

# Externalización de Configuración

Bootcamp Java Microservicios

## Contenido

1.	Herramientas Necesarias .....	2
2.	Caso de Uso .....	2
3.	Creando el servidor de configuración .....	2
4.	Habilitar servidor de configuración .....	3
5.	Modificar archivo properties del servidor de configuración .....	4
6.	Subiendo properties del microservicio customer al repositorio GIT .....	4
7.	Probando microservicio de configuraciones.....	6
8.	Conectando microservicio customer al servidor de configuraciones.....	6
9.	Refrescar en automático cambios desde el servidor .....	10

## 1. Herramientas Necesarias

- ✓ JDK Java 11
- ✓ IDE Java (IntelliJ IDEA 2022.1.2 Community, Spring Tool Suite).
- ✓ Postman <https://www.postman.com/downloads/>
- ✓ Lombok <https://projectlombok.org/>
- ✓ GIT <https://git-scm.com>
- ✓ Cliente GIT (Opcional) <https://tortoisegit.org/>
- ✓ Repositorio en Github.
- ✓ Microservicio desarrollado previamente.

## 2. Caso de Uso

Se utilizará el mismo caso de uso del proyecto semanal

## 3. Creando el servidor de configuración

- ✓ Ingresar a la siguiente dirección (<https://start.spring.io/>)
- ✓ Ingresar los siguientes datos

Project	Language
<input checked="" type="radio"/> Maven Project <input type="radio"/> Gradle Project	<input checked="" type="radio"/> Java <input type="radio"/> Kotlin <input type="radio"/> Groovy
<b>Spring Boot</b>	
<input type="radio"/> 3.0.0 (SNAPSHOT) <input type="radio"/> 3.0.0 (M5) <input type="radio"/> 2.7.5 (SNAPSHOT) <input checked="" type="radio"/> 2.7.4	
<input type="radio"/> 2.6.13 (SNAPSHOT) <input type="radio"/> 2.6.12	
<b>Project Metadata</b>	
Group	com.bootcamp.java
Artifact	configServer
Name	configServer
Description	Microservices for externalize configuration
Package name	com.bootcamp.java.configserver
Packaging	<input checked="" type="radio"/> Jar <input type="radio"/> War
Java	<input type="radio"/> 19 <input type="radio"/> 17 <input checked="" type="radio"/> 11 <input type="radio"/> 8

- ✓ Seleccionar las siguientes dependencias

### Dependencies

ADD DEPENDENCIES... CTRL + B

#### Spring Boot DevTools **DEVELOPER TOOLS**

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

#### Config Server **SPRING CLOUD CONFIG**

Central management for configuration via Git, SVN, or HashiCorp Vault.

- ✓ Hacer click en el botón Generate que aparece en la web.

GENERATE CTRL + G

EXPLORE CTRL + SPACE

SHARE...

- ✓ Abrir desde el IDE de su preferencia

## 4. Habilitar servidor de configuración

- ✓ Modificar clase ConfigServerApplication, añadir etiqueta @EnableConfigServer

```
@EnableConfigServer
@SpringBootApplication
public class ConfigServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(ConfigServerApplication.class, args);
    }

}
```

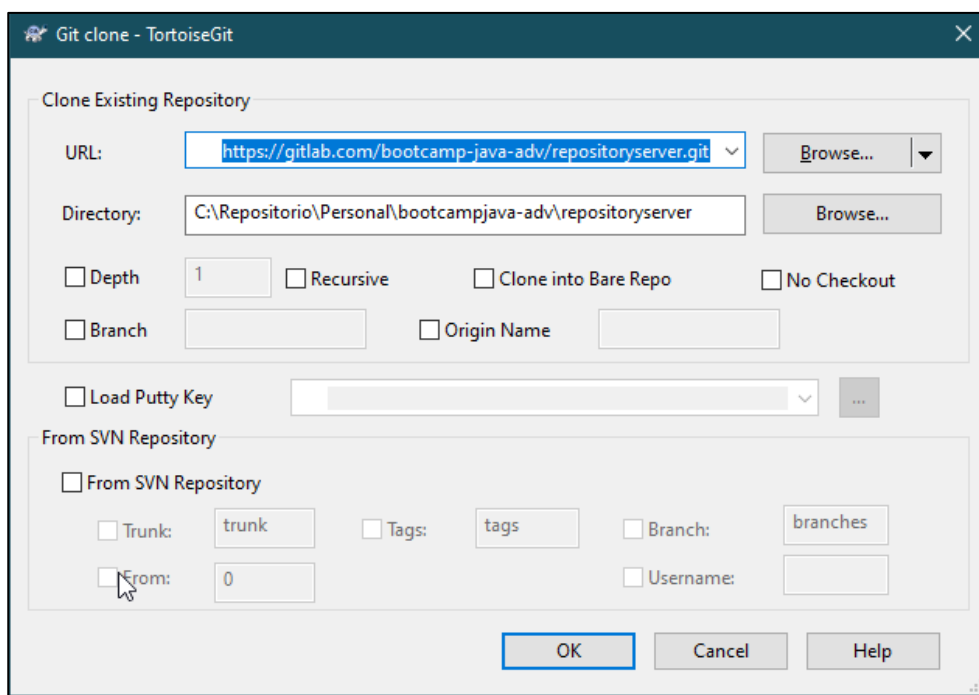
## 5. Modificar archivo properties del servidor de configuración

