



Microservicios con Spring Boot 3 y Spring Cloud

Dockerizar una aplicación	2
Buildpack	2
Subiendo las imágenes a docker hub	3
Docker Compose	3
Crear contenedores con docker compose	4
Ejecutar el docker compose	4
Configuración centralizada en microservicios	4
Generando microservicio config server	5
Actualizar Microservicio Hotels para que tome las configuraciones	7
Configurar application.properties de los microservicios	8
Crear clase de configuración en los microservicios	8
Actualizando imágenes Docker	10
Crear imagen para el microservicio ConfigServer	10
Crear nuevamente las imágenes docker de hotels	10
Subir las imágenes a Docker Hub	11
Actualizando docker-compose.yml	11
Levantar contenedores de imágenes	13



Dockerizar una aplicación

1) Crear docker file

```
#inicia con la imagen base que contiene Java runtime
FROM openjdk:17-jdk-slim as build

# se agregar el jar del microservicio al contenedor
COPY target/hotels-0.0.1-SNAPSHOT.jar hotels-0.0.1-SNAPSHOT.jar

#se ejecuta el microservicio
ENTRYPOINT ["java", "-jar", "/hotels-0.0.1-SNAPSHOT.jar"]
```

2) docker build . -t jzapana/hotels

3) docker images

docker inspect <tres caracteres del id>

4) corremos la imagen para crear el contenedor

docker run -p 8080:8080 jzapana/hotels

docker ps (para ver los contenedores que están corriendo)

docker run -p 8090:8080 jzapana/hotels (inicia dentro de otro contenedor)

Buildpack

Es otra opción para crear una imagen de nuestro proyecto, mediante un comando de Maven utilizado en proyectos de Spring Boot que te permite construir una imagen de contenedor Docker a partir de una aplicación de Spring Boot. Este comando es parte del ecosistema de Spring Boot y se utiliza en combinación con **Spring Boot's Native Image Support** para crear una imagen de contenedor que contiene la aplicación de Spring Boot.

Maven se encargará de realizar varias tareas, incluyendo:

- Compilar la aplicación de Spring Boot.
- Empaquetarla en un formato adecuado para ser incluido en una imagen de contenedor Docker.
- Generar un archivo Dockerfile si no lo has proporcionado explícitamente en el proyecto.
- Invocar Docker para construir la imagen de contenedor.

El resultado final será una imagen de contenedor Docker que contiene la aplicación de Spring Boot, lista para ser ejecutada en un entorno de contenedor.

Para utilizar buildpack se debe agregar lo siguiente en el pom del proyecto correspondiente



```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <configuration>
        <excludes>
          <exclude>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
          </exclude>
        </excludes>
        <image>
          <name>jzapana/${project.artifactId}</name>
        </image>
      </configuration>
    </plugin>
  </plugins>
</build>
```

para generar la imagen se debe correr el siguiente comando, tener en cuenta que docker debe estar funcionando (la segunda opción es para evitar los test):

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

Luego las imágenes se pueden ver con

```
docker images
```

Iniciamos el microservicio en el puerto que corresponda:

```
docker run -p 8081:8081 jzapana/rooms
```

Subiendo las imágenes a docker hub

```
login -u "jzapana" -p xxxxx
docker push docker.io/jzapana/hotels
```

Docker Compose

Es otra herramienta para gestionar contenedores y crear imágenes. Es útil cuando hay muchos microservicios en la aplicación. Previamente hay que verificar que el docker desktop tenga configurado el docker compose con el comando

```
docker-compose --version
```

También es recomendable verificar si están instaladas las últimas actualizaciones de docker



Crear contenedores con docker compose

Se realiza mediante un archivo yaml, aquí se indican los servicios que son candidatos para crear los contenedores, es decir los que se quieren dockerizar. Para cada servicio se indica la imagen que se va a descargar de docker hub (por ejemplo: jzapana/hotels:latest)

```
services:
  hotels:
    image: jzapana/hotels:latest
    mem_limit: 800m
    ports:
      - "8080:8080"
    networks:
      - jzapana-network

  rooms:
    image: jzapana/rooms:latest
    mem_limit: 800m
    ports:
      - "8081:8081"
    networks:
      - jzapana-network

  reservations:
    image: jzapana/reservations:latest
    mem_limit: 800m
    ports:
      - "8082:8082"
    networks:
      - jzapana-network

networks:
  jzapana-network:
```

