

Rabbit MQ



Com Kotlin e Spring

Quem sou eu



Oriol Canalias



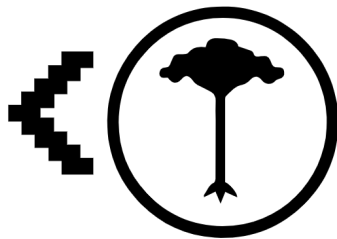
Sabadell



8 anos no Brasil



Dev Backend na Wavy Global
(grupo movile)



Comunidade
Opensanca

Disclaimer

- Baseado na minha experiência
- Não acredite em tudo o que vou falar
- Faça perguntas, mesmo achando que são bobas



Definição

RabbitMQ é um software open source de mensageria.

Fornece uma forma comunicação assíncrona de dados entre processos, aplicações ou servidores.

É um dos brokers de mensagens mais utilizados e implementa o protocolo AMQP

RabbitMQ VS Apache Kafka

- RabbitMQ é uma implementação de message broker
- Kafka é uma ferramenta para processamento de streams de dados

Message Routing

Ordering

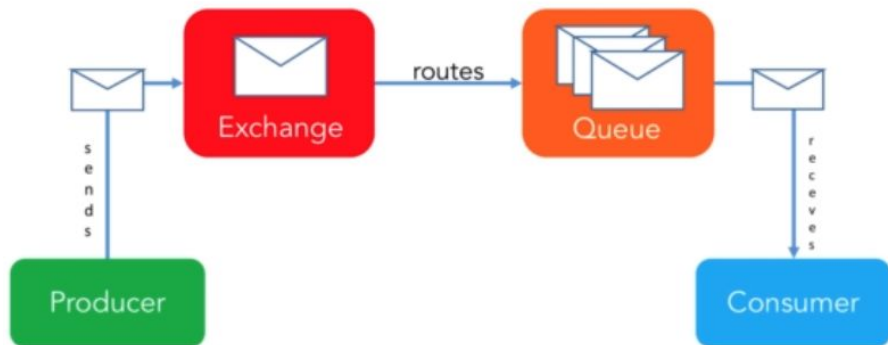
TTL

Retention

Fault Handling

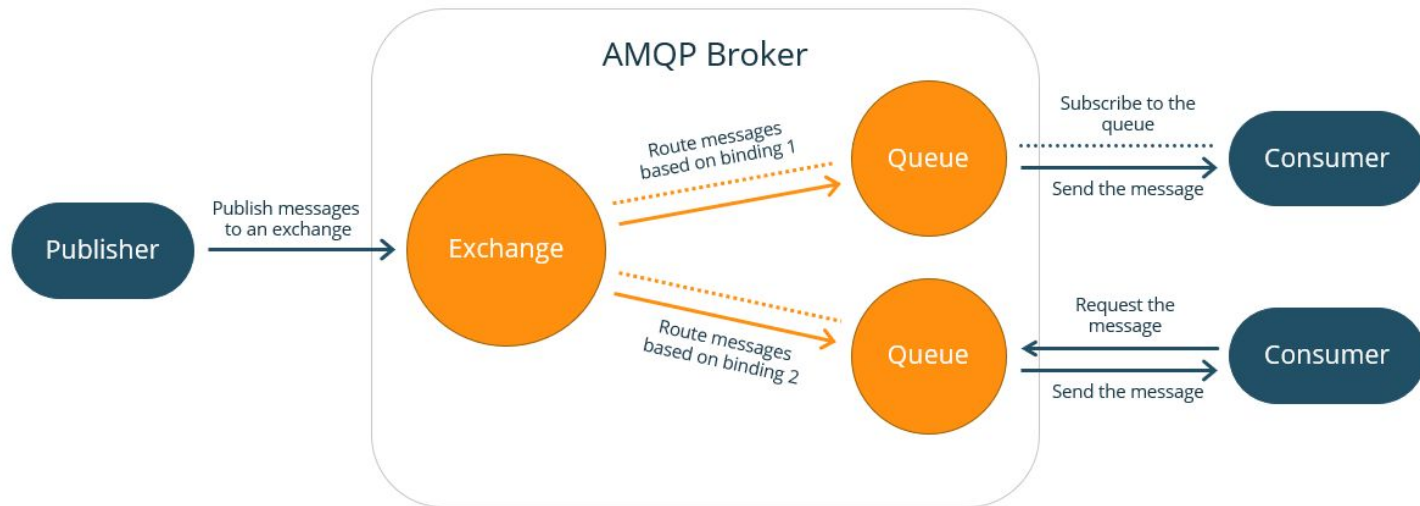
Reprocessing

Como funciona



Como qualquer comunicação:
com **emissor**, **receptor** e **mensagem**, através de um **meio**

