

Deploy Backend Springboot en Fly.io

Al desarrollar una aplicación web, la progresión natural es ponerla en línea y ponerla a disposición de los usuarios finales. Para que esta tarea sea posible y más fácil, existen numerosas plataformas en la nube disponibles para elegir para alojar su aplicación. Fly.io es una de ellas, la cual nos permite implementar cualquier tipo de aplicación de tipo backend o frontend. También admite implementar aplicaciones que estén en un **dockerfile**. Eso significa que puede simplemente implementar una imagen de Docker.

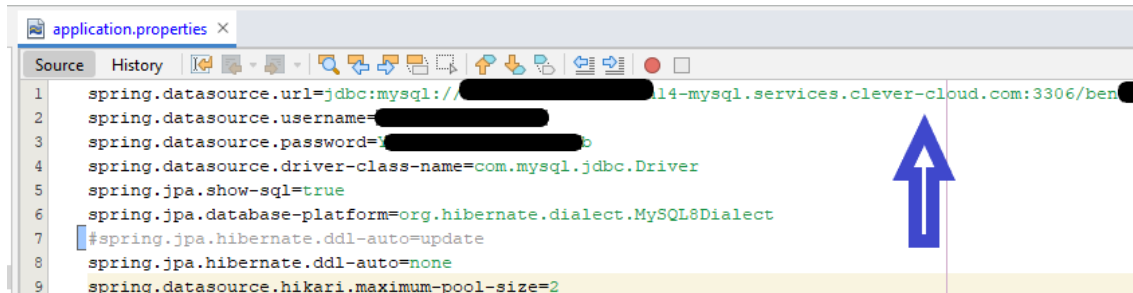
Desplegar aplicación Springboot en Fly.io usando Dockerfile

La siguiente guía sirve únicamente para generar el despliegue de una aplicación Springboot en la nube de fly.io de forma totalmente gratuita.

Para profundizar sobre la generación de imágenes docker de aplicaciones locales de springboot puede seguir la siguiente guía: <https://spring.io/guides/gs/spring-boot-docker/>

Pre requisitos:

- Tener compilación funcionando de proyecto springboot en su computadora local.
- Tener configurada la conexión a la base de datos en la nube de clevercloud:



```
1 spring.datasource.url=jdbc:mysql://[redacted].14-mysql.services.clever-cloud.com:3306/[redacted]
2 spring.datasource.username=[redacted]
3 spring.datasource.password=[redacted]
4 spring.datasource.driver-class-name=com.mysql.jdbc.Driver
5 spring.jpa.show-sql=true
6 spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
7 #spring.jpa.hibernate.ddl-auto=update
8 spring.jpa.hibernate.ddl-auto=none
9 spring.datasource.hikari.maximum-pool-size=2
```

- Tener habilitado CORS en tus endpoint: En algunos casos puede aparecer el error “No hay un encabezado 'Access-Control-Allow-Origin' presente en el recurso solicitado” al intentar invocar la API.

Los errores de uso compartido de recursos entre orígenes (CORS) ocurren cuando un servidor no devuelve los encabezados HTTP que exige el estándar CORS. Para resolver un error CORS de una API REST de API Gateway o API HTTP, debe configurar de nuevo la API para que cumpla con el estándar CORS.

Existen varias alternativas para solucionarlo, lo que se propone es agregar una clase en tu backend para habilitar todos los orígenes que peticionan:

