básica usando Spring Boot y plantillas Thymeleaf.'

Es importante que tengas en cuenta que el repositorio recién creado no reside en tu cuenta, sino en la organización `mads-ua-22-23`. Puedes acceder a él desde el *dashboard* de GitHub que aparece cuando te logeas o pulsando en el icono de GitHub:

The screenshot shows the GitHub dashboard for the user 'domingogallardo2'. At the top, there's a search bar and navigation links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below this, the user's profile picture and name 'domingogallardo2' are displayed. The 'Repositories' section shows a list with one item: 'mads-ua-20-21/springboot-demo-app-domingogallardo2'. A red arrow points to this repository link. To the right, there's a 'Discover interesting projects and people to populate your personal news feed.' section with a 'Explore GitHub' button. Below it, there's a 'ProTip!' message and a link to 'Subscribe to your news feed'.

4. El profesor te invitará a formar parte de la organización `mads-ua-22-23` y recibirás un mensaje de correo electrónico en el que deberás aceptar esta invitación. También se puede aceptar la invitación accediendo a <https://github.com/mads-ua-22-23>.

### 3. Aplicación Demo de Spring Boot

Haremos una primera práctica sencilla en la que primero pondremos en marcha y publicaremos una aplicación inicial en Spring Boot y después añadiremos alguna funcionalidad.

En el documento [Introducción a Spring Boot](#) se explica cómo ejecutar una aplicación Spring Boot y cómo lanzar sus tests. También se proporciona una introducción a los distintos

componentes de la aplicación. Debes leerlo y aprender el funcionamiento básico de este *framework*.

## Construcción y ejecución de la aplicación

Lo primero que deberás hacer será descargar la aplicación `demo-spring-boot` que tienes en el repositorio creado en el punto anterior y comprobar que funciona correctamente. Debes hacer lo siguiente:

1. Configura un **Personal Access Token** (PAT) en GitHub para poder autenticarte desde el terminal. Dale todos los permisos de acceso a repositorios y copia la clave generada. Será la contraseña que deberás introducir cuando un comando git te la pida.
2. Descarga en tu ordenador el repositorio creado en GitHub en el apartado anterior, usando el comando `git clone`:

```
$ git clone https://github.com/mads-ua-22-23/springboot-demo-app-<usuario>.git
```

Cuando git te pida autenticación, usa como nombre de usuario tu usuario de GitHub y como contraseña el PAT que has creado anteriormente.

Una vez descargado el repositorio con la aplicación deberás ejecutarla desde la línea de comandos, probar los tests e importarla en IntelliJ y ejecutar y depurar con el IDE la aplicación y los tests.

3. Desde el directorio donde está la aplicación, probamos todos sus tests usando el Maven Wrapper:

```
$ ./mvnw test
```

4. Para poner en marcha la aplicación la arrancamos como una aplicación Java:

```
$ ./mvnw package  
$ java -jar target/demoapp-0.0.1-SNAPSHOT.jar
```

También podemos lanzarla usando el plugin `spring-boot` de Maven:

```
$ ./mvnw spring-boot:run
```

La aplicación se arranca por defecto en el puerto local 8080. Una vez arrancada la aplicación podemos conectarnos desde un navegador a sus páginas de inicio.

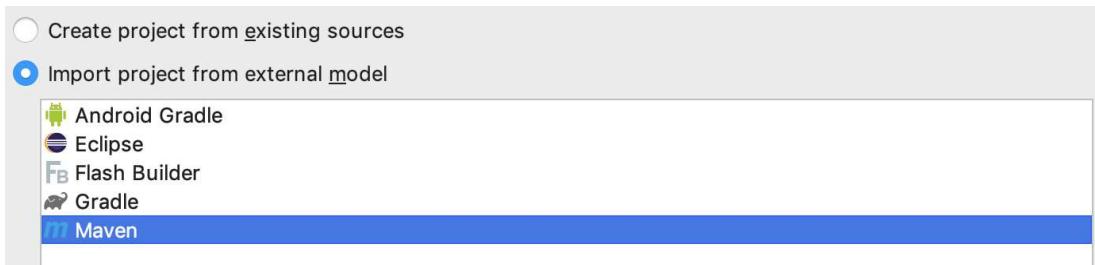
En el caso de la aplicación demo descargada, podemos probar las siguientes páginas:

- <http://localhost:8080>

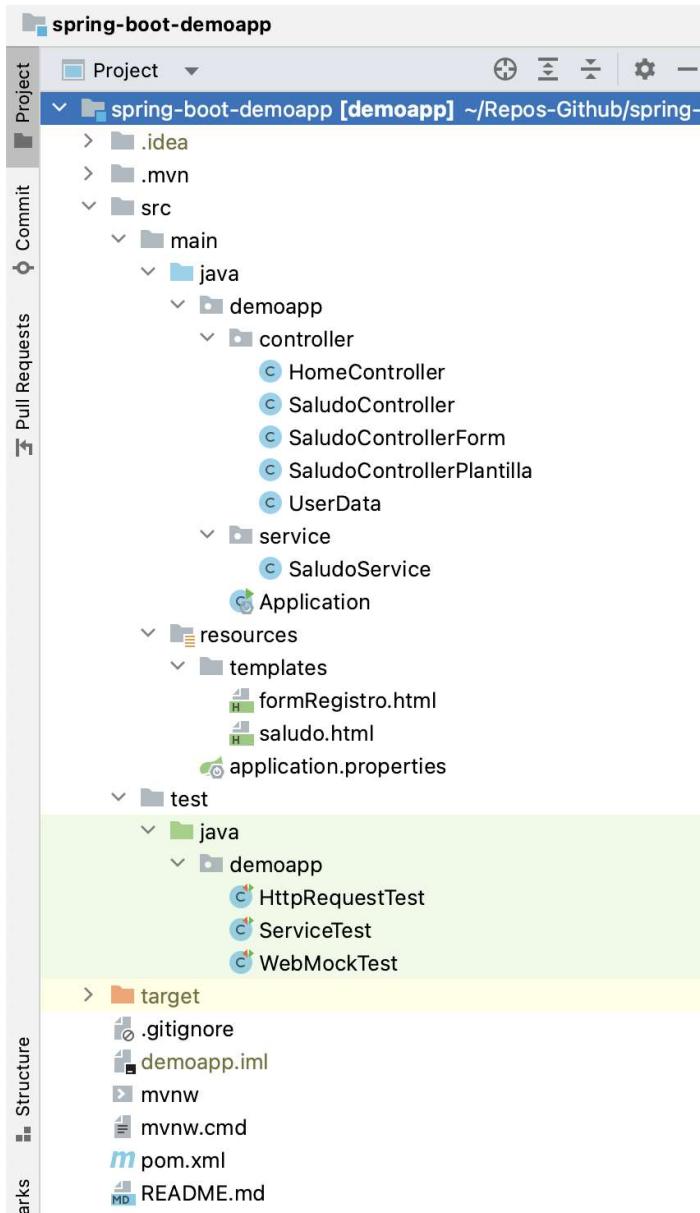
- <http://localhost:8080/saludo/Pepito>
- <http://localhost:8080/saludoplantilla/Pepito>
- <http://localhost:8080/saludoform>

Recomendamos hacer el desarrollo usando el IDE [IntelliJ Ultimate](#). Aunque es de pago, es posible [obtener una licencia de estudiante](#) usando la dirección de correo de la UA.

5. Abre proyecto el en IntelliJ. Debes importar el directorio donde se encuentre el fichero `pom.xml`. Se puede hacer desde la pantalla de bienvenida de IntelliJ con la opción **Import Project** o usando la opción **File > Open** o "**File > New > Project from Existing Sources**". Aparecerá la pantalla de importación y seleccionamos el importador **Maven**:



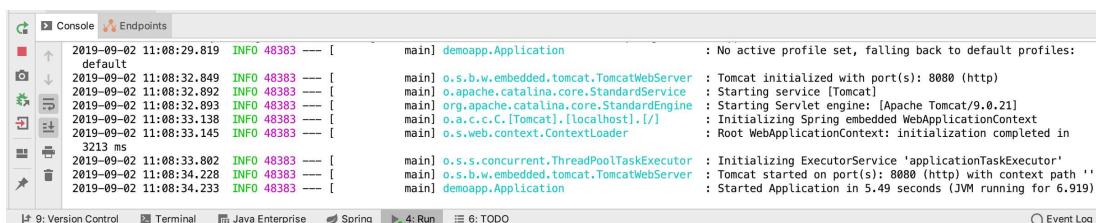
IntelliJ abre el proyecto correctamente:



