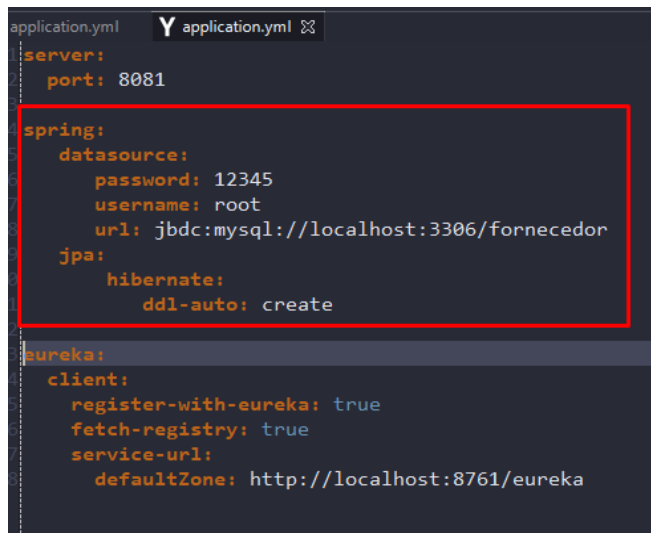


## Spring Cloud Config

- Criação da configuração do micro serviço Floricultura usando Spring Boot e o projeto anteriormente utilizado:

<https://github.com/annie-bot/spring-cloud-gateway-example>

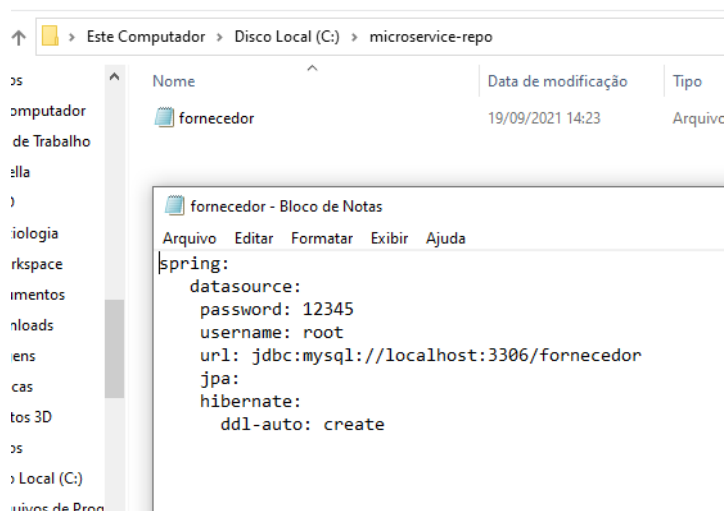
No Eclipse, dentro da configuração do "fornecedor", no "application.yml" dele, temos a configuração de Eureka, mas temos a configuração de banco de dados também. Para trabalhar com o **Spring Config Server**, vamos tirar essa configuração de banco de dados e vamos extraí-la para dentro de um arquivo no servidor, no "file system". Dentro do disco local da máquina, do "C:", vamos criar uma pasta "**microservice-repo**". Essa pasta vai ser a de repositório de configurações. E eu vou criar um arquivo de configuração do fornecedor com essas informações de banco de dados do "application.yml". Basta criar um arquivo no bloco de notas, colar as anotações e salvar como .yml

A screenshot of the Eclipse IDE showing the 'application.yml' file. The file content is as follows:

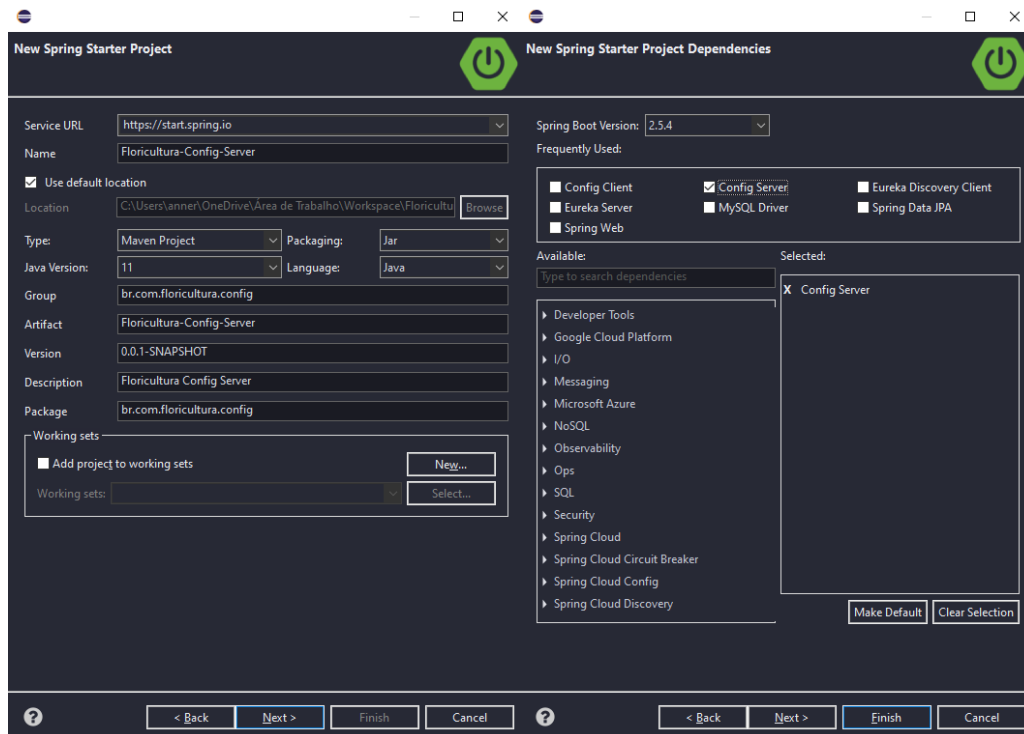
```
server:
  port: 8081

spring:
  datasource:
    password: 12345
    username: root
    url: jdbc:mysql://localhost:3306/fornecedor
  jpa:
    hibernate:
      ddl-auto: create

eureka:
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://localhost:8761/eureka
```

The 'spring' section is highlighted with a red rectangle.

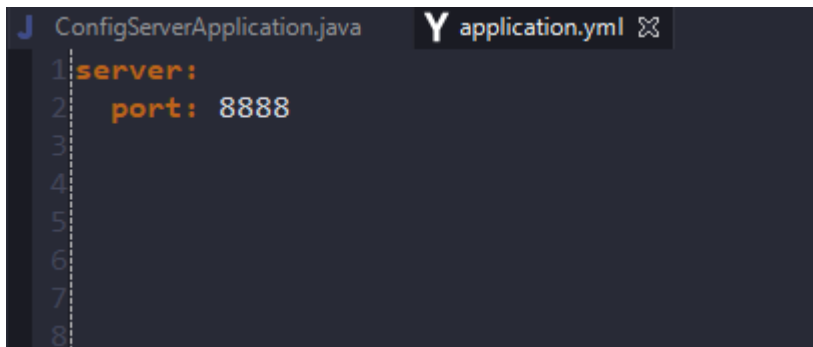
Então vamos criar um novo projeto e selecionar a dependência Config Server.



A primeira coisa é habilitar o "Config Server" nesse projeto, então `@EnableConfigServer` na classe `Application`.

```
1 package br.com.floricultura.config;
2
3 import org.springframework.boot.SpringApplication;
4
5
6
7 @SpringBootApplication
8 @EnableConfigServer
9 public class ConfigServerApplication {
10
11     public static void main(String[] args) {
12         SpringApplication.run(ConfigServerApplication.class, args);
13     }
14
15 }
16
```