Ejecutar el docker compose

`docker compose up`

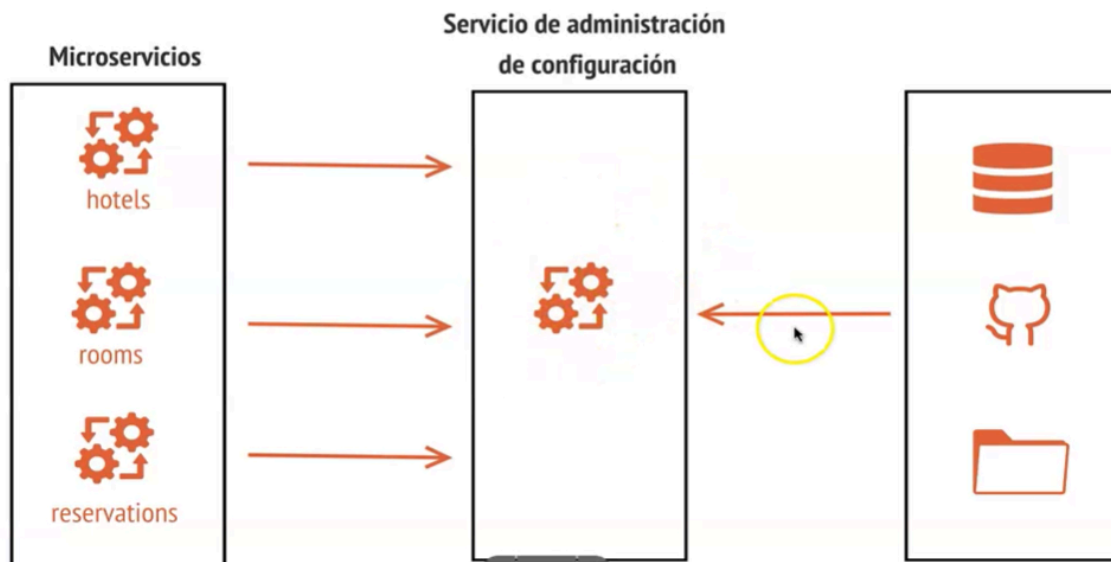
Configuración centralizada en microservicios

- Permite desacoplar la configuración de los archivos properties



- Inyectan las configuraciones que el microservicio necesita cuando se levanta en los diferentes ambientes.
- Mantener las configuraciones en un lugar centralizado con distintas versiones

Administración de Configuraciones



Spring Cloud Config

<https://spring.io/projects/spring-cloud>

- Da soporte para externalizar configuraciones en sistemas distribuidos (microservicios)
- Con un servidor de configuraciones se cuenta con un lugar centralizado para administrar configuraciones de aplicaciones, para todos los ambientes.

Generando microservicio config server

- Crear un proyecto denominado configserver
- Agregar las siguientes dependencias
 - Config Server
 - Spring Boot Actuator



El archivo Pom debe incluir el tag image tal como se muestra a continuación (al igual que el ms rooms)

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <configuration>
        <image>
          <name>jzapana/${project.artifactid}</name>
        </image>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Configurar el proyecto

- **Agregar la anotación @EnableConfigserver** en la clase principal del proyecto. Esta anotación permite que el proyecto trabaje como un servidor de configuraciones y permita leer toda la información de configuración desde un repositorio git.
- **Crear otro proyecto en gitlab denominado configserver-ms** que contendrá los archivos de configuración para todo el proyecto, este proyecto debe ser de tipo público (no privado).
- **Configurar el archivo de propiedades** para que permita leer la información desde gitlab (ver documentación en proyecto spring cloud)

```
spring.application.name=configserver
```

```
spring.profiles.active=git
```

```
#configurar la ubicación del proyecto gitlab
```

```
spring.cloud.config.server.git.uri=https://gitlab.com/cu-rso-microservicios2435144/configserver-ms.git
```



```
spring.cloud.config.server.git.clone-on-start=true  
spring.cloud.config.server.git.default-label=main
```

```
server.port=8085
```

