Fundação Universidade Regional de Blumenau (FURB)

Alunos: Bruno Ricardo Junkes, Igor Christofer Eisenhut e Manoella Marcondes Junkes

Disciplina: Sistemas Distribuídos

Professor: Aurélio Faustino Hoppe

APLICAÇÃO DE RABBITMQ NO CONTEXTO DE UM E-COMMERCE

Com a crescente utilização dos meios digitais para a comercialização em geral, especialmente no período de pandemia onde muitos foram impossibilitados de sair de casa por conta do *lockdown*, os *e-commerces* sentiram um aumento crescente na demanda por seus serviços. Além desta situação em especial, todo ano ocorrem grandes promoções, como por exemplo a *Black Friday*, onde muitos aproveitam para economizar nas compras. Nestas promoções, geralmente observa-se uma lentidão na nos sites que disponibilizam seus produtos com grandes descontos. Isto ocorre justamente pela imensa quantidade de pessoas acessando o serviço ao mesmo tempo, com o sistema não conseguindo atender todas as requisições em um tempo hábil.

ARQUITETURA DE MICROSSERVIÇOS E O RABBITMQ

Estas situações demandam uma arquitetura específica para obter uma melhor performance, no caso, a arquitetura de microsserviços. Na arquitetura de microsserviços, o sistema é separado em pequenas partes independentes, de forma que é possível garantir mais recursos a um determinado processo que está mais sobrecarregado (em termos técnicos, escalar o serviço), sem ser necessário dar mais recursos a todo o restante do sistema. Esta particularidade dos sistemas com arquitetura de microsserviços pode ser garantida através da utilização de mensagerias. Mensageria é um conceito que se aplica a sistemas distribuídos onde a comunicação entre eles é realizada através de mensagens, sendo estas mensagens gerenciadas por um *message broker*, isto é, um servidor central.

O RabbitMQ é um dos serviços de mensageria disponíveis no mercado, onde é possível cadastrar filas para determinados “assuntos” e consumidores que processarão as mensagens inseridas nestas filas. Como utiliza a rede para comunicação, não existe acoplamento entre o produtor da mensagem e o consumidor, o que possibilita que diferentes consumidores processem mensagens de uma determinada fila. Isto cria uma espécie de balanceamento de carga, onde o RabbitMQ divide as requisições igualmente entre os consumidores. Desta forma, para o cenário de um e-commerce citado inicialmente, seria possível inserir múltiplos consumidores para processar as solicitações de compra que chegam ao site da loja, de forma que a carga seja dividida entre eles e, por consequência, o tempo de resposta é igualmente reduzido.

Analisando as diferenças arquiteturais entre a implementação com microsserviços e a implementação monolítica, é possível notar a versatilidade da utilização de serviços de mensageria como o RabbitMQ: enquanto na arquitetura monolítica é necessário que o servidor web conheça o serviço que irá processar as requisições, com microsserviços o servidor apenas sabe como se comunicar com o RabbitMQ, e ele realiza a distribuição entre os consumidores disponíveis.

