Arquitetura de Software

Prof. Bernardo Copstein

Micro Serviços

Roteiro 7: Usando o RabbitMq

Introdução

No roteiro 6 vimos como usar o "Docker Compose" para disparar uma aplicação composta por vários micros serviços. Em relação ao nosso objetivo (figura 1) ficou faltando acrescentar a fila implementada pelo "RabbitMq" para deixar nosso sistema mais confiável.

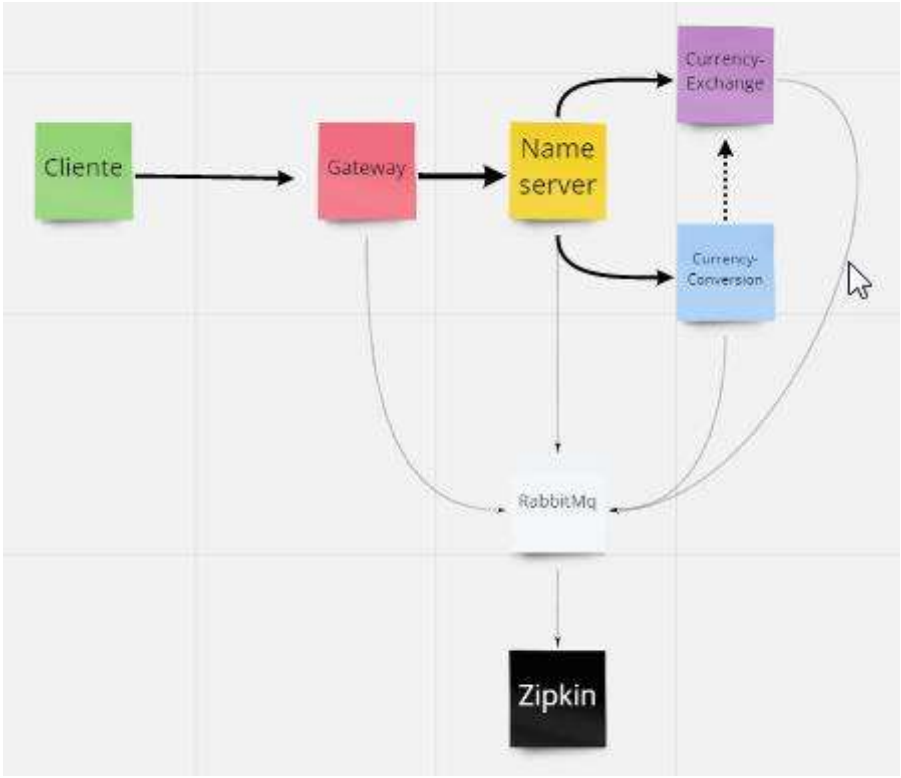

Figura 1 – Arquitetura do sistema alvo

A fila será importante nesse caso para manter as mensagens enviadas para o Zipkin de maneira que ele possa processar as mesmas na medida de sua velocidade sem perder nenhuma.

Passo 1: ativando o "RabbitMq"

O primeiro passo é indicar as dependências nos micros serviços que desejam se comunicar através do "RabbitMQ". Em cada "cliente" do "RabbitMQ" devemos inserir a seguinte dependência:

```
<dependency>
        <groupId>org.springframework.amqp</groupId>
        <artifactId>spring-rabbit</artifactId>
</dependency>
```

Adicionalmente precisamos indicar nas propriedades do micro serviço cliente que ele deve se comunicar com o "Zipkin" através do "RabbitMq". Isso é feito através da seguinte propriedade:

```
spring.zipkin.sender.type=rabbit
```

Não adianta o cliente querer se comunicar através do "RabbitMQ" se não temos um container com o "RabbitMQ" executando. É necessário então, acrescentar uma seção no arquivo "docker-compose" para descrevermos a imagem correspondente ao "RabbitMQ" e as configurações que devem ser usadas para disparar o container. Assim como no caso do "Zipkin" iremos usar uma imagem pronta disponível no "Docker Hub". A descrição segue:

```
  rabbitmq:
    image: rabbitmq:3.8.12-management
    mem_limit: 300m
    ports:
      - "5672:5672"
      - "15672:15672"
    networks:
      - currency-network
```

Por fim, é necessário indicar a localização do "RabbitMQ" no arquivo "docker-compose" para cada um dos clientes como já foi feito, por exemplo, para o "Zipkin". As linhas que seguem devem ser acrescentadas na seção "environment" tanto do micro serviço de câmbio como no de conversão e no "gateway":

```
      RABBIT_URI: amqp://guest:guest@rabbitmq:5672
      SPRING_RABBITMQ_HOST: rabbitmq
      SPRING_ZIPKIN_SENDER_TYPE: rabbit
```

No caso do micro serviço do "Zipkin" também iremos acrescentar uma seção "environment" como segue:

```
environment:
   RABBIT_URI: amqp://guest:guest@rabbitmq:5672
```

Finalmente é necessário declarar que todos os micros serviços da nossa aplicação (cambio, conversão, gateway e zipkin) dependem do "RabbitMQ". Então devemos indicar essa dependência para todos eles:

```
    depends_on:
      - rabbitmq
```

Por fim, pode ocorrer do "ZipKin" falhar sua inicialização devido a dependência do "RabbitMQ". Então é importante sinalizar que ele deve reinicializar caso sua inicialização falhe. Para tanto acrescente a seguinte cláusula na descrição do container do "ZipKin":

```
    restart: always         #Restart if there is a problem starting up
```