

Arquitetura de MicroServices com Spring Cloud e Spring Boot — Parte 2



João Rafael Campos da Silva [Follow](#)

Jul 6, 2017 · 6 min read



Sumário

1. Introdução
2. Implementando o Config Server.
3. Subindo um Eureka Server e conectando-o ao Config Server.
4. Construindo um Servidor de Autorização OAuth2.
5. Implementando Serviço de Pedidos.

Implementando o Config Server

Olá pessoal, hoje vamos criar nosso Servidor de Configuração, que tem o objetivo de centralizar toda a configuração da nossa rede de Microservices em um só lugar.

Todos os arquivos de configuração do nosso sistema estarão em um repositório git e o Servidor de Configuração será o responsável por ler as informações no repositório e fornece-las às aplicações através de requests HTTP.

Gerando a aplicação no Spring Initializr

Acesse o site do Spring Initializr e preencha as configurações como na imagem. Basicamente escolhemos qual vai ser o gerenciador de dependências (*Maven*), a linguagem de programação (*Java*), as dependências necessárias para o projeto (*Web, Actuator e Config Server*) e configuramos os metadados do Maven. Por fim geramos o projeto clicando

em *Generate Project*, *Ctrl + Enter* ou *Command + Enter*. (**Atenção para a versão do Spring, estamos usando a 1.5 nesse projeto**)

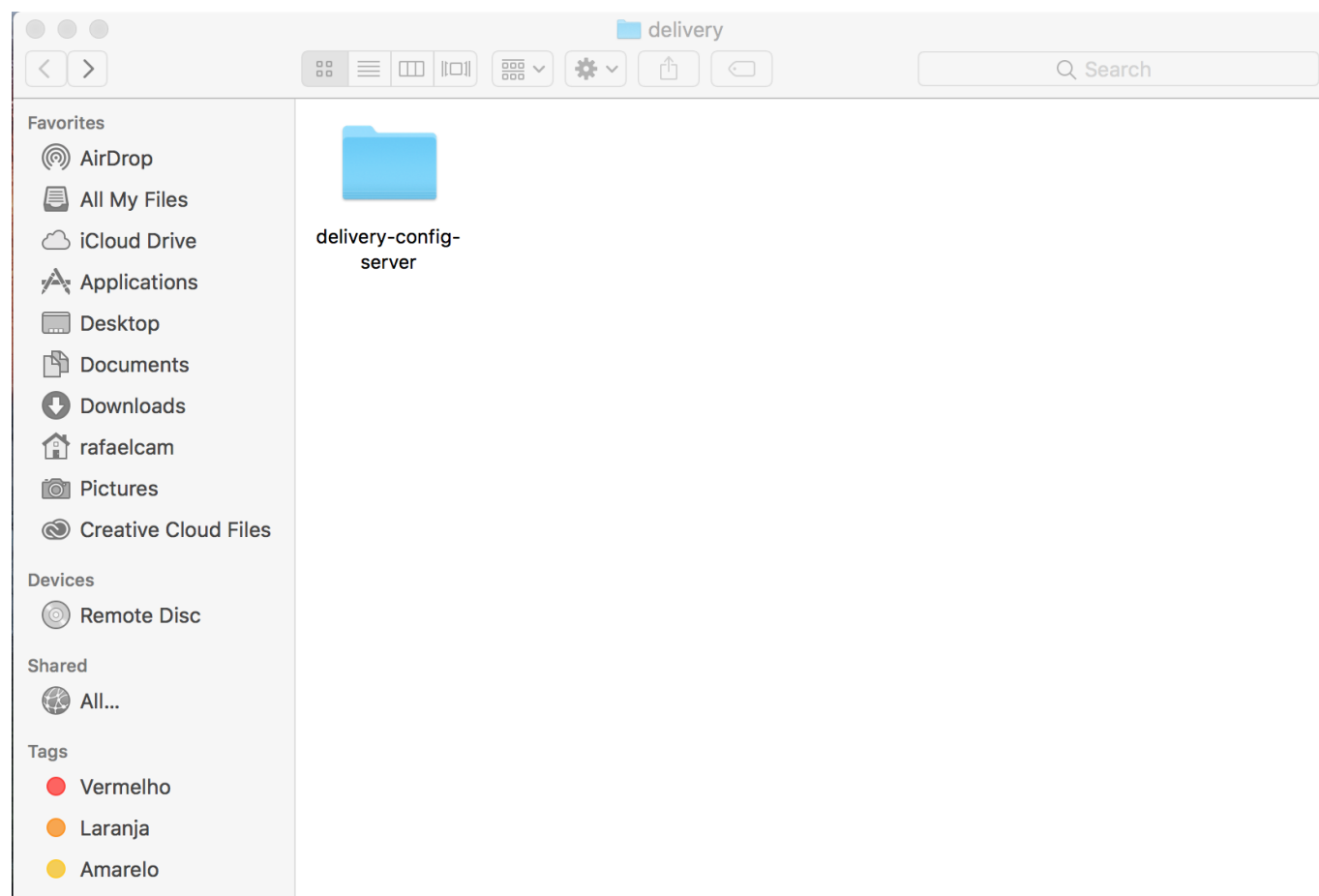
The image shows the Spring Initializr web interface. At the top, it says "SPRING INITIALIZR bootstrap your application now". Below this, there's a header "Generate a" followed by a dropdown menu set to "Maven Project", then "with" followed by a dropdown menu set to "Java", and "and Spring Boot" followed by a dropdown menu set to "1.5.4".

