Mensageria com RabbitMQ

Facef 2020

Agenda

1. Introdução
2. Enviando e Recebendo Mensagens
3. Worker Queues
4. Publish/Subscribe
5. Roteamento
6. Tópicos
7. Dead Letter Queue

Material de apoio https://github.com/diegofernandes/rabbitmq-facef

Introdução

- Padrão de comunicação Assíncrono(Caixa de Correio);
- Passagem de controle e dados;
- Modelo ponto a ponto:
  - Apenas um consumidor irá receber;
- Modelo publish/subscribe:
  - Consumidores registram o tipo de mensagens que irão receber;

Protocolos / Implementações

- AMQP - Advanced Message Queuing Protocol - Rico em opções;
- STOMP - Streaming Text Oriented Messaging Protocol - Simples, orientado a mensagens de Texto;
- MQTT - MQ Telemetry Transport - Super leve,orientado a dispositivos embarcados;
- HTTP/GRPC(HTTP2) - Em alguma implementações proprietárias(Amazon SQS, Google PUB/SUB)

Protocolos / Implementações

- Apache ActiveMQ;
- Apache Kafka;
- RabbitMQ;
- Redis;
- Nats;

Considerações com Sistemas de Mensageria

- Durabilidade - Mensagens armazenadas em memória, disco, banco de dados(DBMS);
- Políticas de Segurança - Quais aplicações podem ter acesso às mensagens;
- Políticas de limpeza - Quanto tempo uma mensagem fica armazenada(time to live - TTL);
- Filtros - Alguns sistemas conseguem filtrar mensagem por algum critério;
- Política de entrega - Entrega garantida ao menos 1 vez, ou mais que uma;
- Roteamento - Quem deveria receber as mensagem ou quais filas;
- Batching - Mensagem devem ser enviadas imediatamente, ou em lote;

RabbitMQ

- Suporta Múltiplos protocolos - STOMP, MQTT, ACTIVEMQ, HTTP, etc;
- Roteamento de mensagens;
- Open Source;
- Notificação de entrega;
- Plugins;
- Console Web, CLI;
- Suporta múltiplas linguagens de programação;

Instalando o RabbitMQ

- https://www.rabbitmq.com/download.html
- docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3-management
  - docker run --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3-management
  - docker stop rabbitmq | docker start rabbitmq
- microk8s helm3 install rabbitmq stable/rabbitmq
- http://localhost:15672/
  - Docker: guest/guest
  - K8s: echo "Password : $(kubectl get secret --namespace default rabbitmq -o jsonpath="{.data.rabbitmq-password}" | base64 --decode)"

Enviando e Recebendo Mensagens

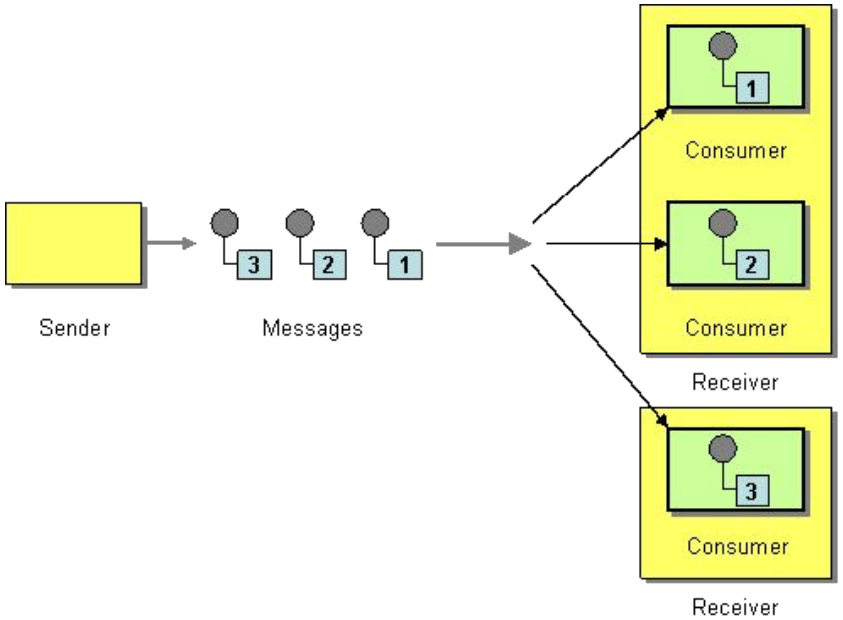
- Produce(P) - produtor da mensagem, origem;
- Fila(Queue) - Recurso gerenciado pelo RabbitMQ;
- Consumer(Consumidor da Fila, destino;

