

Configuração

Índice

- 01** [Configurando valores no Git](#)
- 02** [Configurações do servidor de configuração](#)
- 03** [Configurando clientes do servidor de configuração](#)



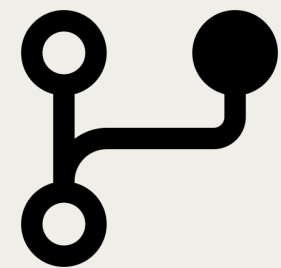
01

Configurando valores no Git

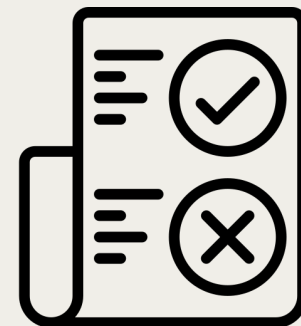
Contexto de valores no Git

As configurações que temos de nossos microserviços serão especificadas como arquivos YML dentro de um repositório GitHub. Haverá um arquivo YML para cada microserviço que precisamos configurar externamente.

Usar o Git é uma prática padrão porque, assim como o código, permite:



**Controle de
versões.**



**Defina regras de
aprovação e rejeição
automaticamente.**



**Fácil de analisar,
aplicar e/ou reverter
alterações.**



**Rastreabilidade das
mudanças.**


No Git os seguintes arquivos são exibidos:





- **application.yml:** contém configurações gerais para todos os serviços.
- **dh-account-service.yml:** contém as configurações para o microserviço chamado “dh-account-service”.
- **dh-transaction-service.yml:** contém as configurações para o microserviço chamado “dh-transaction-service”

main ▾

1 branch

0 tags

 FernandoMelloDH properties cloud

 README.md	Initial commit
 application.yml	properties cloud
 dh-account-service.yml	properties cloud
 dh-transaction-service.yml	properties cloud

Neste exemplo, estamos adicionando as propriedades:

- **server.port:** porta de escuta do microserviço “dh-account-service”.
- **message:** propriedade personalizada com um valor.



FernandoMelloDH Update dh-account-service.yml

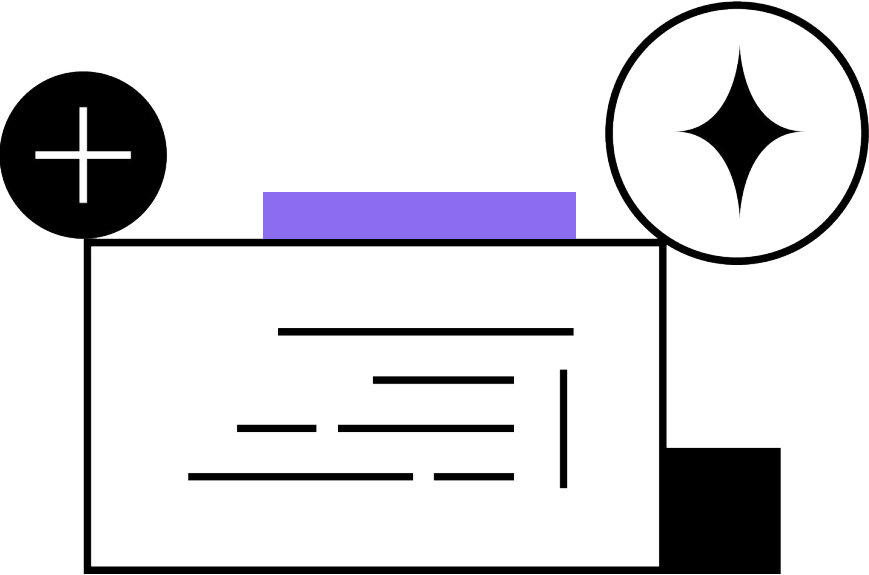
1 contributor

4 lines (3 sloc) | 89 Bytes

```
1  server:
2    port: ${PORT:8889}
3
4  message: configuração padrão com spring cloud config!
```

02

Configurações do servidor de configuração



Agora que temos nosso repositório Git com as configurações que consideramos pertinentes, devemos criar nosso servidor Cloud Config para que ele pegue a base do Git e disponibilize para serviços REST para os microsserviços que dele precisarem.
Os passos que teremos que fazer são:

01

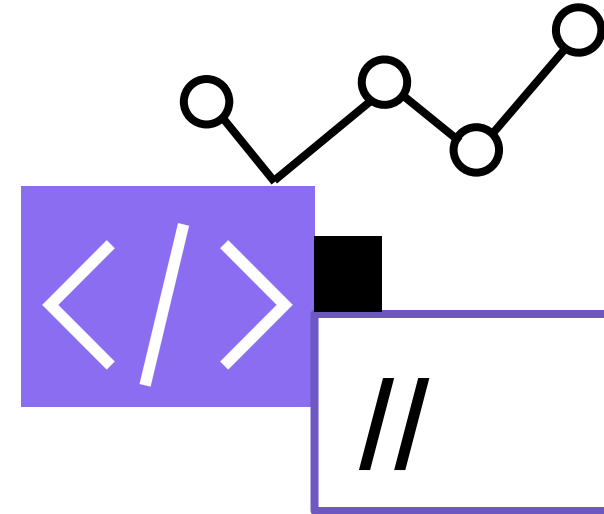
Adicione a dependência de
SpringCloudConfig

02

Configurar o microsserviço
como servidor de configuração

03

Configurar o servidor para
pegar os valores do Git

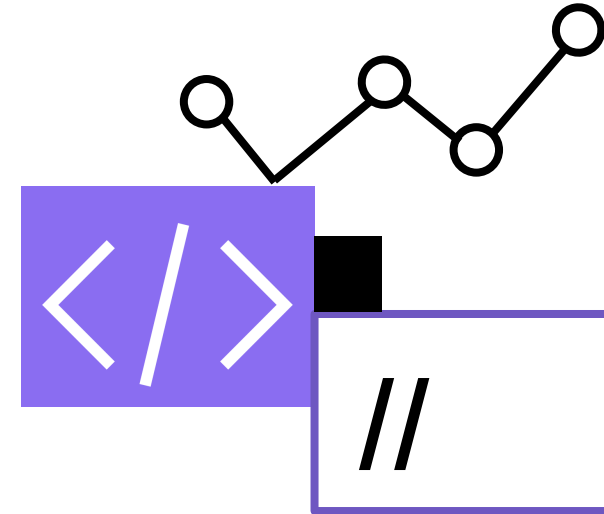


1) Adicione a dependência de Spring Cloud Config

Para isso, devemos adicionar em nosso microserviço com Spring Boot a seguinte dependência dentro do **pom.xml**:

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

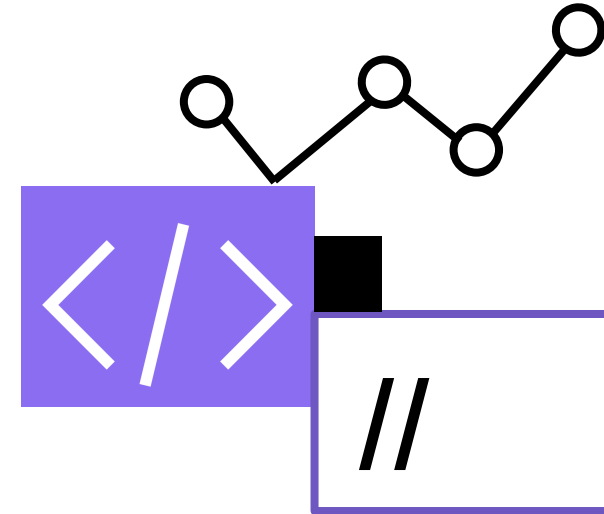
{JSON}
JavaScript Object Notation



2) Configurar o microserviço como um servidor de configuração

Em nossa classe de inicialização do Spring Boot, devemos especificar que esta aplicação será um servidor de configuração por anotação **@EnableConfigServer**:

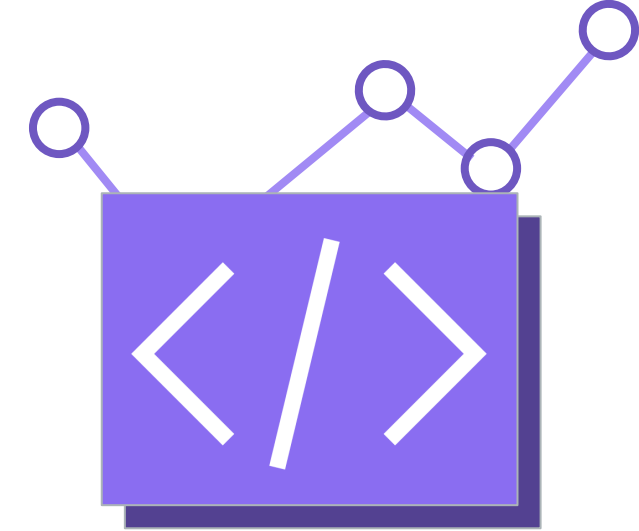
```
@EnableConfigServer
@SpringBootApplication
public class ConfigServerApplication {
    public static void main(String[] args)
    {SpringApplication.run(ConfigServerApplication.class, args);}
}
```



3) Configure o servidor para pegar valores do Git

Em nosso aplicativo YML, precisamos especificar a fonte de configuração Git para o servidor de configurações dentro da chave: `spring.cloud.config.server.git.url`.

```
server:
  port: ${PORT:8888}
spring:
  application:
    name: config-server
  cloud:
    config:
      server:
        git:
          url: https://github.com/FernandoMelloDH/spring-cloud-example
```



Validação

Após estes três passos, podemos validar se configuramos corretamente o servidor consultando algumas das configurações dos serviços declarados via HTTP.

A consulta será realizada sob o seguinte padrão:

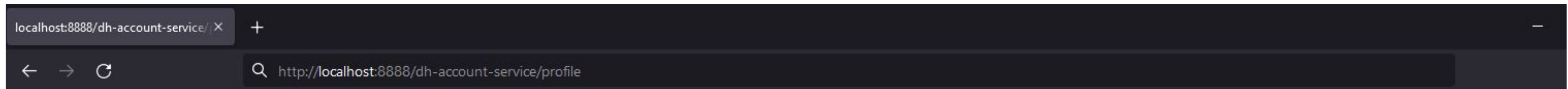
`http://{url-servidor-config}/{nome-servico}/{nome-perfil}`

É o endereço `{url-server-config}` do servidor que fornecerá configurações em SpringCloudConfig

É o nome do microserviço onde é consultada a configuração definida.

Define o nome do perfil de configuração a ser consultado. Isso será usado com o nome do serviço para determinar as configurações a se utilizar (mais tarde veremos isso em profundidade).

Se colocarmos em nosso navegador a URL <http://localhost:8888/dh-account-service/profile>, veremos o seguinte resultado:



```
{"name":"dh-account-service", "profiles": ["profile"],"label":null,"version":null,"state":null,"propertySources":
[{"name": "https://github.com/FernandoMelloDH/spring-cloud-example/dh-account-service.yml","source":
{"server.port":"${PORT:8889}","message":"configuração padrão com spring cloud config!"}},
{"name":"https://github.com/FernandoMelloDH/spring-cloud-example/application.yml","source":{"global-
message":"Configuration server is alive!"}}]}
```

03

Configuração do cliente do servidor de configurações

Para criar nosso cliente de configuração do Spring Cloud Config, devemos:

01

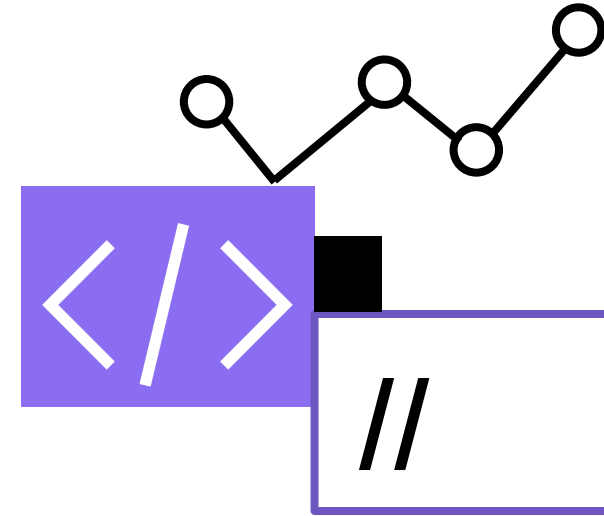
Adicionar a dependência do
Spring Cloud Config