- ✓ Modificar archivo application.properties y agregar las siguientes propiedades
- ✓ Modificar el puerto por el puerto que se desee actualizar
- ✓ Modificar la propiedad git uri por la URL del repositorio GIT a actualizar.


```
spring.application.name=service-config  
server.port=9080  
spring.cloud.config.server.git.clone-on-start=false  
spring.cloud.config.server.git.default-label=main  
spring.cloud.config.server.git.uri=https://gitlab.com/bootcamp-java-  
adv/repositoryserver.git  
spring.cloud.config.health.enabled=false
```

## 6. Subiendo properties del microservicio customer al repositorio GIT

- ✓ Clonar repositorio GIT donde se guardarán los archivos de configuración externalizados.



- ✓ Crear archivo service-customer.properties
- ✓ Copiar contenido del properties del microservicio customer al archivo creado.
- ✓ Darle commit & push



**repositoryServer**
Project ID: 40061828

🔔
☆ Star 0
🍴 Fork 0

1 Commit
 1 Branch
 0 Tags
 72 KB Project Storage

main
repositoryserver /
+

Find file
Web IDE
Download
Clone


**Add customer properties**
b85375bc

Angelo Angulo authored just now

README
Add LICENSE
Add CHANGELOG
Add CONTRIBUTING
Enable Auto DevOps
Add Kubernetes cluster

Set up CI/CD
Configure Integrations

Name	Last commit	Last update
README.md	Initial commit	17 minutes ago
service-customer.properties	Add customer properties	just now

