



Relatório - Implementação de Microsserviços com API Gateway e Eureka para Balanceamento de Cargas e Tolerância a Falhas

Professora Raqueline Penteado

Discente

R.A.	Nome
124501	Eduardo Emilio dos Santos



1. Explicação sobre o RabbitMQ

O RabbitMQ é uma ferramenta essencial no ecossistema de sistemas distribuídos, oferecendo uma solução robusta para mensageria assíncrona. Ele é amplamente utilizado para gerenciar a comunicação entre diferentes partes de um sistema, facilitando a troca de mensagens de forma eficiente e segura. Baseado no protocolo AMQP (Advanced Message Queuing Protocol), o RabbitMQ permite que as mensagens sejam enviadas e recebidas entre aplicações de maneira desacoplada, ou seja, sem que os sistemas envolvidos precisem se conhecer diretamente. Essa abordagem oferece diversas vantagens, incluindo a escalabilidade e a flexibilidade, permitindo que os sistemas sejam expandidos ou modificados sem impactar diretamente as demais partes da infraestrutura.

O funcionamento do RabbitMQ pode ser entendido em três componentes principais: produtores, filas e consumidores. Os produtores são as partes do sistema que geram as mensagens e as enviam para o RabbitMQ. O RabbitMQ, por sua vez, armazena essas mensagens em filas. As filas funcionam como um tipo de buffer, aguardando que os consumidores as processem. Esse modelo de troca de mensagens garante que a comunicação seja assíncrona, ou seja, o produtor pode continuar sua operação sem aguardar a resposta do consumidor. A principal vantagem desse modelo é que ele permite que os processos sejam desacoplados e que a carga de trabalho seja distribuída de forma eficiente entre os consumidores, possibilitando um sistema altamente escalável e resiliente.

A utilização do RabbitMQ em sistemas modernos facilita a implementação de arquiteturas de microsserviços, onde diferentes serviços podem se comunicar de maneira eficiente sem depender diretamente uns dos outros. O RabbitMQ também é altamente configurável, permitindo a criação de topologias de filas complexas, além de garantir que as mensagens sejam entregues de forma segura, mesmo em caso de falhas no sistema. Ele também oferece opções de persistência das mensagens, garantindo que nenhuma informação seja perdida, mesmo que o sistema ou uma das partes falhe temporariamente.

A aplicação desenvolvida foi projetada para exemplificar como integrar mensageria assíncrona utilizando o RabbitMQ, além de demonstrar a automação de processos de envio de



e-mails. O banco de dados escolhido para armazenar as informações dos usuários e os registros dos e-mails enviados foi o PostgreSQL. Nesse banco de dados, foram criadas tabelas específicas para vincular os dados dos usuários cadastrados e os e-mails enviados, garantindo a persistência e o controle das operações realizadas.

2. Descrição da Aplicação Desenvolvida

A aplicação desenvolvida foi projetada para exemplificar como integrar mensageria assíncrona utilizando o RabbitMQ, além de demonstrar a automação de processos de envio de e-mails. O banco de dados escolhido para armazenar as informações dos usuários e os registros dos e-mails enviados foi o PostgreSQL. Nesse banco de dados, foram criadas tabelas específicas para vincular os dados dos usuários cadastrados e os e-mails enviados, garantindo a persistência e o controle das operações realizadas.

A arquitetura da aplicação foi estruturada utilizando microsserviços, sendo um para o gerenciamento de usuários e outro para o envio de e-mails. Para a criação desses microsserviços, foi utilizado o Spring Initializr, uma ferramenta que facilita a configuração inicial de projetos Spring Boot. Através do Spring Initializr, foram definidos os principais componentes e dependências necessários para a aplicação, incluindo Spring Web, Spring Data JPA, PostgreSQL, Spring Boot Mail, entre outros. Essas dependências foram escolhidas para garantir que a aplicação fosse capaz de realizar operações de CRUD no banco de dados, enviar e-mails e interagir com a fila do RabbitMQ para processamento assíncrono.

O fluxo de funcionamento da aplicação começa quando um usuário realiza o cadastro, fornecendo seu nome e e-mail. A aplicação consulta a API externa Behind the Name para buscar o gênero e os países de origem do nome fornecido. Essas informações são então enviadas de volta ao usuário. Além disso, o sistema automatiza o envio de um e-mail de boas-vindas, incluindo os dados obtidos, como o nome, o gênero e os países de origem do nome. O envio desse e-mail foi realizado utilizando o SMTP do Gmail, garantindo uma forma eficiente e segura de enviar as mensagens.

O RabbitMQ desempenha um papel crucial nesse processo, pois ao enviar os dados do usuário para uma fila, ele permite que o envio do e-mail seja feito de forma eficiente e desacoplada do restante do sistema. Isso garante que, mesmo que o sistema de envio de e-mails tenha alguma falha temporária, os e-mails não sejam perdidos, pois as mensagens ficam armazenadas nas filas até que o consumidor esteja pronto para processá-las.