Protocolo AMQP



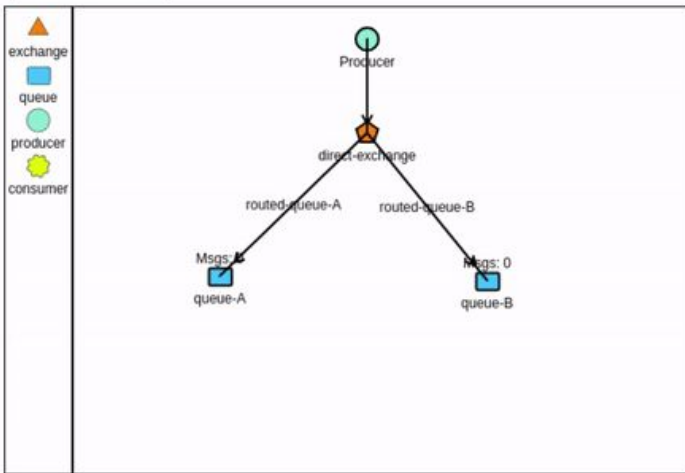
Não enviamos a mensagem **para uma fila diretamente** e sim para um exchange

Definições rápidas

- Binding: É a ligação entre uma fila e um exchange
- Binding key: É uma chave específica da ligação entre a fila e o exchange
- Routing key: É uma chave enviada junto a mensagem que o exchange usa para decidir para que fila (ou filas) vai rotear uma mensagem.

Exchanges: Direct

Encaminha as mensagens procurando por um binding key igual ao routing key



Properties

Edit Producer

Producer

Delete Edit

New Message

testing message

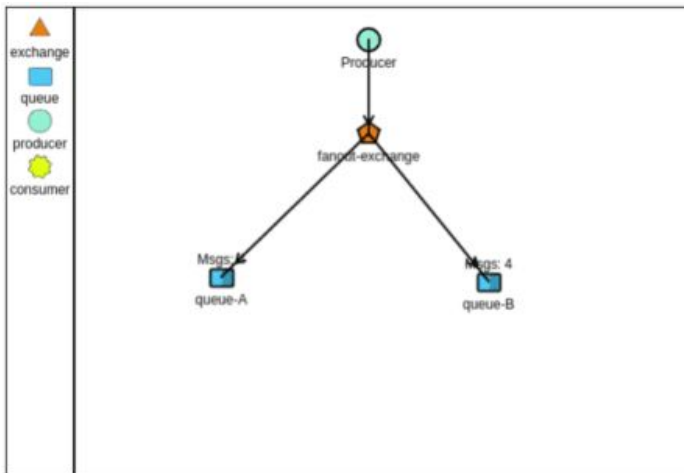
routed-queue-A

seconds

Stop Send

Exchanges: Fanout

Encaminha para todas as filas com binding nele, desconsiderando routing key



Properties

Edit Producer

Producer

Delete Edit

New Message

testing message

routed-queue-A

seconds

Stop Send

Exchanges: Topic e Header

Topic: Parecida ao Direct, porém o binding key é um tipo de “expressão regular” aplicada sobre o routing key.