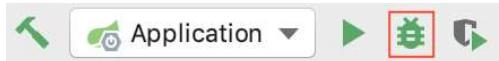
Podemos ejecutarlo abriendo un terminal y lanzándolo con Maven. O también desde la **configuración de Run** que ha creado IntelliJ al realizar la importación:



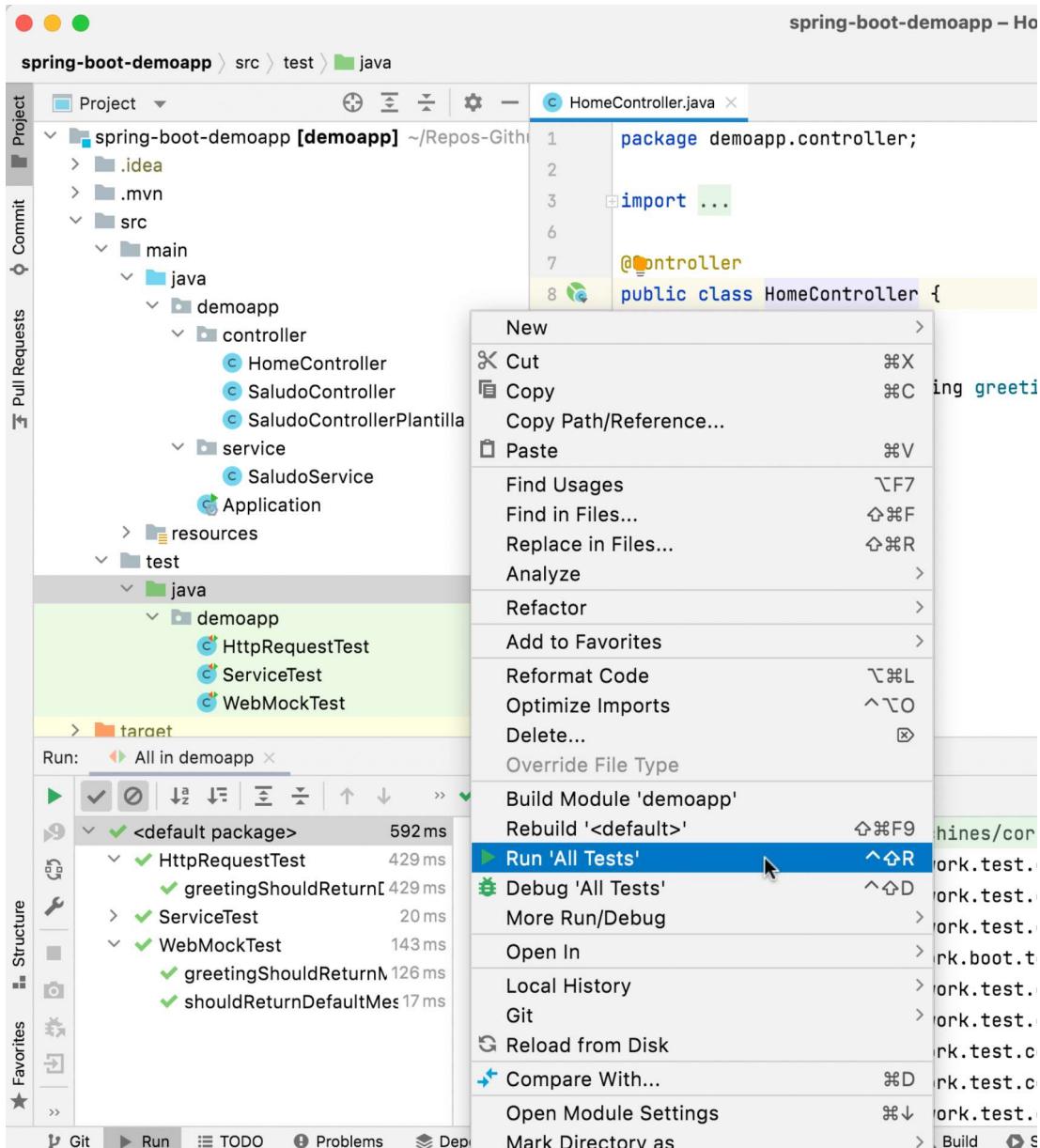
Se abrirá un panel de ejecución desde el que se puede parar la aplicación, volverla a lanzar, etc:



Desde la configuración de Run también podemos depurar el proyecto, pulsando el botón de depuración.



