10

Desafio: Integração com Spring Boot

Pensando em facilitar a vida do desenvolvedor, o time do Spring criou um projeto chamado Spring Boot, cujo objetivo é acelerar o processo de configuração da aplicação. Com isso, estabeleceu configurações *default* baseadas nas experiências dos desenvolvedores do próprio Spring - assim como pelo feedback provido pelos usuários do framework.

Um *case* que se tornou comum é a integração do `Spring Boot` com o Camel para a escrita de *microserviços*. Vamos ver como é fácil fazer essas duas ferramentas conversarem:

1) Baixe o projeto *Spring Boot* (https://s3.amazonaws.com/caelum-online-public/camel/camel-spring-boot-projeto.zip) configurado para trabalharmos e importe o projeto. Caso tenha dúvidas de como importar um projeto *Maven*, você pode consultar nos exercícios (https://cursos.alura.com.br/course/camel/section/1/exercise/4) do primeiro capítulo.

2) Para facilitar a nossa configuração, trabalharemos com os *starters* do Spring Boot, que são dependências que agrupam todas as dependências necessárias para usar aquela ferramenta/biblioteca no nosso projeto. Portanto, adicione como dependência no arquivo `pom.xml` o módulo `camel-spring-boot-stater` :

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-spring-boot-starter</artifactId>
    <version>2.17.0</version>
</dependency>
```

3) Na classe `Boot` do pacote `br.com.caelum.camel` , vamos configurar o `CamelContext` para adicionar o *activemq* no `CamelContext` :

```
@SpringBootApplication
public class Boot {

    @Autowired
    private CamelContext context;

    @PostConstruct
    public void init() throws Exception {
        context.addComponent("activemq", ActiveMQComponent.activeMQComponent("tcp://localhost:6
    }

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

4) Crie uma classe `ProdutoService` que herda de `RouteBuilder` , anote-a com o stereotype `@Service` e implemente o método abstrato `configure` . Moveremos os arquivos da pasta `pedidos` para a fila `pedidos` :

```
@Service
public class ProdutoService extends RouteBuilder {
```

```
    @Override
    public void configure() throws Exception {
        from("file:pedidos").
        to("activemq:queue:pedidos");
    }
}
```

Repare que não precisamos executar os métodos `start` e `stop` do contexto, já que o Spring irá injetar uma instância de `SpringCamelContext` - que é responsável por criar e destruir o contexto, além de automaticamente encontrar todas as instâncias de `RouteBuilder` do contexto do Spring e injetar no `CamelContext` .

5) Rode a classe `Boot` e observe, no painel de administração, que as mensagens já devem estar enfileiradas.