The main content is divided into two columns. The left column is titled "Project Metadata" and contains several input fields: "Artifact coordinates" (Group: com.coderef, Artifact: delivery-config-server, Name: delivery-config-server, Description: Config Server, Package Name: com.coderef.delivery, Packaging: Jar, Java Version: 1.8). The right column is titled "Dependencies" and contains a search bar with the text "Web, Security, JPA, Actuator, Devtools...". Below the search bar, there are three green buttons labeled "Web", "Actuator", and "Config Server".

At the bottom of the form, there is a green button labeled "Generate Project" with a small icon of a document and a plus sign.

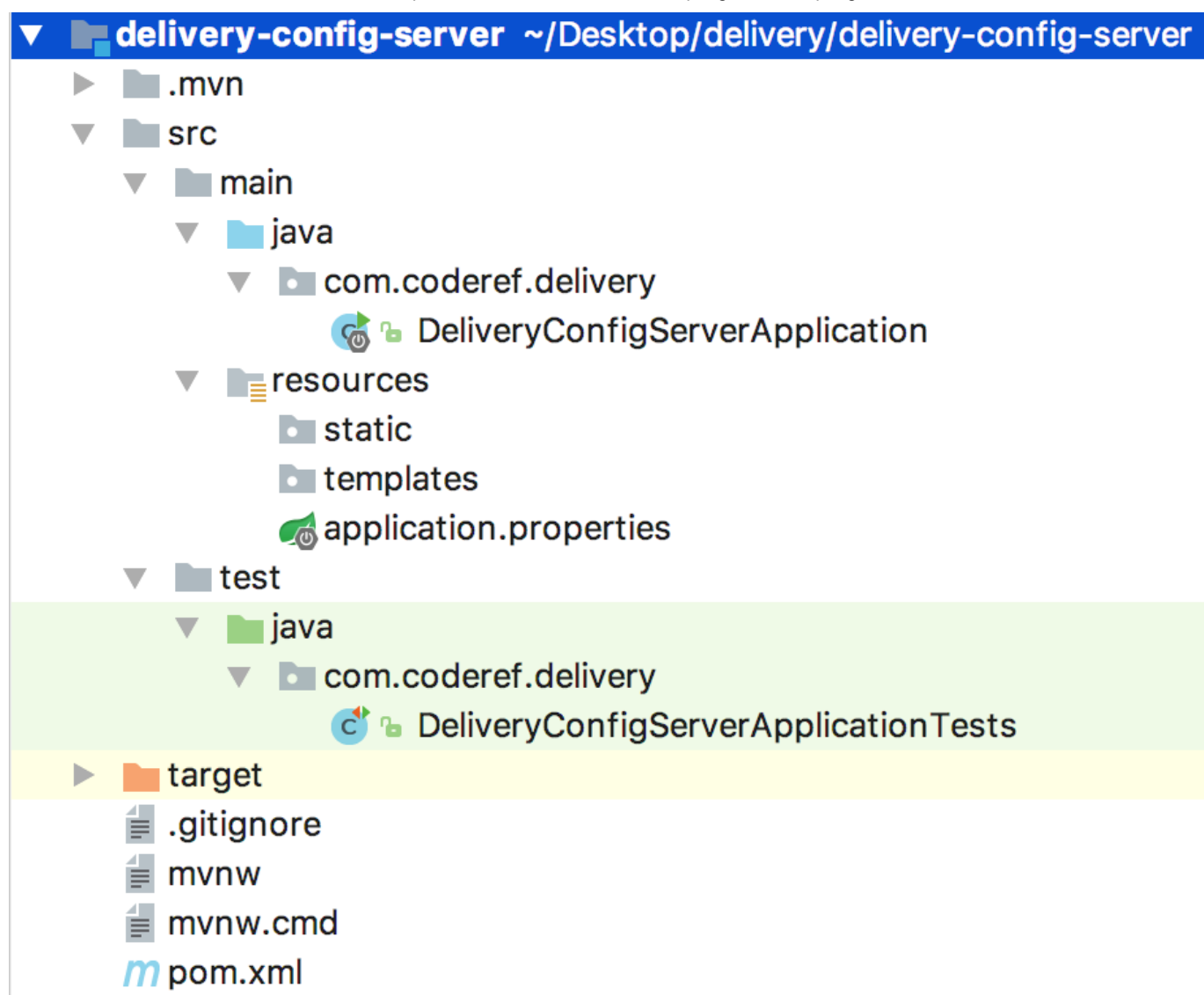
Spring Initializr preenchido para gerar o Config Server

