

03

 RabbitMQ™

Getting started
+ Spring + Kotlin



AOBA!

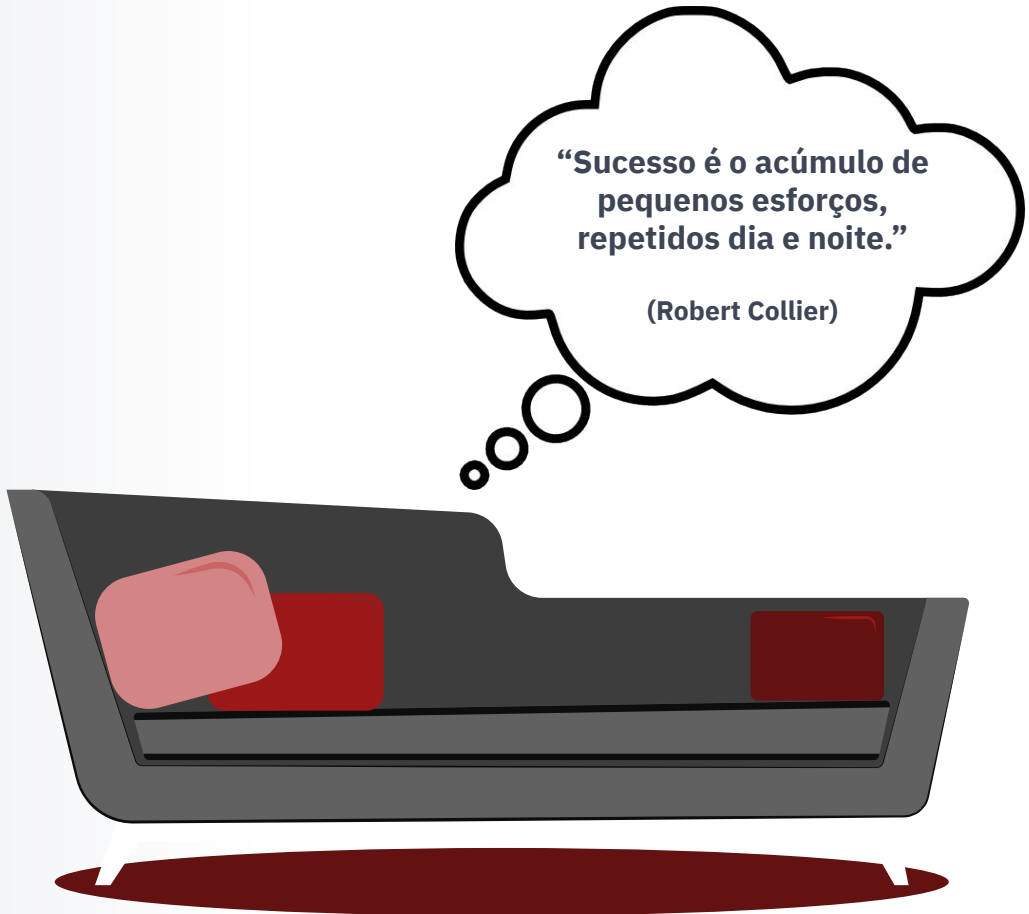
Ricardo de Luna Galdino
Software Engineer



LINKEDIN:
[linkedin.com/in/ricardo-galdino](https://www.linkedin.com/in/ricardo-galdino)

GITHUB:
github.com/ricardogaldino

WHATSAPP GROUP:
engsoft.org



**“Sucesso é o acúmulo de
pequenos esforços,
repetidos dia e noite.”**

(Robert Collier)

Cultura ágil de aprendizado



Microlearning:

- É uma metodologia de ensino que subdivide um assunto em doses menores de conteúdo, com atividades rápidas, auxiliando na compreensão e retenção deste conteúdo.

Pílulas do Conhecimento:

- São pequenos conteúdos apresentados ao profissional para que consiga assimilar de forma mais focada e objetiva, melhorando a eficiência e potencializando os resultados obtidos.
- Microlearning é composto por diversas Pílulas do Conhecimento.