O processo de cadastro e envio de e-mail foi testado e validado utilizando o Postman, que foi configurado como cliente para realizar as requisições HTTP para a API. Através do Postman, foi possível simular o fluxo de envio de dados e a recepção da resposta, garantindo que o sistema estivesse funcionando corretamente, tanto no que se refere ao cadastro dos usuários quanto ao envio dos e-mails com as informações detalhadas.

A escolha do PostgreSQL para o armazenamento dos dados garante a integridade e a persistência das informações de forma confiável, além de permitir consultas complexas para o gerenciamento dos dados do sistema. As tabelas criadas no banco de dados são responsáveis por armazenar não apenas os dados dos usuários, mas também os registros dos e-mails enviados, permitindo o controle e a auditoria do processo de comunicação realizado pela aplicação.

3. Integração com API Gateway e Eureka

Além da comunicação com o RabbitMQ, a aplicação foi expandida para usar o Eureka e um API Gateway com balanceamento de carga e tolerância a falhas. Para isso, os microsserviços foram registrados no Eureka Server, permitindo que o API Gateway realizasse o balanceamento de carga entre instâncias replicadas de cada microsserviço. Eureka é um serviço de descoberta de serviços que permite que os microsserviços se registrem e se localizem automaticamente. O API Gateway, por sua vez, recebe as requisições do cliente e as distribui para as instâncias apropriadas dos microsserviços, garantindo que a carga seja equilibrada entre as instâncias e que o sistema continue funcionando mesmo em caso de falhas em algumas instâncias. A configuração do API Gateway e Eureka foi feita com o Spring Cloud. O balanceamento de carga é tratado pelo Spring Cloud LoadBalancer, garantindo que as requisições sejam distribuídas de forma eficiente.

4. API Gateway: Reactive Gateway

Para implementar o API Gateway, foi utilizado o Reactive Gateway, disponibilizado no Spring Initializr. O Reactive Gateway é uma solução baseada no modelo reativo, aproveitando os recursos do Spring WebFlux, que oferece um processamento assíncrono e não bloqueante. Essa abordagem permite que o API Gateway manipule um grande número de



requisições simultâneas de forma eficiente, sem comprometer o desempenho. As vantagens do uso do Reactive Gateway incluem:

- **Desempenho Não Bloqueante:** Ao usar o modelo reativo, o Reactive Gateway pode lidar com muitas requisições ao mesmo tempo, de maneira eficiente, sem bloquear threads para cada requisição. Isso ajuda a reduzir o consumo de recursos e a melhorar a escalabilidade do sistema.
- **Integração com Spring WebFlux:** O Spring WebFlux é um framework para o desenvolvimento de aplicações reativas, e o Reactive Gateway foi projetado para trabalhar perfeitamente com ele. Isso permite que a arquitetura da aplicação seja reativa desde o início, oferecendo benefícios como maior capacidade de resposta e tolerância a falhas.
- **Integração com Eureka:** O Reactive Gateway também é capaz de integrar-se facilmente ao Eureka Server para a descoberta dinâmica de serviços. Com isso, o Gateway pode consultar o Eureka para descobrir as instâncias disponíveis de microsserviços e realizar o balanceamento de carga de forma dinâmica, sem a necessidade de configurações manuais estáticas.

5. Eureka: Eureka Server e Eureka Discovery Client

Para o gerenciamento de serviços e descoberta dinâmica, foi utilizado o Eureka Server e o Eureka Discovery Client, ambos disponibilizados no Spring Initializr.

5.1. Eureka Server

O Eureka Server é o servidor central de registro de serviços, onde os microsserviços se registram para que possam ser descobertos por outros serviços.

Ele permite que os microsserviços se comuniquem sem a necessidade de conhecimento explícito sobre o endereço ou porta dos outros microsserviços, pois eles podem consultar o Eureka para obter essas informações dinamicamente.




- Registro e Descoberta de Serviços: O Eureka Server facilita o registro dos microserviços assim que eles são iniciados e permite que outros serviços façam descobertas de forma simples e rápida, otimizando a comunicação entre eles.
- Alta Disponibilidade: O Eureka Server pode ser configurado para ser altamente disponível, com replicação de instâncias, garantindo que o sistema continue funcionando mesmo em caso de falhas de instâncias do servidor de registro.

5.2. Eureka Discovery Client

O Eureka Discovery Client é usado pelos microserviços para se registrar no Eureka Server e para descobrir outros microserviços disponíveis. Cada microserviço, ao ser iniciado, se registra no Eureka Server utilizando o Eureka Discovery Client, permitindo que ele seja localizado por outros serviços ou pelo API Gateway.

- Cliente de Descoberta: O Eureka Discovery Client permite que o microserviço se mantenha atualizado sobre o status das instâncias de outros serviços e se ajuste automaticamente caso haja falhas ou mudanças nas instâncias disponíveis.
- Balanceamento de Carga: Com a integração entre o Eureka Discovery Client e o API Gateway, o tráfego pode ser balanceado entre as instâncias dos microserviços registrados, melhorando a distribuição de carga e a resiliência do sistema.

6. Vídeo demonstrando a execução da aplicação.

 Trabalho3video.mp4