Agora você deve ter em mãos um **delivery-config-server.zip**. Descompacte-o em uma pasta chamada **delivery**, onde ficaram todos os seus Microservices criados nesse projeto.



Pasta principal do projeto

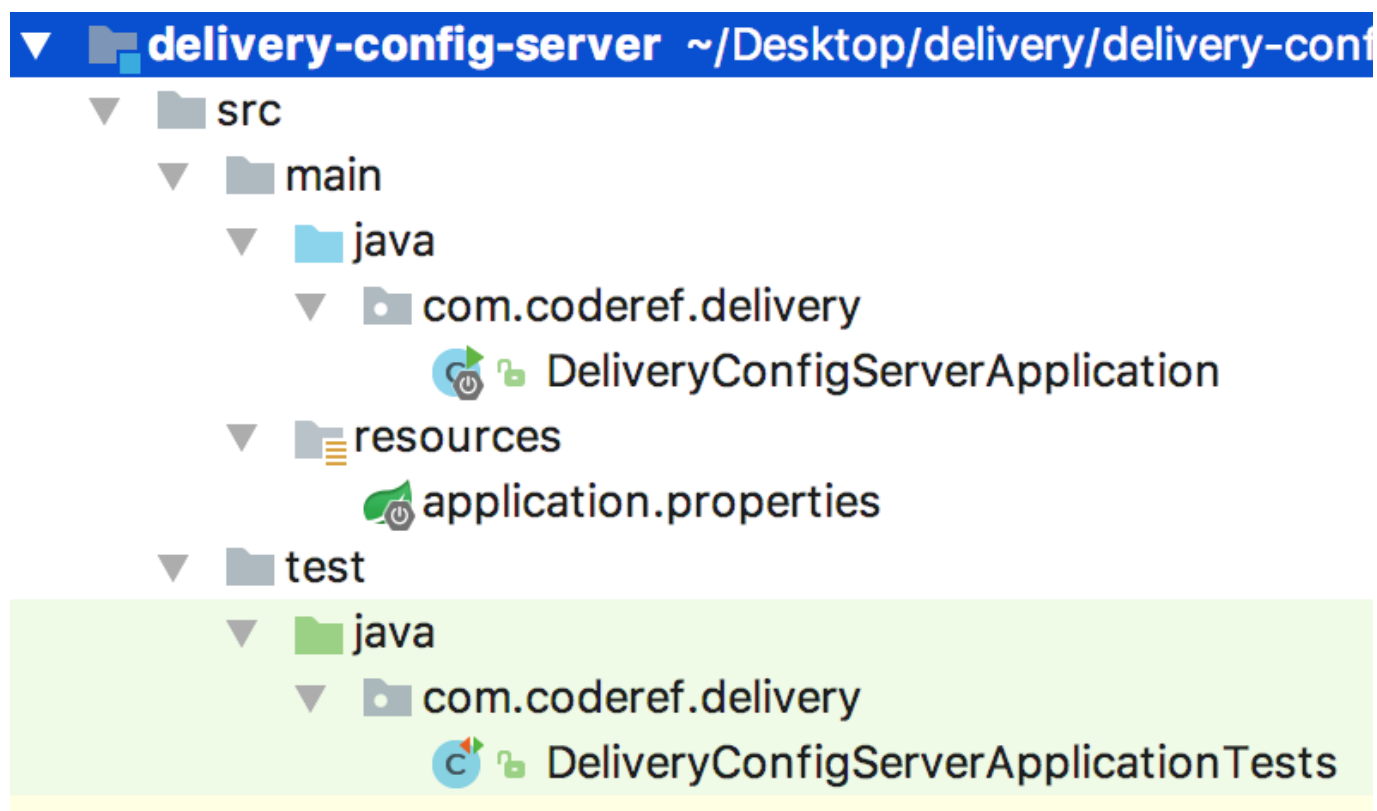
Importe o projeto em sua IDE de preferência, aqui estou usando o IntelliJ. Como podemos ver na imagem abaixo essa é a estrutura de projeto padrão gerada pelo Spring Initializr.