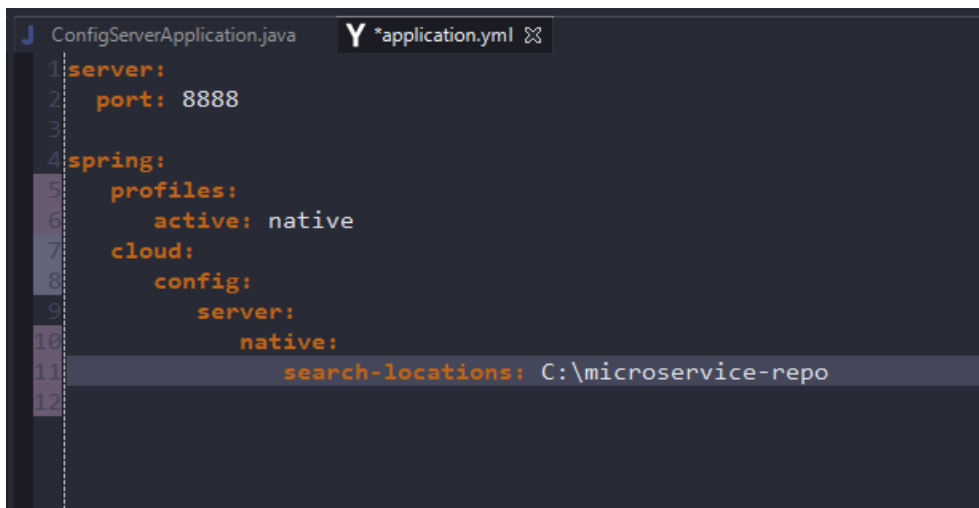
Esse passo habilita o servidor de configuração, ele já está disponível, só precisamos configurar ele. Então existem duas configurações que temos que fazer no "application.yml". A primeira é o server port na porta específica: 8888.



The screenshot shows a code editor with two tabs: 'ConfigServerApplication.java' and 'application.yml'. The 'application.yml' tab is active, displaying the following configuration:

```
1 server:
2   port: 8888
3
4
5
6
7
8
```

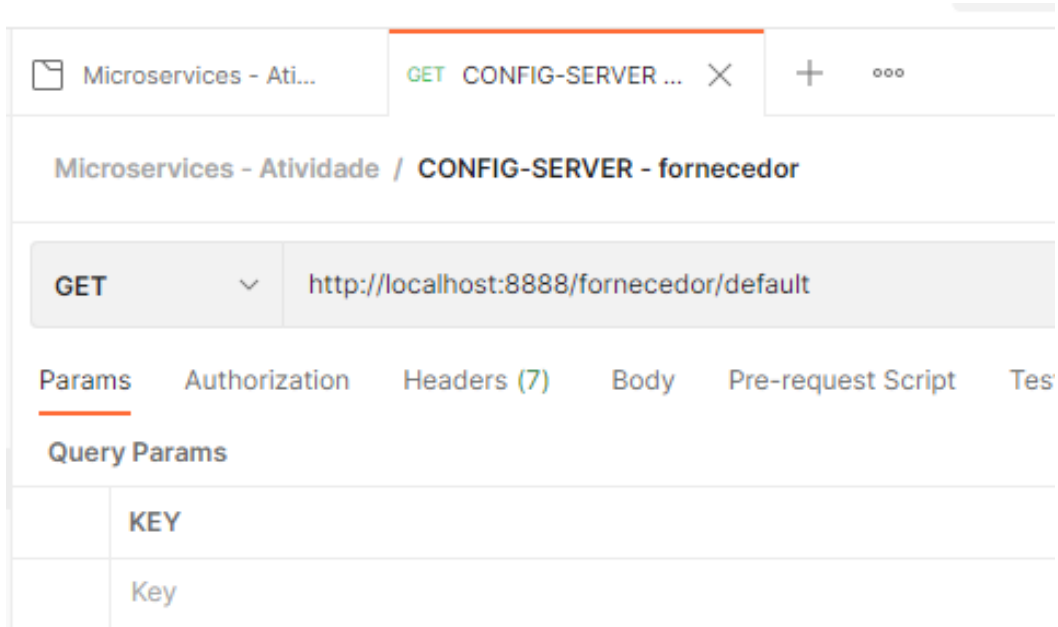
A próxima configuração é do tipo de **profile**, ele pede para configurarmos um profile, indicando que ele vai buscar as configurações dentro do file system.