Glossário

Vamos rever alguns conceitos e termos...

Spring Framework

- ▶ O Spring é um **framework** open source para a **plataforma Java**.
- ▶ Criado por **Rod Johnson** e descrito em seu livro "Expert One-on-One: JEE Design e Development" em **2002**.
- ▶ Trata-se de um framework **baseado** nos padrões de projeto de inversão de controle e injeção de dependência (**IoC**).
- ▶ É **constituído por** diversos e completos **módulos**.

<https://spring.io/>

Spring AMQP (RabbitMQ)

- ▶ É um **módulo** do Spring Framework para desenvolvimento de **soluções em mensageria** baseadas em **AMQP**.
- ▶ Ele fornece uma abstração de alto nível para enviar e receber mensagens.

<https://spring.io/projects/spring-amqp>
<https://spring.io/guides/gs/messaging-rabbitmq/>

Spring Boot

- ▶ É um **módulo** do Spring Framework desenvolvido com base na ideia de **convenção** sobre configuração, ou seja, **apenas utilizar submódulos necessários sem** preocupação com o trabalho de **configuração**.
- ▶ **Facilita a criação e execução de aplicações** como **microsserviços**.

<https://spring.io/projects/spring-boot>

Kotlin

- ▶ Kotlin é uma **linguagem de programação** multiplataforma, orientada a objetos e funcional, concisa e estaticamente tipada.
- ▶ Desenvolvida pela **JetBrains** em **2011**.
- ▶ Utiliza a **Máquina virtual Java** para compilação.

<https://kotlinlang.org/>

Spring + RabbitMQ

Desenvolvendo uma aplicação com Kotlin

Etapas para construção da Aplicação

- ▶ **Projeto** em Spring Boot com Kotlin e RabbitMQ
- ▶ **Estrutura** do projeto baseada em Domain-Driven Design (**DDD**)
- ▶ Configuração das **propriedades** do RabbitMQ
- ▶ Configuração dos **Beans** do RabbitMQ
- ▶ **Criação programática** (via código) das **queues**, **exchanges** e **bindings** do RabbitMQ
- ▶ Infraestrutura para o **Producer**
- ▶ Infraestrutura para o **Consumer**
- ▶ **Serviço** para o **Producer**
- ▶ **Serviço** para o **Consumer**
- ▶ **Envio** direto de mensagens com erro para **Dead Letter Queue**
- ▶ **Controlador** para **Publicação da Mensagem**
- ▶ **Publicação da Mensagem**
- ▶ **Consumo da Mensagem**

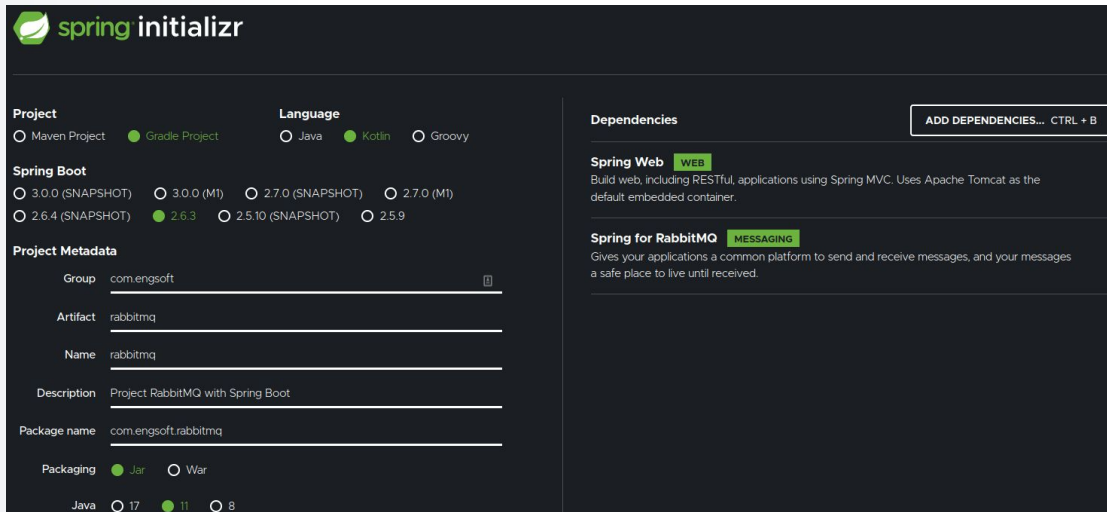
Código fonte:

- ▶ O código fonte **completo** do **projeto** já está **disponível** em:
<https://github.com/ricardogaldino/microlearning-rabbitmq>

Projeto em Spring Boot com Kotlin e RabbitMQ

<https://spring.io/quickstart>

- ▶ **Spring Initializr** é uma ferramenta Web que **gera** a estrutura do **projeto Spring Boot** com suas **dependências**.
- ▶ Acesse o Spring Initializr:
<https://start.spring.io/>
- ▶ Crie um projeto **Kotlin** com as **dependências** “**Spring Web**” (API) e “**Spring for RabbitMQ**”.



The screenshot shows the Spring Initializr web interface. The 'Project' section has 'Maven Project' selected. The 'Language' section has 'Kotlin' selected. The 'Spring Boot' section has '2.6.3' selected. The 'Project Metadata' section has 'Group' set to 'com.engsoft', 'Artifact' set to 'rabbitmq', 'Name' set to 'rabbitmq', 'Description' set to 'Project RabbitMQ with Spring Boot', and 'Package name' set to 'com.engsoft.rabbitmq'. The 'Packaging' section has 'Jar' selected. The 'Dependencies' section has 'Spring Web' and 'Spring for RabbitMQ' selected. A button 'ADD DEPENDENCIES... CTRL + B' is visible in the top right of the dependencies section.

Project

☐ Maven Project ☒ Gradle Project

Language

☐ Java ☒ Kotlin ☐ Groovy

Spring Boot

☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (M1) ☐ 2.7.0 (SNAPSHOT) ☐ 2.7.0 (M1)

☐ 2.6.4 (SNAPSHOT) ☒ 2.6.3 ☐ 2.5.10 (SNAPSHOT) ☐ 2.5.9

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging

☒ Jar ☐ War

Java

☐ 17 ☒ 11 ☐ 8

Dependencies ADD DEPENDENCIES... CTRL + B

Spring Web WEB

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

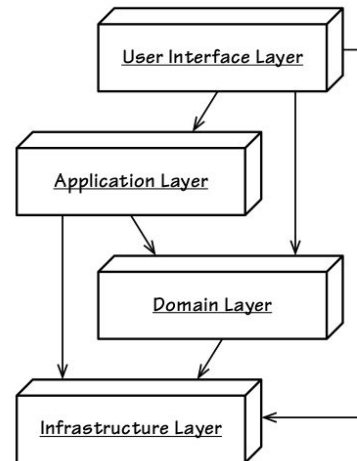
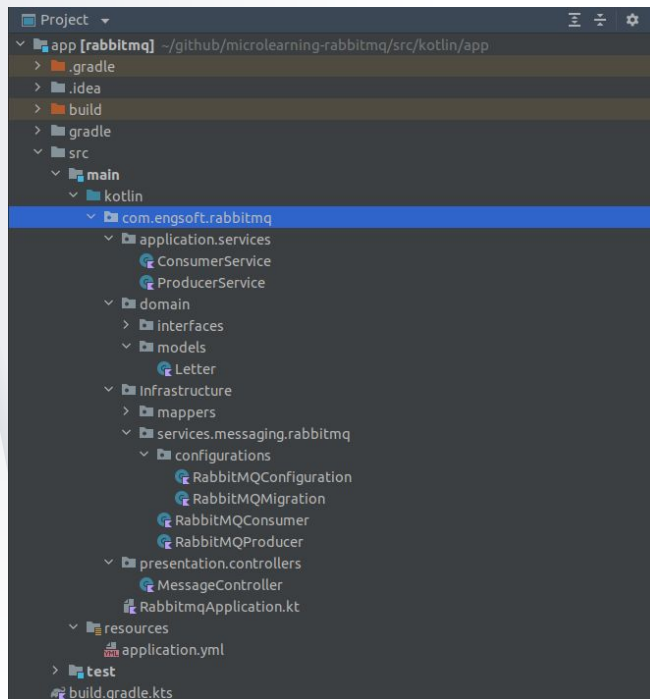
Spring for RabbitMQ MESSAGING

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

Estrutura do Projeto

<https://github.com/ricardogaldino/microlearning-rabbitmq/tree/main/src/kotlin/app>

- ▶ Siga com a criação dos pacotes (packages) e classes baseados no Domain-Driven Design (DDD).