Arvore de diretórios do Projeto

Os arquivos listados abaixo podem ser descartados por não serem importantes para o projeto, ou seja ficamos com uma raiz de projeto tendo somente a pasta src e o pom.xml

1. Diretório .mvn (*Configurações específicas do maven*);
2. Diretório static (*Arquivos estáticos para uso em paginas web*);
3. Diretório templates (*Templates do Spring MVC*);
4. Arquivo .gitignore (*Arquivos ignorados pelo git, teremos um na pasta raiz do projeto delivery*);
5. Arquivo mvnw (*Usar versão e parâmetros específicos do maven*);
6. mvnw.cmd (*Usar versão e parâmetros específicos do maven no Windows*);





Arvore de diretórios do Projeto sem arquivos desnecessários

Agora que já estamos com o projeto importado, vou abrir os arquivos gerados e explicar em detalhes cada um.

Vamos começar pelo `pom.xml`.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-i
3      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4      <modelVersion>4.0.0</modelVersion>
5
6      <groupId>com.coderef</groupId>
7      <artifactId>delivery-config-server</artifactId>
8      <version>0.0.1-SNAPSHOT</version>
9      <packaging>jar</packaging>
10
11     <name>delivery-config-server</name>
12     <description>Config Server</description>
13
14     <parent>
15         <groupId>org.springframework.boot</groupId>
16         <artifactId>spring-boot-starter-parent</artifactId>
17         <version>1.5.4.RELEASE</version>
18         <relativePath/> <!-- lookup parent from repository -->
19     </parent>
20
21     <properties>
22         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
23         <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
24         <java.version>1.8</java.version>
25         <spring-cloud.version>Dalston.SR1</spring-cloud.version>
26     </properties>
27
28     <dependencies>
29         <dependency>
30             <groupId>org.springframework.boot</groupId>
31             <artifactId>spring-boot-starter-actuator</artifactId>
32         </dependency>
33         <dependency>
34             <groupId>org.springframework.cloud</groupId>
35             <artifactId>spring-cloud-config-server</artifactId>
36         </dependency>
37         <dependency>
38             <groupId>org.springframework.boot</groupId>
39             <artifactId>spring-boot-starter-web</artifactId>
40         </dependency>
41
42         <dependency>
43             <groupId>org.springframework.boot</groupId>
44             <artifactId>spring-boot-starter-test</artifactId>
45             <scope>test</scope>
46         </dependency>
47     </dependencies>
48
49     <dependencyManagement>
50         <dependencies>
51             <dependency>
52                 <groupId>org.springframework.cloud</groupId>
53                 <artifactId>spring-cloud-dependencies</artifactId>
54                 <version>${spring-cloud.version}</version>
55                 <type>pom</type>

```