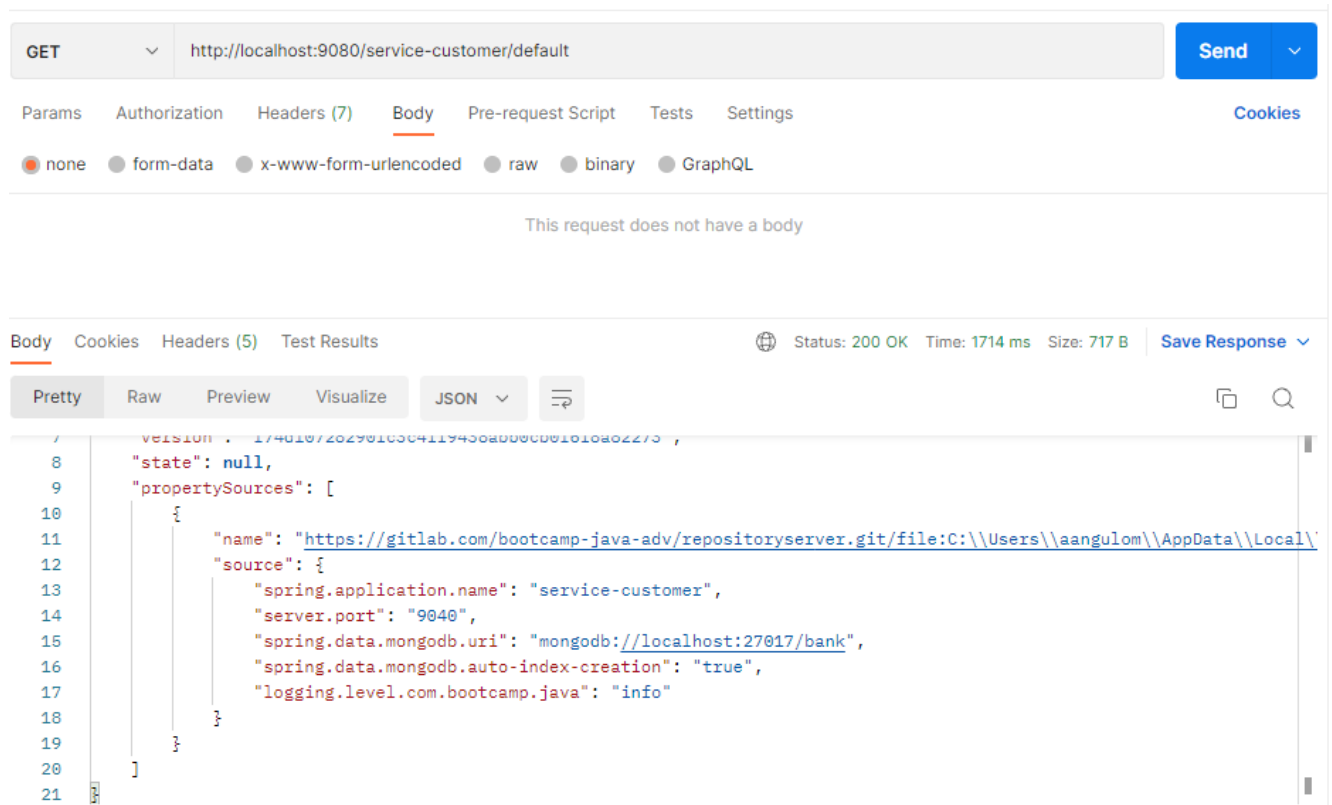

**service-customer.properties**
216 bytes

```

1  spring.application.name=service-customer
2  server.port=9040
3
4  #Spring Data
5  spring.data.mongodb.uri=mongodb://localhost:27017/bank
6  spring.data.mongodb.auto-index-creation=true
7
8  #Log
9  logging.level.com.bootcamp.java=info
10
11
  
```

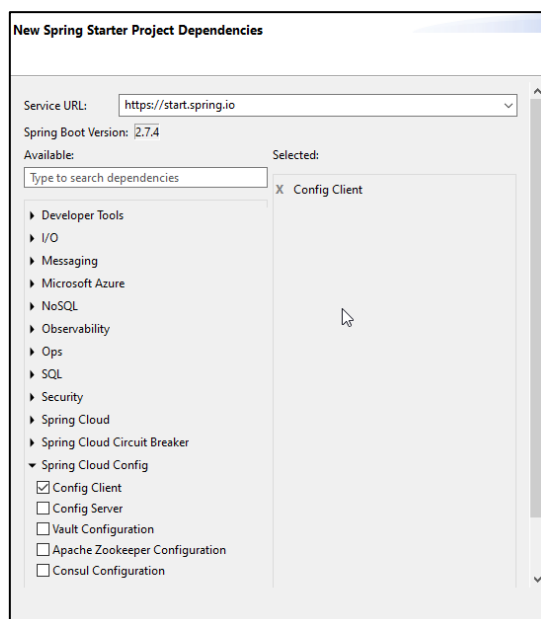
## 7. Probando microservicio de configuraciones

- ✓ Probar desde el Postman con la siguiente URL `http://localhost:8888/{service}/{branch}`



## 8. Conectando microservicio customer al servidor de configuraciones (vía Spring Starters)

- ✓ Añadir el siguiente starter a microservicio existente



## 9. Conectando microservicio customer al servidor de configuraciones (vía POM)

- ✓ Modificar el archivo POM del microservicio customer y agregar dependencias de Spring Cloud Config.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.4</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.bootcamp.java</groupId>
  <artifactId>customer</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>customer</name>
  <description>Microservice for CRUD Customer</description>
  <properties>
    <java.version>11</java.version>
    <org.mapstruct.version>1.5.2.Final</org.mapstruct.version>
    <lombok.version>1.18.24</lombok.version>
    <spring-cloud.version>2021.0.4</spring-cloud.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-mongodb-reactive</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-webflux</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <scope>runtime</scope>
      <optional>true</optional>
    </dependency>
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <optional>true</optional>
    </dependency>
  </dependencies>
```



```

        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>io.projectreactor</groupId>
        <artifactId>reactor-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
    <dependency>
        <groupId>org.mapstruct</groupId>
        <artifactId>mapstruct</artifactId>
        <version>${org.mapstruct.version}</version>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.mapstruct</groupId>
        <artifactId>mapstruct-processor</artifactId>
        <version>${org.mapstruct.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springdoc</groupId>
        <artifactId>springdoc-openapi-webflux-core</artifactId>
        <version>1.4.3</version>
    </dependency>
    <dependency>
        <groupId>org.springdoc</groupId>
        <artifactId>springdoc-openapi-webflux-ui</artifactId>
        <version>1.4.3</version>
    </dependency>
    <dependency>
        <groupId>io.github.classgraph</groupId>
        <artifactId>classgraph</artifactId>
        <version>4.8.139</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-config</artifactId>
    </dependency>
</dependencies>