Exemplo 1

- Dependências JDK(java), Mavem, IDE Favorita Eclipse, VSCode, etc
  - **sudo apt install openjdk-11-jdk maven**
  - **brew cask install adoptopenjdk11**
  - **brew install maven**
  - **choco install jdk11 maven**
  - https://www.oracle.com/java/technologies/javase-jdk11-downloads.html
  - https://www.eclipse.org/downloads/packages/
  - https://maven.apache.org/install.html
- https://start.spring.io/ - Criador de projetos Spring Boot
- Adicionar a dependência Spring RabbitMQ
  - mvn spring-boot:run
  - https://github.com/diegofernandes/rabbitmq-facef - hello-consumer, hello-emitter

Workers Queues

- Distribuir carga de trabalho entre os workers;
- Padrão de concorrência entre os consumidores;
- Múltiplos consumidores;
- Mensagem entregue ao menos uma vez;
- Alto Volumes;
- Escala do Ambiente;
- Horizontal ou Vertical;



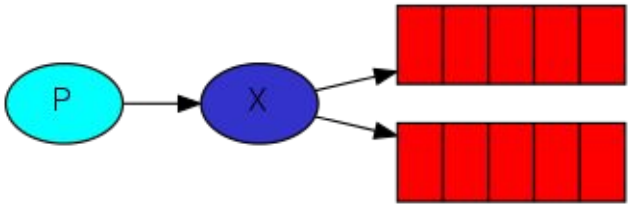
Exemplo 2

- https://github.com/diegofernandes/rabbitmq-facef - hello-consumer, hello-emitter;
- Vamos iniciar várias instâncias do hello-consumer;
- Vamos alterar o hello-emitter para que ele emita várias mensagens;
- Vamos alterar o consumer afim de simular uma carga de trabalho;

Código pronto no branch - workerqueues

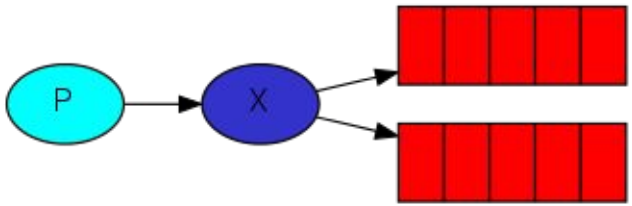
Publish/Subscribe

- Difundir a mensagem(broadcast);
- Todas as filas recebem a mensagem;
- Eventos de Notificação para áreas/sistemas distintos;
- Cada fila pode tem um conjunto de consumidores;
- Tópicos(Topic)
    - rabbitmq Exchanges
    - AWS SNS/SQS
    - GCP PUB/SUB



RabbitMQ Exchanges

- Exchange: recebe as mensagens do produtor e encaminhar para filas;
- Queues: Armazenam as mensagens e encaminha para os consumers;
- Default exchange;
  - Exchange padrão do RabbitMQ do tipo direct;
- Direct exchange;
  - Faz um encaminhamento das mensagens **direto** para uma fila;
  - Usa de base o ***routingKey;***
- Fanout exchange;
  - Encaminha cópias das mensagens para cada fila associada a exchange;
  - Ignora o **routingKey**;
- Topic exchange;
  - Encaminha as mensagens com base no **routingKey;**



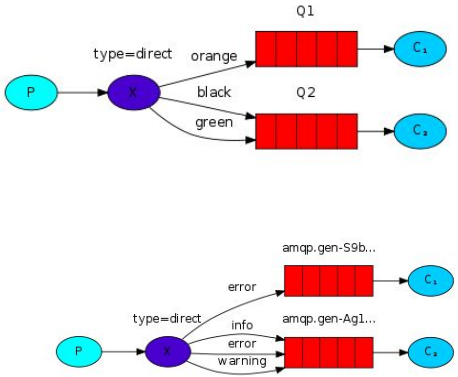
Exemplo 3

- Vamos criar duas filas "Sonia" e "Leo" (Webconsole)
- Vamos criar um exchange do tipo fanout com o nome de "fofoqueiro"; (Webconsole)
- Vamos associar as filas ao exchange; (Webconsole)
- Vamos subir um consumidor para a cada fila;
- Vamos alterar o hello-emitter para enviar via exchange "fofoqueiro";
- (Consumer)mvn clean package
- (Consumer)java -jar target/hello-consumer-0.0.1-SNAPSHOT.jar --queueName=leo
- (Consumer)java -jar target/hello-consumer-0.0.1-SNAPSHOT.jar --queueName=sonia
- (Emitter)mvn spring-boot:run

Código pronto no branch - pubsub

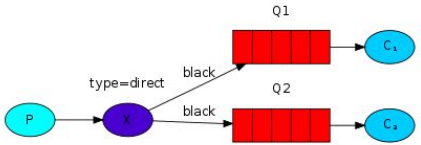
Routing

- Exchange do tipo **fanout**: todas as mensagens são enviadas para todas as queues (Broadcasting).
- Exchange do tipo **direct**: realização do roteamento das mensagens através da **routing-key**
- Mensagens serão enviadas para as queues que possuam a **binding-key** que realize o match com a **routing-key**



Routing - Multiples bindings

- É possível realizar o bind para múltiplas queues utilizando a mesma **binding-key**.
- Queues Q1 e Q2 irão fazer o binding utilizando a **binding-key black**.
- Nesse exemplo a exchange do tipo **direct** irá se comportar exatamente igual a exchange do tipo **fanout**, pois as filas Q1 e Q2 irão receber as mensagens.