```
56         <scope>import</scope>
57     </dependency>
58 </dependencies>
59 </dependencyManagement>
60
61 <build>
62     <plugins>
63         <plugin>
64             <groupId>org.springframework.boot</groupId>
65             <artifactId>spring-boot-maven-plugin</artifactId>
66         </plugin>
67     </plugins>
68 </build>
69
70
71 </project>
```

pom.xml hosted with ❤ by GitHub [view raw](#)

Vou abordar aqui só configurações específicas do projeto, para mais informações sobre o Maven consulte a documentação oficial aqui.

Vemos logo de cara que possuímos o `spring-boot-starter-parent` como nosso parent. Por ser um projeto spring-boot existem uma série de configurações que são previamente feitas nesse projeto fazendo com que você não se preocupe com configuração e sim com a implementação de seu produto. Todas as configurações impostas pelo Spring Boot podem ser sobrepostas estendendo a classe que a implementa ou em casos mais simples usando o `application.yml`.

Logo abaixo temos a declaração das dependências do projeto, explicarei cada uma delas abaixo:

- **spring-boot-actuator:** Se trata de um sub projeto do Spring Boot. Ele adiciona vários serviços de qualidade de produção à sua aplicação com pouco esforço de sua parte como o famoso `/health` por exemplo.
- **spring-boot-starter-web:** Essa dependência é uma das mais importantes. Usando-a você já tem um projeto totalmente configurado para trabalhar com qualquer serviço web, como fornecer recursos REST e um tomcat embedded por padrão para subir o projeto.
- **spring-cloud-config-server:** Aqui está a dependência mágica que irá transformar nosso MicroService em um Servidor de Configuração sem muito esforço adicional, apenas algumas configurações no `application.yml`.
- **spring-boot-starter-test:** Por último, e não menos importante, a dependência de testes do Spring. Ela torna nosso projeto apto para a implementação de testes unitários, de API, integração, carga entre diversos outros.

Como eu disse acima cada dependência dessa fornece tudo que o seu projeto precisa para tratar de um certo nicho de frameworks. O Spring Boot hoje é bem completo, e você consegue encontrar quase todo tipo de framework pré inicializado para utilizar em sua aplicação.

Hoje em nosso projeto, temos apenas uma classe chamada

`DeliveryConfigServerApplication.java`, e acredite se quiser, é só dela que vamos precisar para montar nosso Servidor de Configuração. Essa classe é responsável por subir a nossa aplicação usando a seguinte linha de código encapsulada dentro de um método `main`:

```
SpringApplication.run(DeliveryConfigServerApplication.class, args);
```

Para que nosso projeto seja reconhecido como um Aplicativo Spring Boot precisamos adicionar também a anotação `@SpringBootApplication`.

Nós queremos que essa aplicação seja um Servidor de Configuração, e nada mais fácil que informar isso utilizando uma anotação. Isso é possível utilizando a anotação `@EnableConfigServer`.

Veja abaixo como fica a classe completa.

```
1  package com.coderef.delivery;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.cloud.config.server.EnableConfigServer;
6
7  @SpringBootApplication
8  @EnableConfigServer
9  public class DeliveryConfigServerApplication {
10
11      public static void main(String[] args) {
12          SpringApplication.run(DeliveryConfigServerApplication.class, args);
13      }
14  }
```

DeliveryConfigServerApplication.java hosted with ❤ by GitHub

[view raw](#)

Para finalizar precisamos configurar nosso `application.yml` e o `bootstrap.yml`.

- **application.yml:** Aqui são declaradas as configurações que você gostaria de substituir no starter do Spring boot e serão usadas no seu projeto.
- **bootstrap.yml:** Esse arquivo é usado para realizar algumas configurações de inicialização do nosso parent, como o nome da

aplicação no ecossistema do Spring Cloud e a conexão com um Servidor de Configuração.

Como não há como o Servidor de Configuração utilizar ele mesmo para descrever sua configuração, ele será o único que descreveremos um `application.yml`.

Obs:.. Como optamos pelo padrão de arquivos `.yml` podemos deletar o arquivo `application.properties`.

Crie o arquivo `application.yml` no diretório `delivery-config-server/src/main/resources`.