Diagrama 1: Implementação monolítica

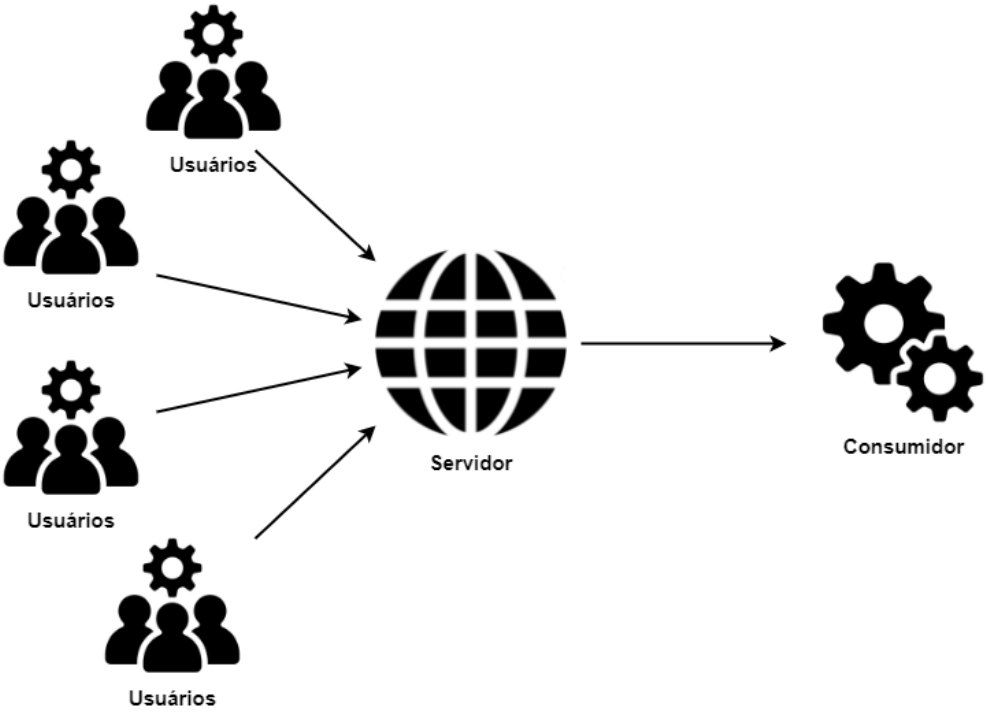
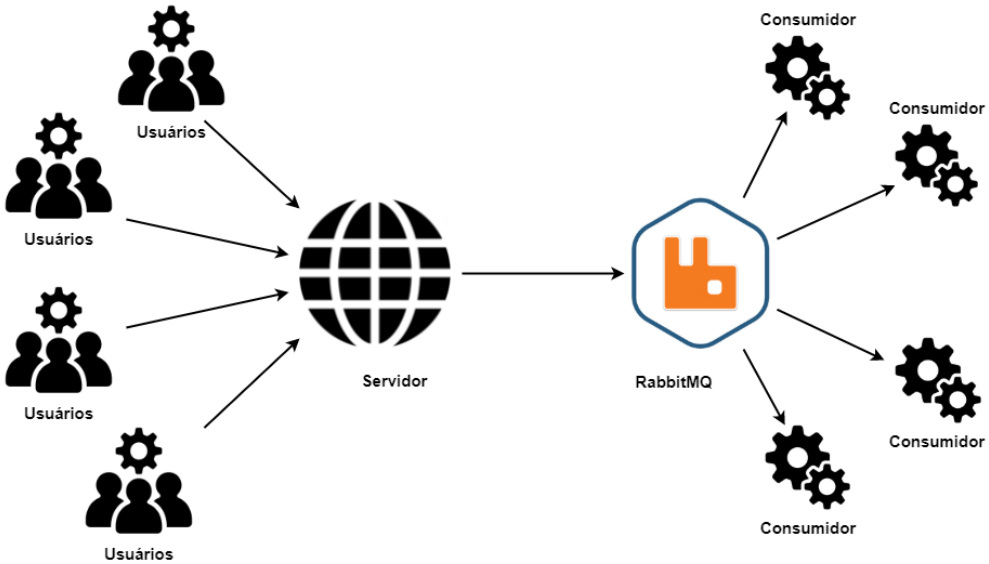


Diagrama 2: Implementação com microsserviços e o RabbitMQ



COMPARAÇÃO PRÁTICA

Realizando uma comparação entre um sistema de e-commerce que utiliza uma arquitetura monolítica e um sistema de e-commerce que utiliza uma arquitetura de microsserviços com o RabbitMQ, em um cenário hipotético onde são realizados 5000 pedidos de compra, com cada um levando em média 200 milissegundos para ser processado, teríamos os seguintes tempos de processamento como resultado:

Tipo de arquitetura	Tempo de processamento total
Monolítica	1000 segundos (16min e 40s)
Microsserviços com 3 consumidores	333,33 segundos (5min e 33s)
Microsserviços com 5 consumidores	200 segundos (3min e 20s)
Microsserviços com 10 consumidores	100 segundos (1min e 40s)
Microsserviços com 100 consumidores	10 segundos

CONCLUSÃO

Nota-se o enorme ganho de performance alcançado utilizando a arquitetura adequada e a quantidade de consumidores adequada. Obviamente, a escala do serviço envolve custos, portanto é necessário encontrar uma configuração com um bom custo-benefício para o cenário que está tentando se resolver.

Além da divisão de processamento, um outro benefício da utilização de mensageria é a possibilidade de tratamento de erros em um segundo momento, ou de realizar ações de compensação, por processos distintos. Isto porque a existência de filas permite que erros sejam tratados como uma mensagem comum, portanto, seria possível configurar uma fila secundária para receber as ocorrências de erro de um determinado fluxo. As mensagens desta fila podem ser consumidas por um segundo processo, liberando o processo principal para o consumo de novas requisições.