routing-key: kotlin.rabbitmq.tecnologia

binding-key: #.kotlin.# ou #.tecnologia.#

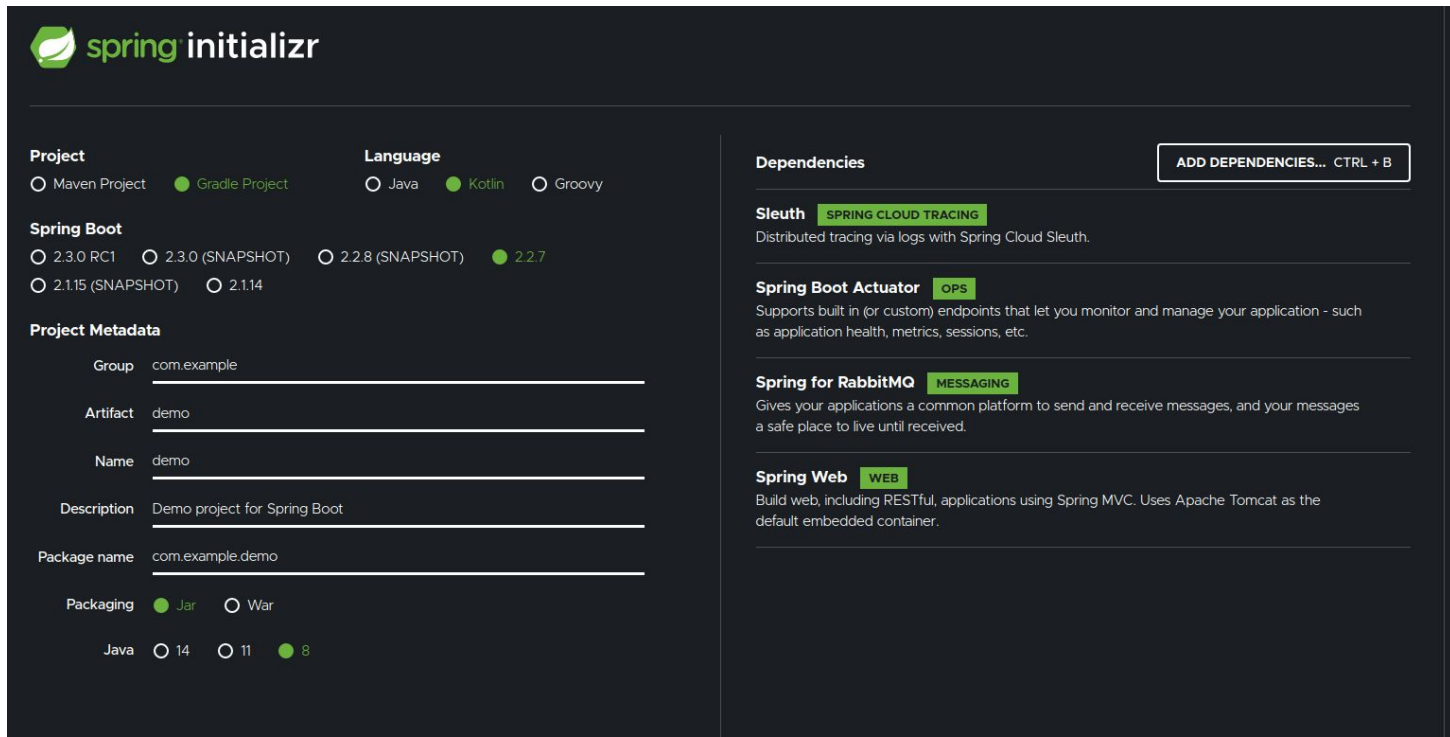
Header: Usa os valores do header da mensagem para fazer o encaminhamento, ignorando o routing key**

Hands-on: RabbitMQ em local

Rodar em local:

```
$ docker run -d --name local-rabbit -p 15672:15672 -p 5672:5672 \  
-e RABBITMQ_DEFAULT_USER=admin \  
-e RABBITMQ_DEFAULT_PASS=admin \  
rabbitmq:3-management
```

Usando Rabbit com Kotlin e Spring



The image shows the Spring Initializr web interface. On the left, under 'Project', 'Maven Project' is selected. Under 'Language', 'Kotlin' is selected. Under 'Spring Boot', '2.2.7' is selected. The 'Project Metadata' section contains fields for Group (com.example), Artifact (demo), Name (demo), Description (Demo project for Spring Boot), and Package name (com.example.demo). Under 'Packaging', 'Jar' is selected. At the bottom, 'Java' is selected with version '8'. On the right, the 'Dependencies' section lists 'Sleuth' (Spring Cloud Tracing), 'Spring Boot Actuator' (Ops), 'Spring for RabbitMQ' (Messaging), and 'Spring Web' (Web). A button 'ADD DEPENDENCIES... CTRL + B' is at the top right of the dependencies section.