6. Lanza los tests desde el propio IntelliJ, pulsando en el panel del proyecto sobre el directorio de tests con el botón derecho. Los tests se lanzarán y aparecerá un panel en el que se mostrará si pasan correctamente (verde) o no.



Por último, haz algún pequeño cambio a la aplicación.

7. Cambia el mensaje de saludo que da el controller de la raíz para incluir tu nombre. Comprueba que los tests pasan (modifícalos si no es así) y que la aplicación funciona correctamente.



Hello World, me llamo Domingo

## Dockerización de la aplicación

**Docker** es un software de virtualización que utiliza el propio sistema operativo compartimentado y permite gestionar *contenedores* (similares a las máquinas virtuales) de forma mucho menos pesada y rápida que con sistemas de virtualización tradicionales como VirtualBox.

Las máquinas Docker son muy eficientes porque comparten los servicios del sistema operativo en el que se ejecutan, utilizando menos recursos que las máquinas virtuales tradicionales.

Docker proporciona un sistema muy sencillo de distribución y puesta en producción de software, ya que las máquinas Docker pueden ser distribuidas usando repositorios (como [Docker Hub](#)) y ejecutadas en cualquier ordenador que tenga instalado el *Docker Engine*.

La tecnología es muy popular y se usa en gran cantidad de empresas de desarrollo para simplificar la ejecución en múltiples entornos y para que los contenedores (máquinas Docker en ejecución) se puedan configurar y combinar o ejecutar en clusters usando herramientas como [Kubernetes](#).

En nuestro caso, vamos a construir una *máquina Docker* basada en la aplicación demo. Posteriormente, la publicaremos en *Docker Hub* y la desplegaremos en un *host* para ponerla en producción.

1. Instala [Docker Desktop](#). Los usuarios de Linux debéis seguir las instrucciones de [esta página](#) para instalar Docker Engine. Usaremos la línea de comando para lanzar los comandos Docker. La aplicación Docker Desktop permite usar una interfaz de usuario para interactuar con imágenes y contenedores, pero no proporciona ninguna funcionalidad que no esté disponible en la línea de comando.

Una vez instalado puedes probar el tutorial rápido (2 minutos) desde Docker Desktop para comprobar que todo funciona correctamente. También puedes desde el terminal comprobar la versión de Docker instalada:

```
$ docker version
```

2. Crea una cuenta de usuario en [Docker Hub](#). De esta forma tendrás un repositorio en el que podrás subir las imágenes de las máquinas Docker que construyas. Deberás dar un nombre de usuario que será el que utilizarás para publicar estas imágenes.

3. Crea un fichero llamado `Dockerfile` (sin extensión) en el directorio raíz de la aplicación con el siguiente contenido:

**Fichero `./Dockerfile`:**

```
FROM openjdk:8-jdk-alpine
COPY target/*.jar app.jar
ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/urandom", "-jar", "/app.jar"]
```

El fichero `Dockerfile` consiste en un conjunto secuencial de instrucciones con las que se construye la máquina Docker:

- `FROM`: este comando indica la máquina base sobre la que se van a ejecutar el resto de comandos. En nuestro caso una máquina de la organización `openjdk` que contiene la distribución 8 del Java Development Kit (JDK) y que está basada en una distribución linux Alpine. El primer paso de la construcción de nuestra máquina Docker consiste por tanto en descargar esta máquina `openjdk:8-jdk-alpine` y usarla como máquina base.
- `COPY`: este comando indica que se debe copiar un fichero o conjunto de ficheros de la máquina host (el directorio en el que estamos) en la máquina base. En este caso se copia el fichero JAR que constituye nuestra aplicación que está situado en el directorio `./target` y se copia en la máquina Docker con el nombre `app.jar`.
- `ENTRYPOINT`: este comando indica el comando a ejecutar cuando se pone en marcha la máquina Docker. En este caso se lanza la aplicación (`app.jar`) con el comando `java -jar`.