<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>
            </configuration>
        </plugin>
    </plugins>
</build>

```

```

        </configuration>
    </plugin>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.5.1</version>
        <configuration>
            <source>${java.version}</source>
            <target>${java.version}</target>
            <annotationProcessorPaths>
                <path>
                    <groupId>org.projectlombok</groupId>
                    <artifactId>lombok</artifactId>
                    <version>${lombok.version}</version>
                </path>
                <path>
                    <groupId>org.mapstruct</groupId>
                    <artifactId>mapstruct-processor</artifactId>
                    <version>${org.mapstruct.version}</version>
                </path>
                <path>
                    <groupId>org.projectlombok</groupId>
                    <artifactId>lombok-mapstruct-binding</artifactId>
                    <version>0.2.0</version>
                </path>
            </annotationProcessorPaths>
            <compilerArgs>
                <compilerArg>
                    -Amapstruct.verbose=true
                </compilerArg>
            </compilerArgs>
        </configuration>
    </plugin>
</plugins>
</build>
<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>
</project>

```

- ✓ Modificar archivo application.properties del microservicio customer

```

spring.application.name=service-customer
server.port=9040
management.endpoints.web.exposure.include=*

#Spring Data
spring.data.mongodb.uri=mongodb://localhost:27017/bank

```

```
spring.data.mongodb.auto-index-creation=true

#Log
logging.level.com.bootcamp.java=Info
message.demo=leido desde local

#Spring Cloud Config
spring.config.import=optional:configserver:http://localhost:8888
```

## 10. Refrescar en automático cambios desde el servidor

- ✓ Modificar Controller (CustomerController.java) y añadir etiqueta @RefreshScope

```
@Slf4j
@RequiredArgsConstructor
@RestController
@RequestMapping("/v1/customer")
@RefreshScope
public class CustomerController {

    @Value("${spring.application.name}")
    String name;

    @Value("${server.port}")
    String port;

    @Value("${message.demo}")
    private String demoString;

    @Autowired
    private CustomerCmdService customerService;

    @Autowired
    private CustomerQueryService customerQueryService;

    @Autowired
    private CustomerMapper customerMapper;

    @GetMapping
    public Mono<ResponseEntity<Flux<CustomerModel>>> getAll(){
        log.info("getAll executed");
        log.info(demoString);
        return Mono.just(ResponseEntity.ok()
                                .body(customerQueryService.findAll()
                                    .map(customer -> customerMapper.entityToModel(customer)))));
    }

    @GetMapping("/{id}")
    public Mono<ResponseEntity<CustomerModel>> getById(@PathVariable String id){
        log.info("getById executed {}", id);
        Mono<Customer> response = customerQueryService.findById(id);
        return response
            .map(customer -> customerMapper.entityToModel(customer))
    }
}
```

```

        .map(ResponseEntity::ok)
        .defaultIfEmpty(ResponseEntity.notFound().build());
    }

    @PostMapping
    public Mono<ResponseEntity<CustomerModel>> create(@Valid @RequestBody CustomerModel
request){
        Log.info("create executed {}", request);
        return customerService.create(customerMapper.modelToEntity(request))
            .map(customer -> customerMapper.entityToModel(customer))
            .flatMap(c ->
Mono.just(ResponseEntity.created(URI.create(String.format("http://%s:%s/%s/%s", name,
port, "customer", c.getId()))))
                .body(c)))
            .defaultIfEmpty(ResponseEntity.notFound().build());
    }

    @PutMapping("/{id}")
    public Mono<ResponseEntity<CustomerModel>> updateById(@PathVariable String id, @Valid
@RequestBody CustomerModel request){
        Log.info("updateById executed {}:{}", id, request);
        return customerService.update(id, customerMapper.modelToEntity(request))
            .map(customer -> customerMapper.entityToModel(customer))
            .flatMap(c ->
Mono.just(ResponseEntity.created(URI.create(String.format("http://%s:%s/%s/%s", name,
port, "customer", c.getId()))))
                .body(c)))
            .defaultIfEmpty(ResponseEntity.badRequest().build());
    }

    @DeleteMapping("/{id}")
    public Mono<ResponseEntity<Void>> deleteById(@PathVariable String id){
        Log.info("deleteById executed {}", id);
        return customerService.delete(id)
            .map(r -> ResponseEntity.ok().<Void>build())
            .defaultIfEmpty(ResponseEntity.notFound().build());
    }
}

```