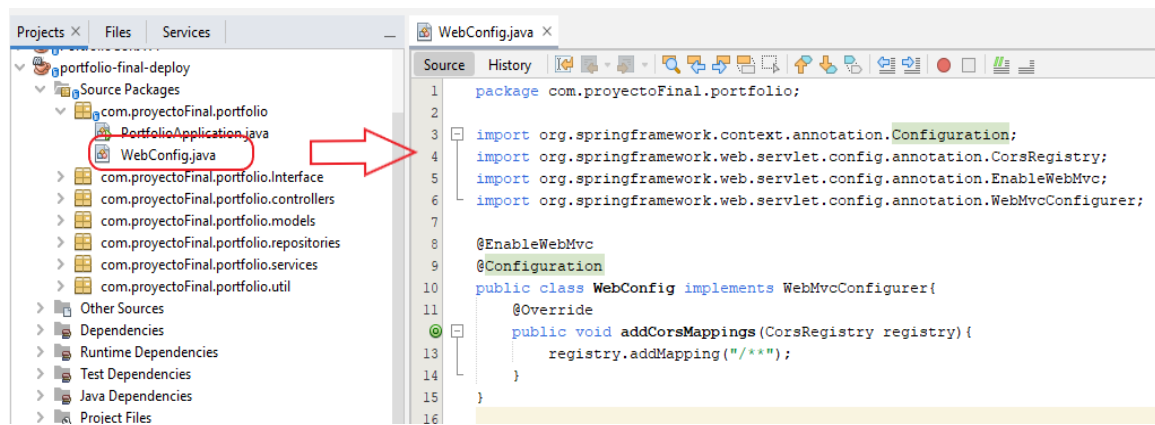
```
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
```

```

/**
 * Clase que habilita CORS
 * @author YOProgramo
 */
@EnableWebMvc
@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/*");
    }
}

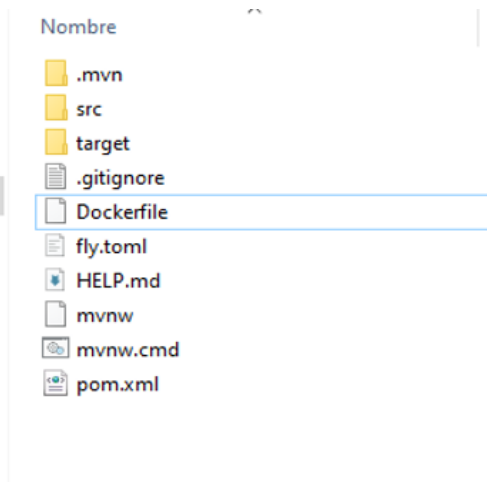
```

Su clase debería quedar tal cual muestra el ejemplo:



A continuación, se detallan los pasos a seguir:

1. Crear archivo *Dockerfile*, en la carpeta de tu proyecto springboot:



2. Al archivo creado agregar la **dependencia** necesaria para levantar una versión de JDK igual a la que usas en tu proyecto local, en el ejemplo suponemos que tenemos la versión 11, por lo tanto elegimos una versión JDK11: **amazoncorretto:11-alpine-jdk** (Existen múltiples paquetes de JDK, puedes elegir el que prefieras en: https://hub.docker.com/_/openjdk) y la referencia del archivo **.jar** que contiene la compilación de nuestro proyecto springboot:

Código base:

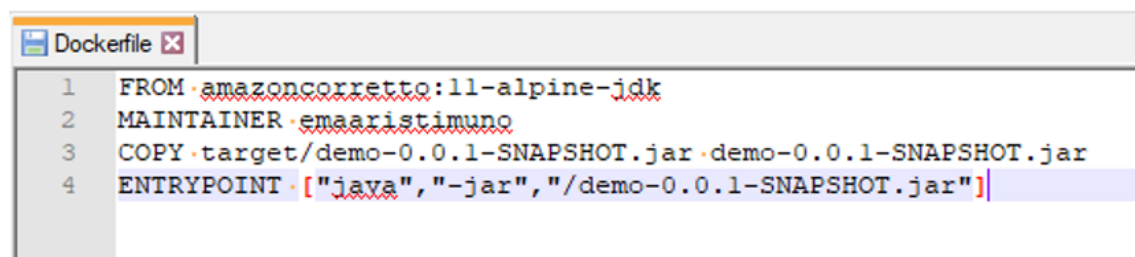
```
FROM amazoncorretto:11-alpine-jdk
```

```
MAINTAINER emaaristimuno
```

```
COPY target/NAME-YOUR-FILE-BUILD-SPRINGBOOT.jar NAME-YOUR-FILE-BUILD-SPRINGBOOT.jar
```

```
ENTRYPOINT ["java", "-jar", "/NAME-YOUR-FILE-BUILD-SPRINGBOOT.jar"]
```