El modificador `-Djava.security.egd` hace que se inicialice el generador de números aleatorios de Java usando el fichero del sistema `/dev/urandom` en lugar del fichero por defecto `/dev/random`. Es necesario para resolver un bug que aparece al ejecutar el contenedor en un servidor alojado DigitalOcean (el servidor de la asignatura, en el que pondrás la aplicación en producción, usa ese servicio).

4. Asegúrate de que en el directorio raíz de la aplicación está el fichero JAR resultado de la compilación de la aplicación.

Tal y como se explica en la [introducción a Spring Boot](#), el fichero `.jar` es el ejecutable Java de la aplicación, que se crea cuando con el objetivo `package` de Maven:

```
$ ./mvnw package
$ ls -l ./target/*.jar
./target/demoapp-0.0.1-SNAPSHOT.jar
```

5. Ya puedes construir la máquina Docker con el siguiente comando, desde el directorio raíz de la aplicación (en el que debe estar el fichero `Dockerfile` anterior):

```
$ docker build -t <usuario-docker>/spring-boot-demoapp .
```

Comprueba que la imagen se ha creado correctamente. Debe aparecer en el *Docker Desktop* y con el comando `docker image ls`:

```
$ docker image ls
REPOSITORY           TAG
domingogallardo/spring-boot-demoapp    latest
```

#### 6. Pon en marcha un la imagen con la aplicación:

```
$ docker run -p 8080:8080 <usuario-docker>/spring-boot-demoapp
```

El comando `docker run` pone en marcha la imagen indicada, creando lo que se denomina un *contenedor Docker*. Es similar a una máquina virtual en ejecución. El parámetro `-p 8080:8080` indica que el puerto interno 8080 del contenedor se va a mapear en el puerto 8080 del *host*. De esta forma podremos conectarnos desde el *host* a la aplicación Spring Boot en funcionamiento.

Verás que en la consola aparecen los mensajes de salida de la aplicación Spring Boot que se ejecuta en el contenedor.

Prueba a abrir un navegador y conectarte a la URL [localhost:8080](http://localhost:8080). Deberás ver el mensaje de saludo de la aplicación ejecutándose en el contenedor.

Haciendo `ctrl+c` puedes parar el contenedor. El efecto es similar a suspender una máquina virtual. Puedes ver el identificador del contenedor con el comando:

```
$ docker container ls -a
CONTAINER ID   IMAGE          NAMES
5bd9d0b055a9   domingogallardo/spring-boot-demoapp  inspiring_feynman
```

Puedes usar tanto el ID del contenedor (`5bd9d0b055a9`) como su nombre (`inspiring_feynman`) para identificarlo.

Estando parado, puedes volver a poner en marcha el contenedor haciendo:

```
$ docker container start <identificador>
```

También podemos parar el contenedor:

```
$ docker container stop <identificador>
```

Y borrarlo definitivamente con

```
$ docker container rm <identificador>
```

Otros comandos útiles de Docker son:

- `docker run -d` : lanza el contenedor en modo *background*.
- `docker run --rm` : lanza el contenedor de forma que al pararlo se borra automáticamente.
- `docker container logs <identificador>` : muestra los logs del contenedor indicado.

En la aplicación Docker Engine podemos realizar también muchos de estos comandos interactuando directamente con la interfaz. Pruébalo.

7. Ahora que has comprobado que el fichero `Dockerfile` funciona correctamente debes añadirlo a git y subirlo al repositorio GitHub:

```
$ git status
$ git add .
$ git status
$ git commit -m "Añadido Dockerfile"
$ git push
```

8. Vamos a terminar publicando la imagen en tu cuenta de Docker Hub.

- Ve a [Docker Hub](#) y logéate.
- Crea un repositorio con el nombre `spring-boot-demoapp`. En ese repositorio vas a subir la imagen con el mismo nombre. En un repositorio Docker puedes mantener múltiples versiones de una misma imagen, usando *tags*.

**IMPORTANTE:** Escribe un enlace a al repositorio en Docker Hub en el fichero `README.md` del repositorio de GitHub. De esta forma podré consultar y descargar las imágenes que publique en Docker Hub.