Configuração das propriedades do RabbitMQ

app/src/main/resources/application.yml

Siga com a **configuração** das **propriedades** do RabbitMQ no arquivo “**application.yml**”.

Parâmetros do **Servidor**:

- ▶ **host**: endereço
- ▶ **port**: porta
- ▶ **username**: usuário
- ▶ **password**: senha

Parâmetros do **mecanismo de retentativas** de envio de mensagens com problemas (**retry**) :

- ▶ **initial-interval**: o envio da mensagem deve ser repetido após um intervalo de “n” segundos.
- ▶ **max-attempts**: o envio da mensagem deve ser repetido no máximo “n” vezes. Depois disso, ele será enviado para a fila de mensagens mortas (dead letter queue).
- ▶ **max-interval**: O intervalo de tempo máximo entre duas tentativas nunca deve exceder “n” segundos.
- ▶ **multiplier**: O intervalo entre a segunda tentativa é multiplicado por “n”. Mas esse intervalo nunca pode exceder o intervalo máximo.

```
application.yml
1 server:
2   port: 8087
3
4   spring:
5     rabbitmq:
6       host: localhost
7       port: 5672
8       username: guest
9       password: guest
10      listener:
11        simple:
12          retry:
13            enabled: true
14            initial-interval: 3s
15            max-attempts: 3
16            max-interval: 10s
17            multiplier: 2
18
19  app:
20    messaging:
21      rabbitmq:
22        letter:
23          exchange: "letter.exchange.direct"
24          queue: "letter.queue"
25          queue-routing-key: "letter.queue.routing.key"
26          dead-letter-queue: "letter.dead.letter.queue"
27          dead-letter-queue-routing-key: "letter.dead.letter.queue.routing.key"
```

Configuração das propriedades do RabbitMQ

app/src/main/resources/application.yml

Parâmetros das **Entidades** do RabbitMQ:

- ▶ **exchange**: nome do roteador de mensagens
- ▶ **queue**: nome da fila de mensagens
- ▶ **queue-routing-key**: chave de roteamento da fila de mensagens
- ▶ **dead-letter-queue**: nome da fila de mensagens mortas
- ▶ **dead-letter-queue-routing-key**: chave de roteamento da fila de mensagens mortas

```
application.yml
1 server:
2   port: 8087
3
4   spring:
5     rabbitmq:
6       host: localhost
7       port: 5672
8       username: guest
9       password: guest
10      listener:
11        simple:
12          retry:
13            enabled: true
14            initial-interval: 3s
15            max-attempts: 3
16            max-interval: 10s
17            multiplier: 2
18
19      app:
20        messaging:
21          rabbitmq:
22            letter:
23              exchange: "letter.exchange.direct"
24              queue: "letter.queue"
25              queue-routing-key: "letter.queue.routing.key"
26              dead-letter-queue: "letter.dead.letter.queue"
27              dead-letter-queue-routing-key: "letter.dead.letter.queue.routing.key"
```

Configuração dos Beans do RabbitMQ

Infrastructure/services/messaging/rabbitmq/configurations/RabbitMQConfiguration.kt

O RabbitMQ precisa da **declaração inicial** de alguns **beans** para funcionar (vide classe “**RabbitMQConfiguration**”):

- ▶ **RabbitAdmin**: classe para administração de operações em “**Entidades**” no RabbitMQ, tais como criação de queues, exchanges e bindings.
- ▶ **SimpleRabbitListenerContainerFactory**: classe para administração de operações de “**Consumo**” de mensagens no RabbitMQ.
- ▶ **RabbitTemplate**: classe para administração de operações de “**Publicação**” de mensagens no RabbitMQ.