```
1  server:
2    port: 9090
3
4  spring:
5    cloud:
6      config:
7        server:
8          git:
9            uri: https://github.com/rafaelcam/delivery-configs
```

application.yml hosted with  by GitHub [view raw](#)


Aqui configuramos a porta em que a aplicação irá subir e qual o repositório git que será usado para fornecer os arquivos que o nosso Servidor de Configuração irá disponibilizar para os outros Microservices via HTTP.

Esse repositório pode ser remoto ou local, eu estou utilizando um repositório remoto e público. Para configurar um repositório local, ou obter mais informações sobre como usar um repositório privado consulte a documentação do Spring Cloud Config.

No arquivo `bootstrap.yml` vamos informar somente o nome da aplicação. Como estamos tratando de uma aplicação distribuída com o Spring Cloud, essa informação é muito importante porque funcionará como um DNS quando uma instância dessa aplicação for registrada no Service Registry. Os outros MicroServices também consultarão o Service Registry utilizando este nome, pois o host da instância pode ser dinâmico.

Crie o arquivo `bootstrap.yml` no diretório `delivery-config-server/src/main/resources`.

```
1  spring:
2    application:
3      name: educafacil-config-server
```

bootstrap.yml hosted with  by GitHub [view raw](#)

OK! Nosso Servidor de Configuração está pronto, agora vamos realizar um teste simples para ver se tudo está funcionando corretamente.

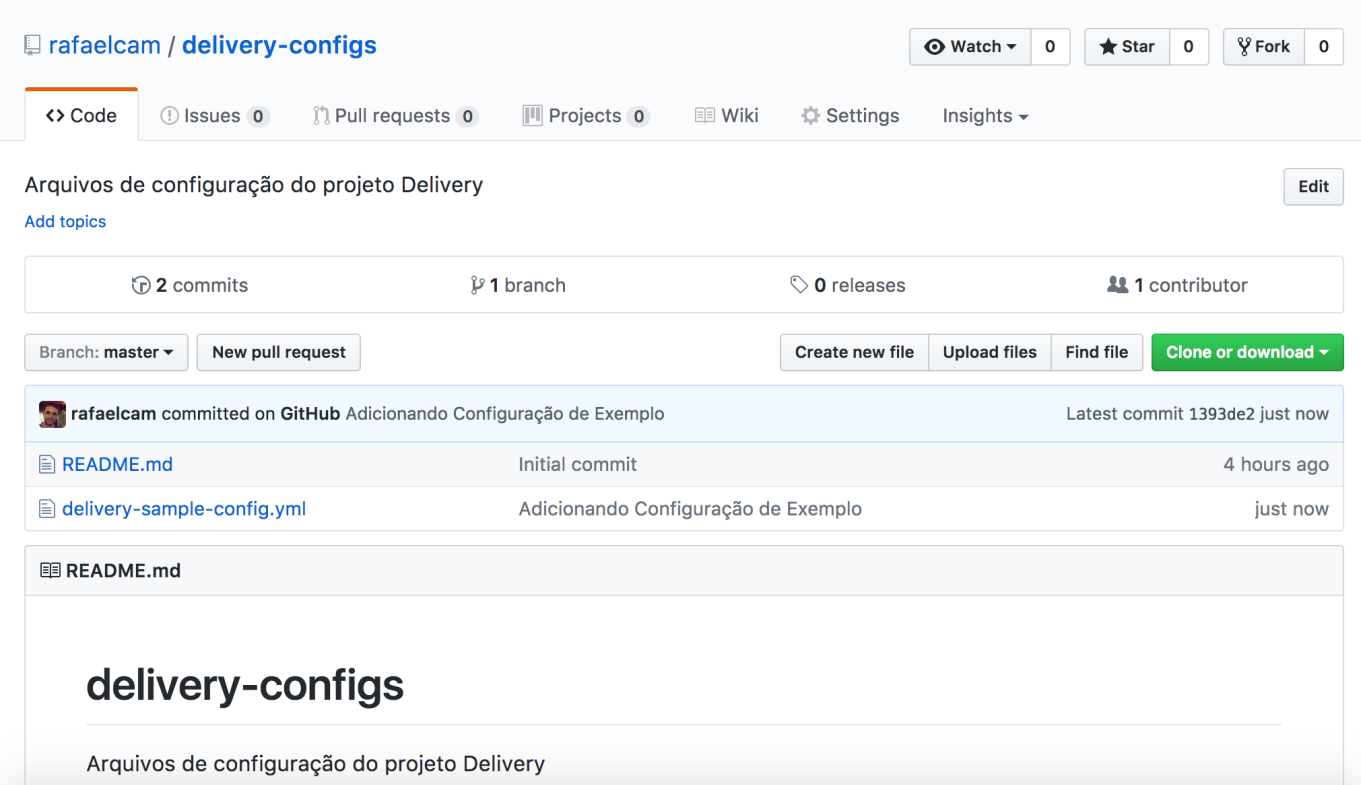
Antes de subir a aplicação vamos adicionar um arquivo de exemplo no repositório de configurações chamado `sample-config-app.yml` com uma propriedade qualquer.