- Una vez creado el repositorio puedes publicar la imagen en él logeándote desde la línea de comando (introduce tu usuario y contraseña de Docker Hub) y con el comando `docker push`:

```
$ docker login
$ docker push <usuario-docker>/spring-boot-demoapp
```

Verás que automáticamente se asigna la etiqueta `latest` (etiqueta por defecto) a la imagen y que ésta se sube al repositorio. Podrías asignar una etiqueta específica a la imagen con el comando `docker tag`. Por ejemplo, si quisieramos fijar esta imagen con la versión `1.0` podríamos hacerlo con el siguiente comando:

```
$ docker tag <usuario-docker>/spring-boot-demoapp <usuario-docker>/spring-boot-demoapp:1.0
```

- Comprueba en la página web del repositorio que se ha subido. El repositorio es público y cualquiera puede descargar la imagen haciendo:

```
$ docker pull <usuario-docker>/spring-boot-demoapp
```

Al no indicar la etiqueta, se descargaría la imagen etiquetada con `latest`. Si quisieramos descargar una versión concreta habría que especificar la etiqueta:

```
$ docker pull <usuario-docker>/spring-boot-demoapp:1.0
```

## Puesta en producción de la aplicación

Por último deberás poner en producción la aplicación, conectándote al servidor linux de la asignatura y poniendo allí en marcha la aplicación.

1. Consulta en el foro de Moodle la dirección IP del servidor linux de la asignatura y tu usuario.
2. Conéctate al servidor con tu usuario con la contraseña `mads22` y cambia tu contraseña.  
Por ejemplo, si tu usuario es `alu02` y la dirección IP del servidor es `160.66.120.177`:

```
$ ssh alu02@160.66.120.177  
$ passwd
```

3. Comprueba si alguien más está utilizando el servidor:

```
$ who  
alu02 pts/1 2021-08-11 07:01 (80.29.50.137)
```

Si algún otro compañero está usando el servidor puede ser posible que se ralentice o que ya el puerto `8080` ya esté ocupado por la aplicación del compañero. En este último caso puedes usar un puerto diferente para poner la aplicación en producción.

4. Descarga tu imagen de la aplicación y ponla en funcionamiento:

```
$ docker pull <usuario-docker>/spring-boot-demoapp  
$ docker run --rm --name spring-boot-alu<num> -p 8080:8080 <usuario-docker>/spring-boot-demoapp
```

El indicador `--rm` hace que cuando se pare el contenedor automáticamente se borre. De esta forma evitamos tener que borrarlo a mano después.

El indicador `--name` define el nombre del contenedor. Ponemos nuestro nombre, para poder identificar quiénes han creado cada contenedor.

En el caso en que otro compañero tenga la aplicación en marcha en ese puerto aparecerá el siguiente mensaje de error:

```
docker: Error response from daemon: driver failed programming ...
Bind for 0.0.0.0:8080 failed: port is already allocated.
```

En ese caso puedes usar otro puerto. El primer puerto es el que se refiere al host y el segundo a la aplicación corriendo en el contenedor. Por ejemplo, puedes usar el puerto 8081:

```
$ docker run --rm --name spring-boot-alu02 -p 8081:8080 <usuario-docker>/spring-boot-demoapp
```

5. Comprueba que la aplicación funciona correctamente conectándote desde tu navegador al servidor linux de la asignatura y al puerto correctamente: <http://161.35.65.197:8080>.  
¡Enhорabuena, ya tienes tu aplicación en producción! Puedes llamar a cualquier amigo para que se conecte a esa URL y la pruebe.
6. Si en algún momento tenemos problemas de espacio en el disco duro, podemos borrar la imagen y el contenedor:

```
$ docker container ls -a
$ docker container rm spring-boot-alu02
$ docker image ls -a
$ docker image rm <nombre-imagen> o <image-id>
$ exit
```



#### No guardar ficheros en el servidor de la asignatura

El servidor de la asignatura tiene una capacidad limitada de disco duro y debemos tener cuidado de no sobrepasarla entre todos.

Por ello, no debes guardar ningún fichero ajeno a la asignatura en este servidor.

## 4. Estudia el funcionamiento de la aplicación y su arquitectura

En el documento [Introducción a Spring Boot](#) se comenta el código fuente de la aplicación Spring Boot con la que estamos trabajando. Léelo despacio, revisando también el código fuente, para entender los aspectos básicos (controladores, servicios, inyección de dependencias, plantillas) del funcionamiento de Spring Boot.