The screenshot shows the same code editor with the 'application.yml' tab active. The configuration is now more complete:

```
1 server:
2   port: 8888
3
4 spring:
5   profiles:
6     active: native
7   cloud:
8     config:
9       server:
10        native:
11          search-locations: C:\microservice-repo
12
```

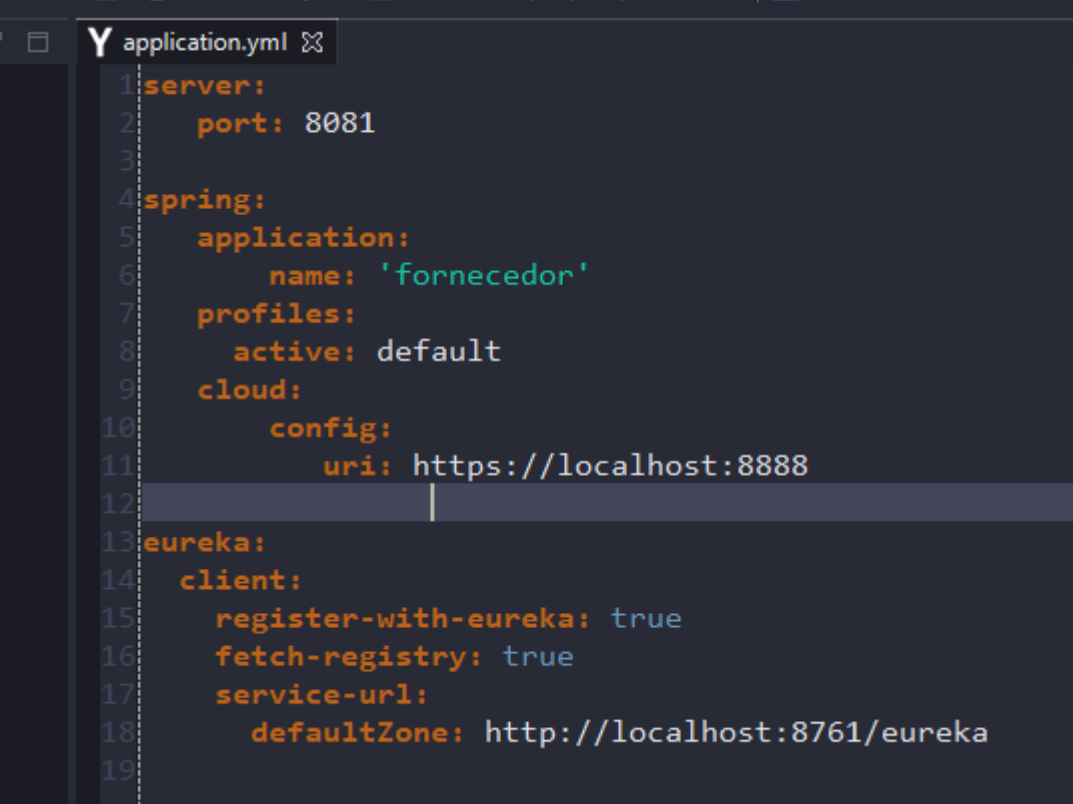
Então, no Postman, vamos criar uma requisição chamada de "CONFIG-SERVER - fornecedor" do tipo "GET" para "http://localhost:8888/fornecedor/default". Então quando você tem o nome do arquivo, na pasta do computador, sem extensão nenhuma, ele considera que aquele arquivo .yaml é o arquivo default de configuração.



Agora o Configuration Server já está funcionando. Nossa segunda etapa é ensinar o fornecedor a acessar o Configuration Server.