```
1  delivery:
2    maxOrders: 10
3    minOrders: 1
```

delivery-sample-config.yml hosted with ❤ by GitHub

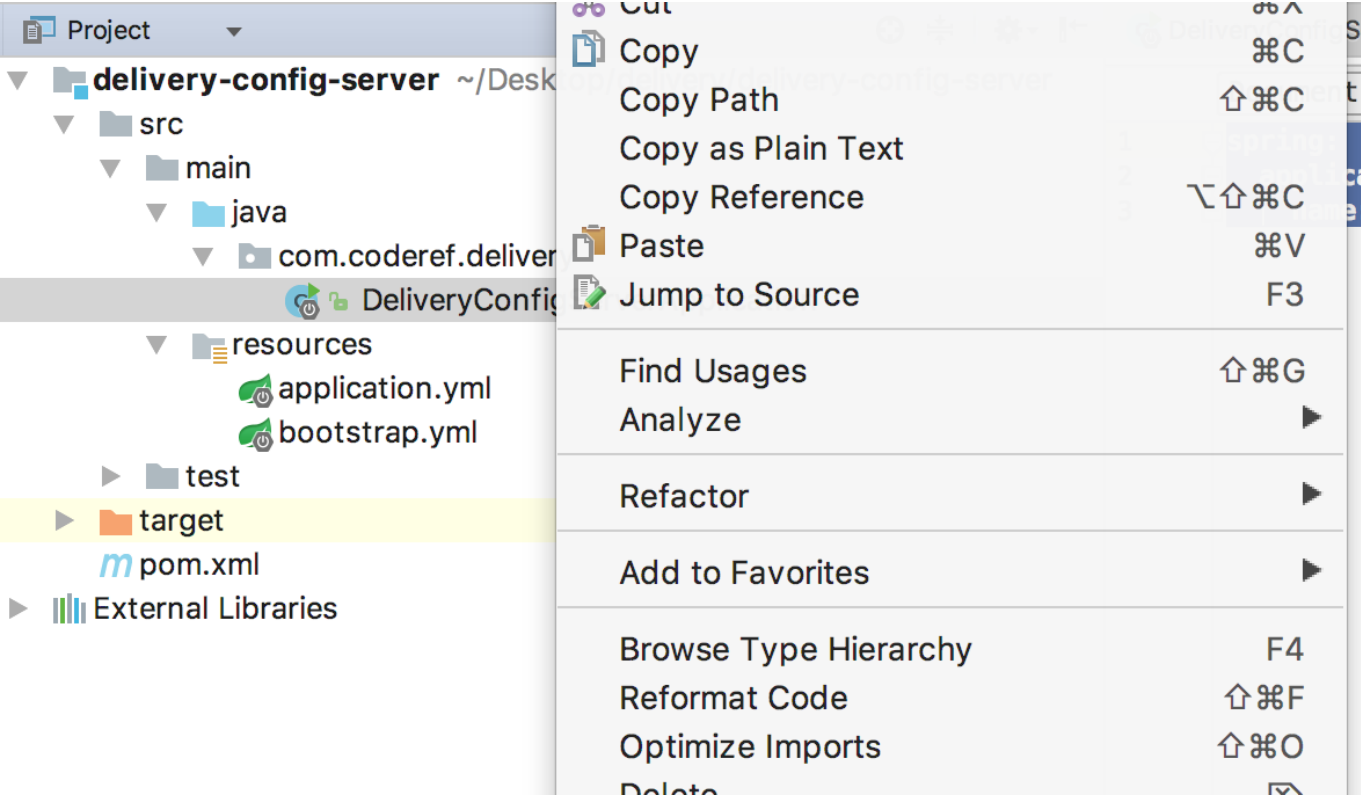
view raw

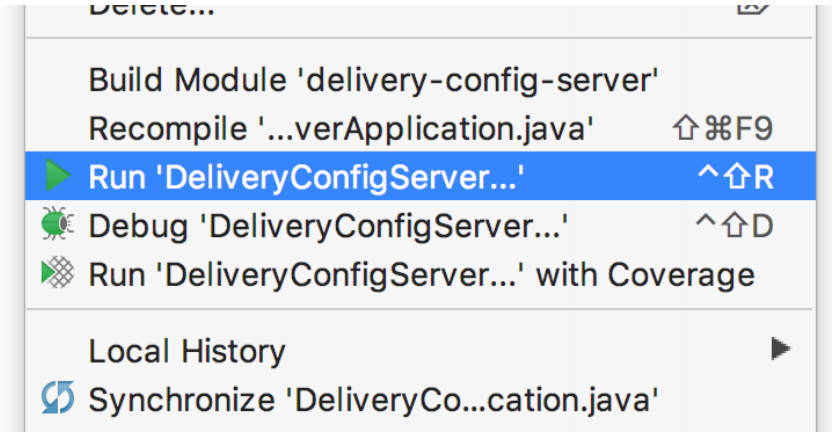
Depois de efetuar o commit e o push para o GitHub nosso repositório de configuração está assim.



Arquivo de configuração foi adicionado ao repositório remoto.

Agora sim! Suba a aplicação no Tomcat embedded executando a classe principal do projeto, ela deverá estar em pé na porta `9090` que nós configuramos no `application.yml`.





Executando a classe principal para subir a aplicação no Tomcat Embedded

Se você está vendo no log da aplicação a mensagem `Tomcat started on port(s): 9090 (http)` quer dizer que está tudo certo. Abra um navegador qualquer e acesse a seguinte UR