Estudia también despacio el funcionamiento de los tests y el funcionamiento del formulario y la validación.

Puedes ver un ejemplo adicional de validación de un formulario en el repositorio [domingogallardo/spring-boot-validate](#).

Verás también ahí varios ejemplos de tests en los que se realiza una petición POST pasando parámetros y se obtiene información del modelo resultante, llamando al método `model()`. El siguiente es un ejemplo de uno de los tests:

```
@Test  
public void checkPersonInfoWhenNameTooShortThenFailure() throws Exception {  
    mockMvc.perform(post("/")  
        .param("name", "R")  
        .param("age", "20"))  
        .andExpect(model().hasErrors());  
}
```

## 5. Añadimos alguna funcionalidad sencilla a la aplicación

Para demostrar que comprendes el funcionamiento de una aplicación Spring Boot, debes añadir alguna funcionalidad sencilla a la aplicación Demo que realice lo siguiente:

- Leer datos de un formulario usando Thymeleaf y **realizar alguna validación**.
- Llamar a un **método de servicio** que procese los datos leídos.
- Mostrar el resultado devuelto por el servicio en una página Thymeleaf.
- Incluir al menos 2 tests:
  - 1 de la capa de servicio
  - 1 de la capa de presentación usando MockMvc.
- En la página principal de la aplicación debe aparecer tu nombre y apellidos.

Muy importante, debemos desarrollar la aplicación en **pequeños commits**. Cada commit debe compilar correctamente y añadir una pequeña funcionalidad. Debemos subir los commits al repositorio personal de GitHub.

Debes definir tú la funcionalidad a implementar. Por ejemplo, cualquiera de los siguientes ejemplos o alguno similar que se te ocurra:

- Palíndroma: lee una palabra y comprueba si es palíndroma.
- Número par: lee un número y comprueba si es par
- Cuadrado: lee dos números y comprueba si el segundo es el cuadrado del primero
- Calculadora: lee un par de números y una operación y devuelve el resultado.

Cuando compruebes que los tests funcionan correctamente y que la aplicación funciona bien en local, debes crear la máquina Docker con la etiqueta `final` y probar que funciona bien en producción en el servidor de la asignatura.

## 6. Comandos Git

Comandos Git necesarios para realizar la práctica:

- `git clone`
- `git status`
- `git add`
- `git commit`
- `git push`
- `git log`

Puedes encontrar más información sobre estos comandos en el documento [Resumen de comandos Git](#) que resume los conceptos más importantes de Git necesarios para estas primeras prácticas de la asignatura.

## 7. Entrega

- La práctica tiene una duración de 1 semana y debe estar terminada el martes 21 de septiembre. El miércoles 22 de septiembre el profesor comprobará en clase de prácticas el funcionamiento de la práctica en producción.
- La calificación de la práctica tiene un peso de un 10% en la nota final de prácticas.

Para realizar la entrega debes hacer lo siguiente:

- Realizar la aplicación en el repositorio creado e ir subiendo los commits a GitHub conforme se van realizando.
- Actualizar el fichero `README.md` con la URL del repositorio Docker Hub donde se ha subido la máquina Docker final.
- Añadir una página de documentación `doc/practica1.md` en la que se explique la funcionalidad y el código añadido. Incluir en la documentación la URL de los repositorios en GitHub y en Docker Hub. Deberás escribir esta documentación en Markdown. Tienes disponible en GitHub una breve pero útil [introducción a Markdown](#).
- Entregar en Moodle un ZIP con el directorio del proyecto (incluyendo el directorio `.git` con el repositorio git), después de haber hecho `./mvnw clean` para eliminar los binarios compilados.

Para la evaluación se tendrá en cuenta:

- Desarrollo continuo (los *commits* deben realizarse a lo largo de toda la semana y no dejar todo para el último día).
- Correcto desarrollo de la metodología.

- Diseño e implementación del código y de los tests de las características desarrolladas.

**Importante**

Después de entregar la práctica deberás ponerla en producción en el servidor de la asignatura y el profesor comprobará que funciona correctamente. Lo haremos en el horario de clase de prácticas.