02

Especifique o caminho do
servidor de configuração a ser
usado

03

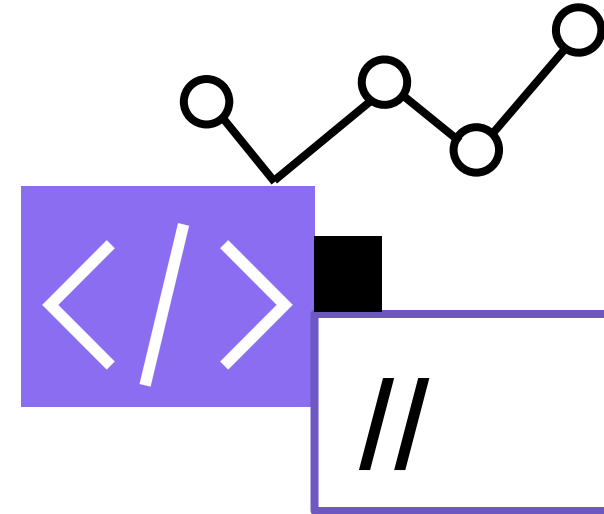
Use as configurações obtidas



1) Adicione a dependência de Spring Cloud Config

Para isso devemos adicionar a seguinte dependência dentro do **pom.xml**:

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

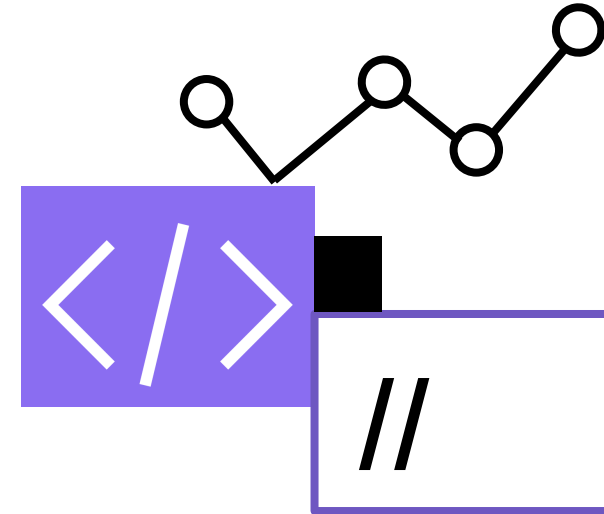



2) Especifique o caminho do servidor de configuração a ser usado

Devemos criar o arquivo de configuração **bootstrap.yml** especificando o nome do microserviço, um nome que será utilizado para obter a configuração do servidor conforme os nomes dos serviços definidos no Git, como vimos no início do uso deste framework .

Também neste YML, definiremos a URL do servidor de configuração:

```
spring:
  application:
    name: dh-conta-serviço
  cloud:
    config:
      url: ${CONFIG_SERVER:http://localhost:8888}
```



3) Use as configurações obtidas

Precisamos adicionar a dependência **spring-web** ao nosso projeto, adicionando um endpoint REST onde exibiremos as configurações que obtivemos do nosso servidor de configuração.

Em nosso **pom.xml** incorporamos:

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

Em seguida, criaremos uma classe com um endpoint injetando os valores de configuração vistos anteriormente na configuração do Git:

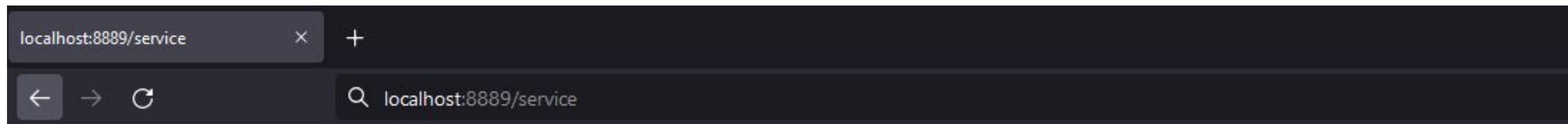
```
@RestController
class AccountService {
    @Value("${message}")
    private String message;
    @Value("${global-message}")
    private String globalMessage;

    @RequestMapping(method = RequestMethod.GET, path="service")
    public Map<String,String> message() {
        Map<String,String> response = new HashMap<>();

        response.put("message", message);
        response.put("global-message", globalMessage);

        return response;
    }
}
```

Por fim, acessamos o serviço "contas" de acordo com a porta de uso definida no Git e inserimos a URL no navegador <http://localhost:8889/service>.



```
{"global-message":"Configuration server is alive!","message":"configuração padrão com spring cloud config!"}
```

Muito obrigado!