Project

☐ Maven Project ☒ Gradle Project

Language

☐ Java ☒ Kotlin ☐ Groovy

Spring Boot

☐ 2.3.0 RC1 ☐ 2.3.0 (SNAPSHOT) ☐ 2.2.8 (SNAPSHOT) ☒ 2.2.7

☐ 2.1.15 (SNAPSHOT) ☐ 2.1.14

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☐ 14 ☐ 11 ☒ 8

Dependencies ADD DEPENDENCIES... CTRL + B

Sleuth SPRING CLOUD TRACING
Distributed tracing via logs with Spring Cloud Sleuth.

Spring Boot Actuator OPS
Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.

Spring for RabbitMQ MESSAGING
Gives your applications a common platform to send and receive messages, and your messages a safe place to live until received.

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Usando Rabbit com Kotlin e Spring

```
21 dependencies {
22     implementation("org.springframework.boot:spring-boot-starter-actuator")
23     implementation("org.springframework.boot:spring-boot-starter-amqp")
24     implementation("org.springframework.boot:spring-boot-starter-web")
25     implementation("com.fasterxml.jackson.module:jackson-module-kotlin")
26     implementation("org.jetbrains.kotlin:kotlin-reflect")
27     implementation("org.jetbrains.kotlin:kotlin-stdlib-jdk8")
28
29     implementation("io.springfox:springfox-swagger2:$swaggerVersion")
30     implementation("io.springfox:springfox-swagger-ui:$swaggerVersion")
31
32     implementation("org.springframework.cloud:spring-cloud-starter-sleuth")
33 }
```

Configuração básica

```
spring:
  application:
    name: rabbitmq-producer-basic
  rabbitmq:
    port: 5672
    username: guest
    password: guest
    host: localhost
```

Config básica no *application.yml*

Enviando uma mensagem

```
rabbitTemplate.convertAndSend(exchange, routingKey, message)
```

Configuração básica

```
@Configuration
class QueueConfig {

    @Bean
    fun firstQueue(): Queue {
        return QueueBuilder
            .durable( name: "FIRST-QUEUE-BASIC")
            .build()
    }

    @Bean
    fun directExchange(): Exchange {
        return ExchangeBuilder
            .directExchange( name: "DIRECT-EXCHANGE-BASIC")
            .durable( isDurable: true)
            .build()
    }

    @Bean
    fun firstDirectBinding(firstQueue: Queue): Binding {
        return BindingBuilder
            .bind(firstQueue)
            .to(directExchange())
            .with( routingKey: "TO-FIRST-QUEUE")
            .noargs()
    }
}
```

Definimos a fila

Definimos o exchange

Definimos o binding

Configuração básica

```
@Component
class FirstQueueConsumer {
    private val log : Logger! = LoggerFactory.getLogger(javaClass)

    @RabbitListener(queues = ["FIRST-QUEUE-BASIC"])
    fun receiveMessageFromFirstQueue(message: Message) {
        val bodyAsString : String = message.body?.let { String(it) } ?: ""
        log.info("body $bodyAsString")
    }
}
```

Live Code: Kotlin e Spring

Os exemplos serão com:

- Kotlin
- Spring-boot
- Spring-rabbit



<https://github.com/iundarigun/learning-rabbitmq>

Configuração (um pouco) avançada

- Políticas de retry: O que devemos fazer quando temos um erro
 - No consumer
 - Fila de delay
 - Reprocessando na mesma fila
- Dead Letter Queue: Ou DLQ para os íntimos. É uma fila onde mandamos tudo o que não foi processado com sucesso

Outras configs

- Delay queues
- Virtual hosts
- Gerenciar mais de uma conexão no Rabbit
- Clusters e replicação

Links e referências

- <https://medium.com/better-programming/rabbitmq-vs-kafka-1ef22a041793>
- <https://medium.com/better-programming/rabbitmq-vs-kafka-1779b5b70c41>
- <https://itnext.io/kafka-vs-rabbitmq-f5abc02e3912>
- <https://medium.com/dev-cave/rabbit-mq-parte-i-c15e5f89d94>
- <https://medium.com/dev-cave/rabbitmq-parte-ii-fa61a469ba2>
- <https://medium.com/dev-cave/rabbit-mq-extras-efa038e53db1>
- <https://www.cloudamqp.com/docs/index.html>
- <http://tryrabbitmq.com/>
- <https://github.com/iundarigun/learning-rabbitmq>
- <https://open.spotify.com/show/1CtIJ3aoBNjPytlmxFcBhv?si=QMf5DsJeTdaxaprJRNaYXw>

Obrigado



slides: <http://ves.cat/eud4>