Exemplo 4

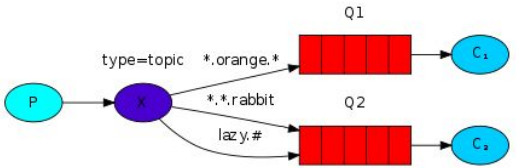
- URL para geração do projeto
- Vamos criar uma exchange chamada **order-exchange** do tipo **direct**
- Vamos criar duas filas **payment-creditcard-queue, payment-bankslip-queue**
- Vamos realizar o binding das filas com a exchange.
- Vamos criar uma classe DTO para utilizar como exemplo do body do request.
- Vamos criar uma Service para realizar o mapeamento da routing-key e enviar a mensagem para o RabbitMQ.
- Vamos criar um controller para expor a rota para receber os dados via API Rest.
- Vamos criar um Listener para cada fila para consumir as mensagens;

Código pronto no branch - rabbitmq-routing

Routing com Topic

- Mensagens enviadas para uma exchange do tipo **topic** não pode ter uma lista de **routing-key**.
- Em exchange do tipo **topic**, podemos ter uma lista de palavras separadas por **pontos(.)**, podendo se utilizar qualquer palavra, mas é comum utilizar palavras que tenham relação com a mensagem.
- Exemplos de routing-keys: **pedido.cartao.debito, pedido.cartao.parcelado**, pode-se utilizar várias palavras respeitando o limite de **256 bytes**.
- Binding-key deve seguir o mesmo formato da routing-key.

Routing com Topic



- Casos especiais sobre **binding-keys**:
  - * **(star)** pode substituir exatamente **uma palavra.**
  - **# (hash)** pode substituir **zero ou mais palavras.**
- Exemplo de roteamento:
  - **<speed>.<colour>.<species>**
  - **Q1** irá receber todos animais laranja.
  - **Q2** irá receber tudo sobre **coelhos** e tudo sobre animais **lentos.**
- Exchange do tipo **Topic** podem se comportar como outras exchanges, utilizando os casos especiais sobre **binding-keys**
  - Queue vinculado a **#** irá receber tudo e se comportar como **fanout**
  - Queue que não possui nenhum caractere especial * **ou #** irá se comportar como uma **direct**.

Exemplo 5

- URL para geração do projeto
- Vamos criar uma exchange chamada **city-exchange** do tipo **topic**
- Vamos criar três filas **small-cities-queue, medium-big-cities-queue, all-cities-queue**
- Vamos realizar o binding das filas com a exchange.
- Vamos criar uma classe DTO para utilizar como exemplo do body do request.
- Vamos criar uma Service para realizar o mapeamento da routing-key e enviar a mensagem para o RabbitMQ.
- Vamos criar um controller para expor a rota para receber os dados via API Rest.
- Vamos criar um Listener para cada fila para consumir as mensagens;

Código pronto no branch - rabbitmq-topic

DLQ - Dead Letter Queue

- Vários tipos de erros podem acontecer em um sistema de troca de mensagens. Até mesmo mais erros do que em sistemas tradicionais (monolítos).
- Tipos de erros comuns:
  - Falhas de rede ou operação de Leitura e Escrita.
  - Erros por falta ou falha de configuração do sistema de mensageria.
  - Falhas nas configurações entre clientes e brokers (limites, autenticação e etc).
  - Exceções que violam alguma regra de negócio ou da aplicação
- Podem existir vários outros tipos de falhas, além das mais comuns já citadas.

Exemplo 6

- URL para geração do projeto
- Vamos criar uma exchange chamada **order-exchange** do tipo **direct**
- Vamos criar uma filas **order-messages-queue**
- Vamos realizar o binding das filas com a exchange.
- Vamos criar uma classe producer para enviar mensagens para o broker
- Vamos alterar a aplicação para enviar mensagens através do producer durante sua inicialização.
- Vamos criar um consumer que irá lançar uma Exception
- Vamos realizar tratativas para não ficar em loop o processamento.
- Vamos realizar a criação e configuração das filas DLQ.

Código pronto no branch - rabbitmq-dlq

Parking Lot Queues

- Cenários onde não podemos perder ou descartar mensagens (transação bancária, pedido e etc)
- Processamento com intervenção manual.
- Armazenamento em outra fonte após **x** tentativas.
- Cenário comum:
    - Processar mensagens nas filas DLQ.
    - Após atingir o limite de falhas enviar para a fila de **parking-lot**.

Exercício para casa

- URL para geração do projeto
- Realizar a implementação de um cenário de parking-log baseado no exemplo 6 (DLQ).