Ejemplo:



Para el ejemplo, la compilación que se tendrá en cuenta para crear la imagen se encuentra dentro de la carpeta target:

target/demo-0.0.1-SNAPSHOT.jar

demo > target				
Nombre	Fecha de modificación	Tipo		
classes	03/10/2022 16:11	Carpeta de archivos		
generated-sources	03/10/2022 16:11	Carpeta de archivos		
generated-test-sources	03/10/2022 16:11	Carpeta de archivos		
maven-archiver	03/10/2022 16:11	Carpeta de archivos		
maven-status	03/10/2022 16:11	Carpeta de archivos		
surefire-reports	03/10/2022 16:11	Carpeta de archivos		
test-classes	03/10/2022 16:11	Carpeta de archivos		
demo-0.0.1-SNAPSHOT.jar	03/10/2022 16:11	Archivo JAR		
demo-0.0.1-SNAPSHOT.jar.original	03/10/2022 16:11	Archivo ORIGINAL		


3. Instalar utilidad de comandos para trabajar con Fly: **flyctl**. Ejecutando en Powershell de Windows:

```
$ iwr https://fly.io/install.ps1 -useb | iex
```

4. Si esta es tu primera vez con Fly.io, tu próximo paso será [Registrarte](#) (usa tu cuenta de github):



Sign in to Your Account

 Sign in with Github

or

Email Address

Password

[Forgot Your Password?](#)

[Need an Account?](#)




Sign In

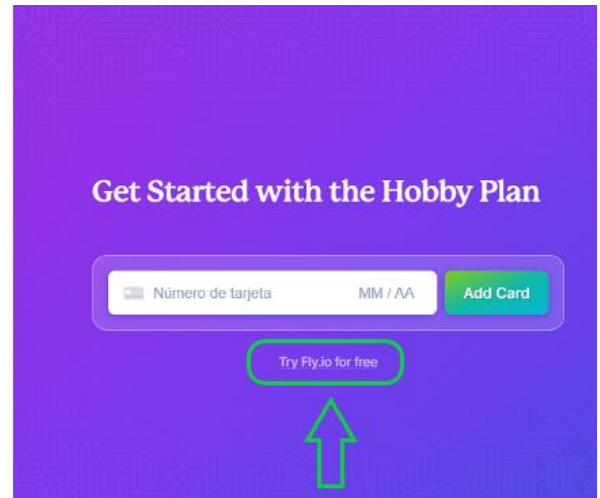
5. Te pedirá que ingreses un método de pago, pero no te preocupes puedes Elegir **Try Fly.io for Free** como plan para comenzar:



GET STARTED WITH OUR FREE TIER

Add a Payment Method

-  **Apps**
Run up to 3 shared CPU VMs for free, full time.
-  **Volumes**
Volumes from 1 to 500GB, up to 3GB are free.
-  **IP addresses**
Unlimited IPv6 and at least one free IPv4 per app.



6. Iniciar sesión en Fly:

```
$ flyctl auth signup
```

Esto lo llevará a la página de registro donde puede:

Regístrese con correo electrónico: Ingrese su nombre, correo electrónico y contraseña.

Regístrese con GitHub: si tiene una cuenta de GitHub, puede usarla para registrarse. Esté atento al correo electrónico de confirmación que se enviara, que le dará un enlace para establecer una contraseña; necesitará establecer una contraseña para que podamos verificar activamente que es usted para algunas operaciones de Fly.io.