*Obs: Um **bean** é um objeto que é instanciado, montado e gerenciado pelo Spring.*

```
RabbitMQConfiguration.kt
10
11 @Configuration
12 class RabbitMQConfiguration(
13     val connection: ConnectionFactory,
14     val listenerConfigurator: SimpleRabbitListenerContainerFactoryConfigurer,
15     val listener: SimpleRabbitListenerContainerFactory
16 ) {
17     @Bean
18     fun createAdmin(): RabbitAdmin {
19         return RabbitAdmin(connection)
20     }
21
22     @Bean
23     fun createListener(
24     ): SimpleRabbitListenerContainerFactory? {
25         listenerConfigurator.configure(listener, connection)
26         return listener
27     }
28
29     @Bean
30     fun createTemplate(
31     ): RabbitTemplate {
32         return RabbitTemplate(connection)
33     }
34 }
```

Criação programática das queues, exchanges e bindings

Infrastructure/services/messaging/rabbitmq/configurations/RabbitMQMigration.kt

- ▶ Através do “**RabbitAdmin**”, **módulo Spring AMQP**, podemos criar programaticamente queues, exchanges e bindings, assim que a aplicação se inicia (vide classe “**RabbitMQMigration**”).

```
private fun createDirectExchange() {  
    val directExchange = ExchangeBuilder  
        .directExchange(exchange)  
        .durable(isDurable: true)  
        .build<Exchange>()  
  
    rabbitAdmin.declareExchange(directExchange)  
}
```

```
private fun createQueue() {  
    val queue = QueueBuilder  
        .durable(queue)  
        .deadLetterExchange(exchange)  
        .deadLetterRoutingKey(deadLetterQueueRoutingKey)  
        .build()  
  
    rabbitAdmin.declareQueue(queue)  
}  
  
private fun createDeadLetterQueue() {  
    val deadLetterQueue = QueueBuilder  
        .durable(deadLetterQueue)  
        .build()  
  
    rabbitAdmin.declareQueue(deadLetterQueue)  
}
```

```
private fun createQueueBinding() {  
    val binding = Binding(  
        queue,  
        Binding.DestinationType.QUEUE,  
        exchange,  
        queueRoutingKey,  
        arguments: null  
    )  
  
    rabbitAdmin.declareBinding(binding)  
}  
  
private fun createDeadLetterQueueBinding() {  
    val binding = Binding(  
        deadLetterQueue,  
        Binding.DestinationType.QUEUE,  
        exchange,  
        deadLetterQueueRoutingKey,  
        arguments: null  
    )  
  
    rabbitAdmin.declareBinding(binding)  
}
```

Criação da Infraestrutura para o Producer

Infrastructure/services/messaging/rabbitmq/RabbitMQProducer.kt

- ▶ Através do “**RabbitTemplate**”, módulo **Spring AMQP**, podemos **enviar a mensagem** ao RabbitMQ, **passando a exchange**, a **routing key** e a própria **mensagem** (vide classe “**RabbitMQProducer**”).

```
@Service
public class RabbitMQProducer(
    private val template: RabbitTemplate,
    private val messageMapper: BaseMapper<Letter, Message>,
    @Value("\${app.messaging.rabbitmq.letter.exchange}") private val exchange,
    @Value("\${app.messaging.rabbitmq.letter.queue-routing-key}") private val queueRoutingKey
) : AMQPProducer {
    public override fun produce(letter: Letter) {
        template.convertAndSend(
            exchange,
            queueRoutingKey,
            messageMapper.from(letter)
        )
    }
}
```