```
http://localhost:9090/delivery-sample-config/default
```

Você deve estar vendo na resposta do navegador a seguinte resposta contendo todas as propriedades que configuramos no `sample-config-app.yml`.

```
{
  "name": "delivery-sample-config",
  "profiles": [
    "default"
  ],
  "label": null,
  "version": null,
  "state": null,
  "propertySources": [
    {
      "name": "https://github.com/rafaelcam/delivery-
configs/delivery-sample-config.yml",
      "source": {
        "delivery.maxOrders": 10,
        "delivery.minOrders": 1
      }
    }
  ]
}
```

É isso ai galera, era isso que eu tinha pra mostrar hoje. No proximo POST vamos continuar com a série de stories criando um Service Registry (Eureka) e o conectando ao Config Server que criamos hoje.

Qualquer dúvida não deixe de comentar. Até mais!

Repositórios do Projeto

Configurações: <https://github.com/rafaelcam/delivery-configs>
Projeto: <https://github.com/rafaelcam/delivery>

Referências

<http://cloud.spring.io/spring-cloud-static/spring-cloud-config/1.3.1.RELEASE/>
<https://spring.io/guides/gs/centralized-configuration/>
<http://maven.apache.org/guides/>
<http://docs.spring.io/spring-boot/docs/1.5.4.RELEASE/reference/htmlsingle/>

Créditos

Yasmin Medeiros Gomes — Revisão e Correção.
Diego Rampim— Revisão e Correção.
Rayner Victor — Revisão e Correção.
Alexandre Carvalho Ramos — Revisão e Correção.
Diego Brener da Silva — Revisão e Correção.

[Spring Cloud](#) [Spring Boot](#) [Java](#) [Spring Cloud Config](#) [Microservices](#)

Discover Medium

Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage - with no ads in sight. Watch

Make Medium yours

Follow all the topics you care about, and we'll deliver the best stories for you to your homepage and inbox. Explore

Become a member

Get unlimited access to the best stories on Medium — and support writers while you're at it. Just \$5/month. Upgrade

[About](#) [Help](#) [Legal](#)