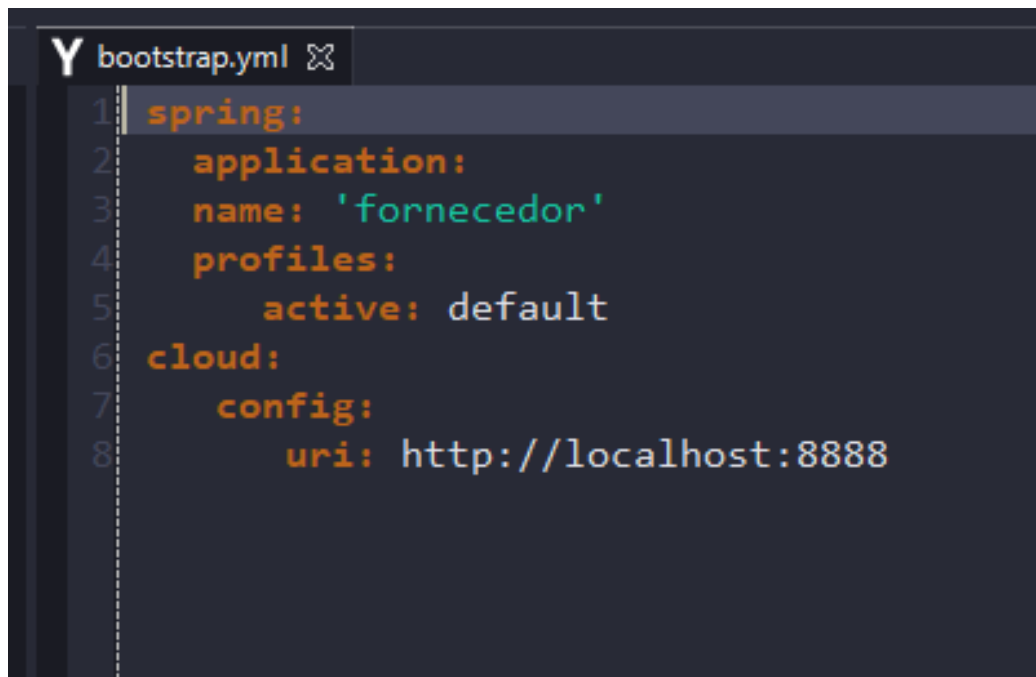
- Acessando o Configuration Server

Na application.yml do fornecedor, adicionaremos as configurações que definimos no Postman, para assim que haja uma requisição do sistema, o fornecedor consiga acessar as informações do config.



```
1 server:
2   port: 8081
3
4 spring:
5   application:
6     name: 'fornecedor'
7   profiles:
8     active: default
9   cloud:
10    config:
11      uri: https://localhost:8888
12
13 eureka:
14   client:
15     register-with-eureka: true
16     fetch-registry: true
17     service-url:
18       defaultZone: http://localhost:8761/eureka
19
```

Como estamos utilizando outras dependências, para que não haja nenhum choque de trânsito de informações, vamos pegar essa configuração que acabamos de colocar no .yml e adicioná-las em um momento anterior ao início da aplicação. Então para executar isso, existe um contexto anterior, do Spring, chamado de bootstrap, que lê um arquivo chamado bootstrap.yml. Então vamos criar um novo arquivo no package fornecedor e pegar essa configuração anterior e colocar nele.

A screenshot of a code editor with a dark theme. The editor shows a file named 'bootstrap.yml' with a search icon. The code is written in a YAML format with orange and green syntax highlighting. Line numbers 1 through 8 are visible on the left side of the editor.

```
1 spring:
2   application:
3     name: 'fornecedor'
4   profiles:
5     active: default
6 cloud:
7   config:
8     uri: http://localhost:8888
```

o que é o bootstrap? Bootstrap é um arquivo que vai ser lido em um contexto anterior ao contexto da nossa aplicação. Então quando a nossa aplicação subir e for ler esse arquivo de configuração "application.yml", que tem o eureka:, o server:, port:, o "bootstrap.yml" já foi lido e processado, então as configurações que vieram do Spring Configuration Server já estão disponíveis. É exatamente neste contexto do "application.yml" que o hibernate vai buscar as propriedades para acessar o banco de dados.

Nesse momento, as configurações já foram carregadas do Configuration Server e já estão disponíveis para a nossa aplicação.