- Probar proyecto: iniciar el proyecto configserver y probar en el browser <http://localhost:8085/hotels/prod>
 - Puede agregar el plugin JSON Viewer para mejorar chrome

Actualizar Microservicio Hotels para que tome las configuraciones

El objetivo es agregar la configuración de spring cloud a los microservicios que van a consumir los valores que están guardados en configserver-ms.

En los POM de los microservicios que van a consumir la configuración se debe agregar la misma versión de spring cloud que figura en el microservicio configserver-ms, esto se agrega debajo de la versión de java de cada microservicio, es decir:

```
<properties>  
  <java.version>17</java.version>  
  <spring-cloud.version>2022.0.4</spring-cloud.version>  
</properties>
```

También se debe copiar toda la sección de las dependency-management, se debe pegar debajo de dependencies:

```
<dependencyManagement>  
  <dependencies>  
    <dependency>  
      <groupId>org.springframework.cloud</groupId>  
      <artifactId>spring-cloud-dependencies</artifactId>  
      <version>${spring-cloud.version}</version>  
      <type>pom</type>  
      <scope>import</scope>  
    </dependency>  
  </dependencies>  
</dependencyManagement>
```

Los microservicios también tienen que incluir la dependencia de spring cloud:

```
<dependency>
```



```
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-config</artifactId>
</dependency>
```

Configurar application.properties de los microservicios

Finalmente resta que los microservicios lean las configuraciones, para ello agregamos esta configuración en el application.properties de los microservicios.

Optional: significa que si está caído el ms de configuración que NO impida que los demás microservicios funcionen.

```
spring.application.name=hotels
```

```
#ambiente, en este caso dev
spring.profiles.active=dev
```

```
#nombre obtenido de la clave spring.application.name del configserver-ms
spring.config.import=optional:configserver:http://localhost:8085
```

Crear clase de configuración en los microservicios

```
package com.reservahotel.hotels.config;
import lombok.Data;
/**
 * Configuración para consumir las properties que empiecen con "hotels"
 * @author jzapana
 *
 */
@Configuration
@ConfigurationProperties(prefix="hotels")
@Data
public class HotelsServiceConfiguration {

    private String msg;
    private String buildVersion;
    private Map<String,String>mailDetails;

}
```

Se define un modelo para leer las propiedades

```
package com.reservahotel.hotels.model;
```




```
import java.util.Map;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.Setter;

/**
 *
 * @author jzapana
 */
@Getter
@Setter
@AllArgsConstructor
public class PropertiesHotels {
    private String msg;
    private String buildVersion;
    private Map<String, String> mailDetail;
}
```

y finalmente se agrega un endpoint en el controlador de hoteles para realizar pruebas, para esto se inyecta una instancia de la configuración.

```
@Autowired
private HotelsServiceConfiguration configHotels;

/**
 * EndPoint de prueba para leer las properties del microservicio
 * configserver-ms
 *
 * @return
 * @throws JsonProcessingException
 */
@GetMapping("/hotels/read/properties")
public String getPropertiesHotels() throws JsonProcessingException{
    ObjectWriter owj = new ObjectMapper().writer().withDefaultPrettyPrinter();
    PropertiesHotels propHotels = new PropertiesHotels(configHotels.getMsg(),
configHotels.getBuildVersion(),configHotels.getMailDetails());

    String jsonString = owj.writeValueAsString(propHotels);
    return jsonString;
}
```