Criação da Infraestrutura para o Consumer

Infrastructure/services/messaging/rabbitmq/RabbitMQConsumer.kt

- ▶ Através da **annotation** “**@RabbitListener**”, **módulo Spring AMQP**, podemos **consumir mensagens de filas** do RabbitMQ, **passando** como **parâmetro** uma ou mais **queues** (vide classe “**RabbitMQConsumer**”).

```
@Service
class RabbitMQConsumer(
    val consumerService: ConsumerService,
    val letterMapper: BaseMapper<Message, Letter>
) : AMQPConsumer {
    @RabbitListener(id = "letter-queue", queues = ["letter.queue"])
    override fun consume(message: Message) {
        consumerService.consume(
            letterMapper.from(message)
        )
    }
}
```


Criação do Serviço para o Producer

application/services/ProducerService.kt

- ▶ O serviço **encapsula** a chamada do módulo Spring AMQP, **classe** “**RabbitMQProducer**”, da camada de **Infraestrutura**.

```
@Service
class ProducerService(
    val producer: AMQPProducer
) : ProducerService {
    override fun produce(letter: Letter) {
        producer.produce(letter)
    }
}
```

Criação do Serviço para o Consumer

application/services/ConsumerService.kt

- ▶ O serviço **recebe** e **imprime** a mensagem do RabbitMQ, vinda da **classe “RabbitMQConsumer”**, da camada de **Infraestrutura**.

```
@Service
class ConsumerService : ConsumerService {
    override fun consume(letter: Letter) {
        println("===== LETTER =====")
        println(letter.text)
        println("=====")
    }
}
```

Envio direto de mensagens com erro para Dead Letter Queue

infrastructure/mappers/LetterMapper.kt

- ▶ Ao tentar consumir uma **mensagem inválida** podemos **lançar** a exception “**AmqpRejectAndDontRequeueException**” a qual fará com que a **mensagem não passe pelo mecanismo de “retry”** (retentativas de envio) e **vá diretamente para a Fila de Mensagens Mortas** (Dead Letter Queue).

```
class LetterMapper() : BaseMapper<Message, Letter> {
    override fun from(message: Message): Letter {
        try {
            val payload = ObjectMapper().readTree(message.body)

            val letter = Letter(
                payload.get("text").toString()
            )

            return letter
        } catch (ex: Exception) {
            // Envia a mensagem para a Dead Letter Queue!
            throw AmqpRejectAndDontRequeueException(ex)
        }
    }
}
```

Controlador para Publicação da Mensagem

presentation/controllers/MessageController.kt

- ▶ O controller (API) **receberá a requisição do envio de mensagem** e **passará para o serviço de producer**.

```
@RestController
@ResponseStatus(HttpStatus.ACCEPTED)
@RequestMapping("/message")
class MessageController(
    val producerService: ProducerService
) {
    @PostMapping("publish")
    fun publish(@RequestBody letter: Letter) {
        producerService.produce(letter)
    }
}
```