7. Cada aplicación Fly.io necesita un archivo **fly.toml** para decirle al sistema cómo nos gustaría implementarlo. Ese archivo se puede generar automáticamente con el comando **flyctl launch**, que hará algunas preguntas para configurar todo. Repasémoslo ahora:

7.1 Ejecutar los siguientes comandos dentro del directorio donde se encuentra el dockerfile:

```
> flyctl launch
```

7.2 Definir un nombre para nuestra aplicación desplegada en Fly:

```
Creating app in C:\Users\ema_2\Documents\NetBeansProjects\BackendArgProg22-main
Scanning source code
Detected a Dockerfile app
? Choose an app name (leave blank to generate one): deploy-springboot
```

7.3 Seleccionar la región que levantara nuestra app:

```
? Choose an app name (leave blank to generate one): deploy-springboot
automatically selected personal organization: emanueluader@gmail.com
? Choose a region for deployment: [Use arrows to move, type to filter]
London, United Kingdom (lhr)
Chennai (Madras), India (maa)
Madrid, Spain (mad)
Miami, Florida (US) (mia)
Tokyo, Japan (nrt)
Chicago, Illinois (US) (ord)
Bucharest, Romania (otp)
> Santiago, Chile (scl)
Seattle, Washington (US) (sea)
Singapore, Singapore (sin)
San Jose, California (US) (sjc)
```

Nos creará un archivo *fly.toml* con las configuraciones necesarias:

```
? Choose an app name (leave blank to generate one): deploy-springboot
automatically selected personal organization: emanueluader@gmail.com
? Choose a region for deployment: Santiago, Chile (scl)
Created app deploy-springboot in organization personal
Wrote config file fly.toml
```

7.4 Nos pregunta si queremos instalar una Base de Datos Postgresql, seleccionar que **No**:

```
? Would you like to set up a Postgresql database now? No
```

7.5 Generar el Deploy respondiendo **Yes** a la pregunta:

```
? Would you like to deploy now? Yes
==> Building image
Waiting for remote builder fly-builder-long-cherry-2293... 🌐
```

Esperar un momento hasta que se genere todos los procesos necesarios para subir la imagen de docker a fly (Según su conexión de internet puede demorar entre 5 y 30 minutos :/):

```

? Would you like to deploy now? Yes
==> Building image
Remote builder fly-builder-long-cherry-2293 ready
==> Creating build context
--> Creating build context done
==> Building image with Docker
--> docker host: 20.10.12 linux x86_64
[+] Building 1471.7s (0/1)
[+] Building 2.8s (6/6) FINISHED

```

Si el despliegue se realizó de manera exitosa, la consola arrojará los siguientes mensajes:

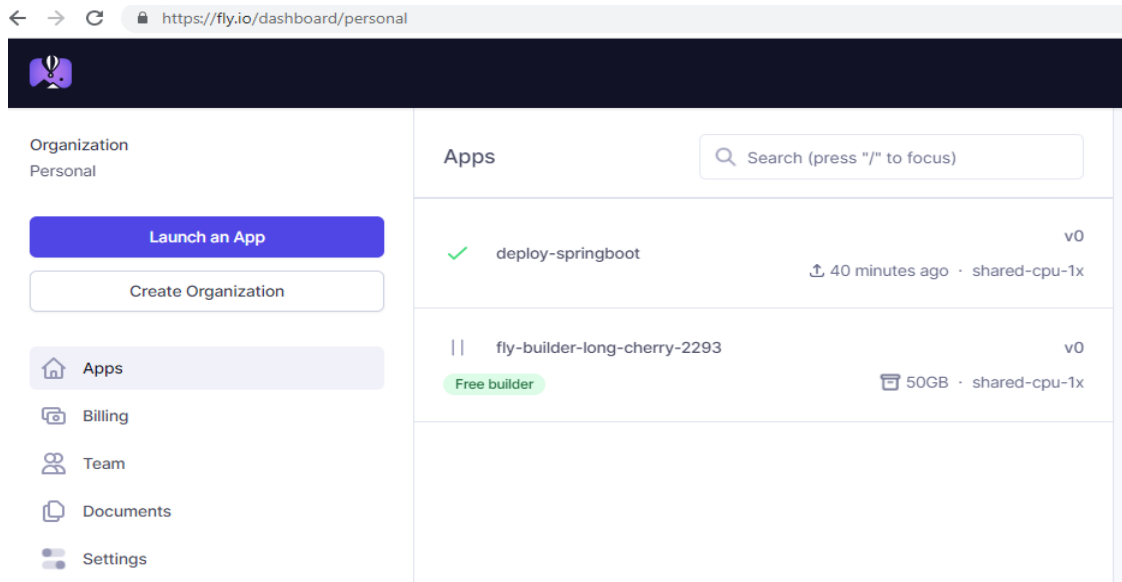
```

[+] Building 2.8s (6/6) FINISHED
=> [internal] load remote build context
=> copy /context /
=> [internal] load metadata for docker.io/library/amazoncorretto:11-alpine-jdk
=> [1/2] FROM docker.io/library/amazoncorretto:11-alpine-jdk@sha256:9eddc0fc6e35c4a09fc402bf67e0aac986ae01e96fb0cb74059f4914016a24d
=> CACHED [2/2] COPY target/portfolio-final-deploy-0.0.1-SNAPSHOT.jar portfolio-final-deploy-0.0.1-SNAPSHOT.jar
=> exporting to image
=> => exporting layers
=> => writing image sha256:caa5d2152a0fcadfe5a9aaf277c68f5d2231563de47a49cb636bc8b8635e841
=> => naming to registry.fly.io/deploy-springboot:deployment-01GH19ATHCPVQ4FKYAWWRC39Y0
--> Building image done
==> Pushing image to fly
The push refers to repository [registry.fly.io/deploy-springboot]
2af73c1c4a30: Pushed
a7579190b8e3: Pushed
994393dc58e7: Pushed
deployment-01GH19ATHCPVQ4FKYAWWRC39Y0: digest: sha256:795c4e6c6307746bc806be58bd04c163e75119032b10c41394651ca0d95d3d10 size: 953
--> Pushing image done
image: registry.fly.io/deploy-springboot:deployment-01GH19ATHCPVQ4FKYAWWRC39Y0
image size: 378 MB
==> Creating release
--> release v2 created

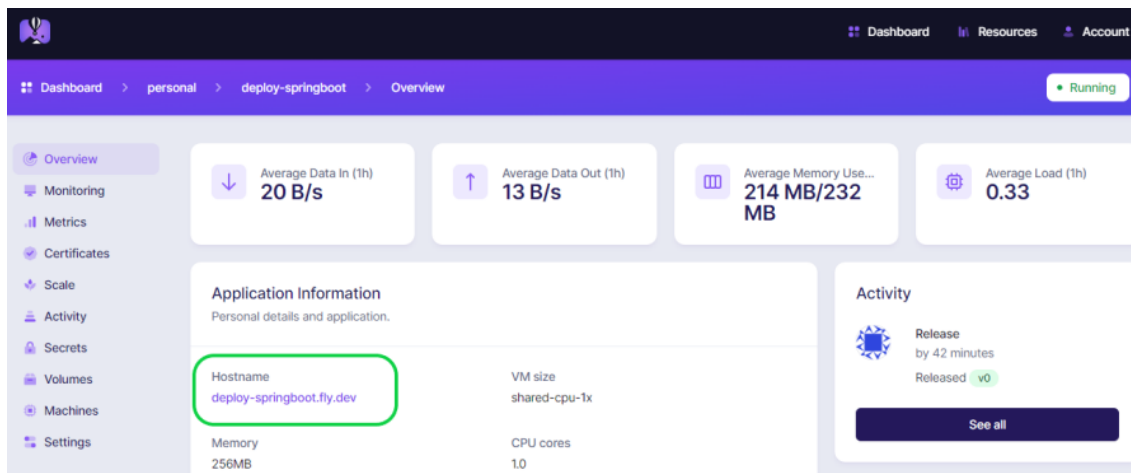
--> You can detach the terminal anytime without stopping the deployment

```

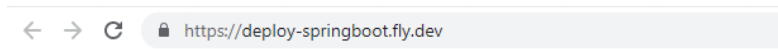
8. Para revisar la aplicación desplegada deberás dirigirte a <https://fly.io/dashboard>



9. Seleccionar la app, Ejemplo: **deploy-springboot**



10. Finalmente, verificar el correcto funcionamiento del servidor:



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Fri Nov 04 13:29:37 GMT 2022

There was an unexpected error (type=Not Found, status=404).

Referencia: <https://fly.io/docs/languages-and-frameworks/dockerfile/>