Resultado del endpoint



```
localhost:8080/hotels/read/properties
1 // 20231106202334
2 // http://localhost:8080/hotels/read/properties
3
4 {
5   "msg": "Welcome to the Reservations Hotels applications",
6   "buildVersion": "1",
7   "mailDetail": {
8     "hostName": "dev-alfredo@gmail.com",
9     "port": "5400",
10    "from": "dev-alfredo@gmail.com",
11    "subject": "Your Reservations is ready"
12  }
13 }
```

Actualizando imágenes Docker

Crear imagen para el microservicio ConfigServer

previamente hay que corregir el nombre del artifact-id de este microservicio ya que no debe tener guiones ni mayúsculas, es decir que debe quedar del siguiente modo:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://maven.apache.org/POM/4.0.0" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.1.5</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.reservahotel.configserver</groupId>
  <artifactId>configserver</artifactId>
```

para dockerizar ejecutamos el comando:

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

Crear nuevamente las imágenes docker de hotels

Crear el jar y construir las imágenes con build pack



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

```
mvn clean install  
docker build . -t jzapana/hotels
```

Subir las imágenes a Docker Hub

```
docker push docker.io/jzapana/hotels  
docker push docker.io/jzapana/rooms  
docker push docker.io/jzapana/reservations  
docker push docker.io/jzapana/configserver
```

Esto es necesario porque docker-compose.yml baja la última versión de estas imágenes y empieza a construir los contenedores.

Actualizando docker-compose.yml

El docker compose fue creado en el microservicio de reservations para generar las imágenes para luego levantar el contenedor y poder utilizarlas.

services:

```
configserver:  
  image: jzapana/configserver:latest  
  mem_limit: 800m  
  ports:  
    - "8085:8085"  
  networks:  
    - jzapana-network
```

```
eureka-server:  
  image: jzapana/eureka-server:latest  
  mem_limit: 800m  
  ports:  
    - "8065:8065"  
  networks:  
    - jzapana-network  
  depends_on:  
    - configserver  
  deploy:  
    restart_policy:
```



```
    condition: on-failure
    delay: 5s
    max_attempts: 3
    window: 120s
environment:
  SPRING_PROFILES_ACTIVE: default
  SPRING_CONFIG_IMPORT: configserver:http://configserver:8085/

hotels:
  image: jzapana/hotels:latest
  mem_limit: 800m
  ports:
    - "8080:8080"
  networks:
    - jzapana-network
  depends_on:
    - configserver
    - eureka-server
  deploy:
    restart_policy:
      condition: on-failure
      delay: 5s
      max_attempts: 3
      window: 120s
  environment:
    SPRING_PROFILES_ACTIVE: default
    SPRING_CONFIG_IMPORT: configserver:http://configserver:8085/
    EUREKA_CLIENT_SERVICEURL_DEFAULTZONE:
http://eureka-server:8065/eureka/

rooms:
  image: jzapana/rooms:latest
  mem_limit: 800m
  ports:
    - "8081:8081"
  networks:
    - jzapana-network
  depends_on:
    - configserver
    - eureka-server
  deploy:
    restart_policy:
      condition: on-failure
```



```
    delay: 5s
    max_attempts: 3
    window: 120s
environment:
  SPRING_PROFILES_ACTIVE: default
  SPRING_CONFIG_IMPORT: configserver:http://configserver:8085/
  EUREKA_CLIENT_SERVICEURL_DEFAULTZONE:
http://eurekaclient:8065/eureka/

reservations:
  image: jzapana/reservations:latest
  mem_limit: 800m
  ports:
    - "8082:8082"
  networks:
    - jzapana-network
  depends_on:
    - configserver
    - eurekaclient
  deploy:
    restart_policy:
      condition: on-failure
      delay: 5s
      max_attempts: 3
      window: 120s
  environment:
    SPRING_PROFILES_ACTIVE: default
    SPRING_CONFIG_IMPORT: configserver:http://configserver:8085/
    EUREKA_CLIENT_SERVICEURL_DEFAULTZONE:
http://eurekaclient:8065/eureka/

networks:
  jzapana-network:
```

Levantar contenedores de imágenes

`docker compose up`

con `docker ps` puedo ver los contenedores creados.