Executando a Aplicação

Spring + RabbitMQ

Iniciando o RabbitMQ via Docker Compose

- ▶ Faça o **download** do projeto no **GitHub**:
<https://github.com/ricardogaldino/microlearning-rabbitmq/>
- ▶ Abra o **terminal** e navegue até a pasta **"/docker"**
- ▶ Execute o **comando**:

```
$ docker-compose -f docker-compose.yml up
```

```
rabbitmq | 2022-01-21 12:40:27.931668+00:00 [info] <0.222.0>
rabbitmq | 2022-01-21 12:40:27.931668+00:00 [info] <0.222.0> Starting RabbitMQ 3.9.12 on Erlang 24.2 [jit]
rabbitmq | 2022-01-21 12:40:27.931668+00:00 [info] <0.222.0> Copyright (c) 2007-2022 VMware, Inc. or its affiliates.
rabbitmq | 2022-01-21 12:40:27.931668+00:00 [info] <0.222.0> Licensed under the MPL 2.0. Website: https://rabbitmq.com
rabbitmq |
rabbitmq | ## ## RabbitMQ 3.9.12
rabbitmq | ## ##
rabbitmq | ##### Copyright (c) 2007-2022 VMware, Inc. or its affiliates.
rabbitmq | #####
rabbitmq | ##### Licensed under the MPL 2.0. Website: https://rabbitmq.com
rabbitmq |
rabbitmq | Erlang: 24.2 [jit]
rabbitmq | TLS Library: OpenSSL - OpenSSL 1.1.1m 14 Dec 2021
rabbitmq |
rabbitmq | Doc guides: https://rabbitmq.com/documentation.html
rabbitmq | Support: https://rabbitmq.com/contact.html
rabbitmq | Tutorials: https://rabbitmq.com/getstarted.html
rabbitmq | Monitoring: https://rabbitmq.com/monitoring.html
rabbitmq |
rabbitmq | Logs: /var/log/rabbitmq/rabbit@rabbitmq_upgrade.log
rabbitmq | <stdout>
rabbitmq |
rabbitmq | Config file(s): /etc/rabbitmq/conf.d/10-default-guest-user.conf
rabbitmq |
rabbitmq | Starting broker...2022-01-21 12:40:27.932394+00:00 [info] <0.222.0>
rabbitmq | 2022-01-21 12:40:27.932394+00:00 [info] <0.222.0> node : rabbit@rabbitmq
rabbitmq | 2022-01-21 12:40:27.932394+00:00 [info] <0.222.0> home dir : /var/lib/rabbitmq
rabbitmq | 2022-01-21 12:40:27.932394+00:00 [info] <0.222.0> config file(s) : /etc/rabbitmq/conf.d/10-default-guest-user.conf
rabbitmq | 2022-01-21 12:40:27.932394+00:00 [info] <0.222.0> cookie hash : jevuNPhWuNmZEscpPUB+Fg==
```

Subindo a Aplicação (API)

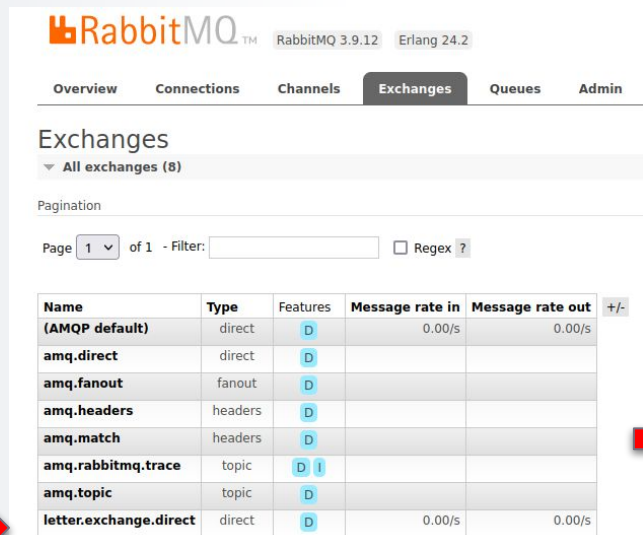
- ▶ Abra o **terminal** e navegue até a pasta **“/src/kotlin/app”**
- ▶ Execute o **comando**:

```
$ ./gradlew bootRun
```

[illegible]

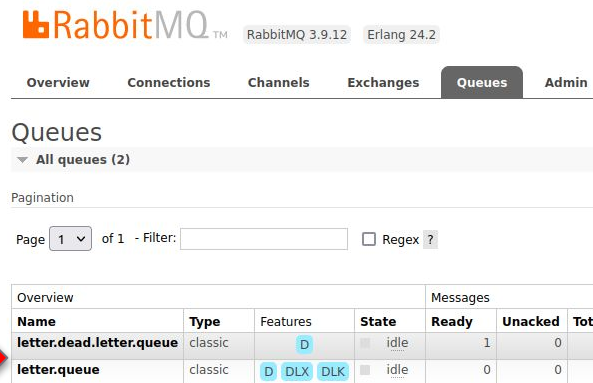
Observe a criação automática das queues, exchanges e bindings!

- Assim que a aplicação roda são criadas automaticamente no RabbitMQ as queues, exchanges e bindings.



The screenshot shows the RabbitMQ web interface with the 'Exchanges' tab selected. The page title is 'Exchanges' and it shows 'All exchanges (8)'. The pagination is 'Page 1 of 1 - Filter:'. Below the pagination is a table with columns: Name, Type, Features, Message rate in, and Message rate out. The table lists several exchanges, including 'letter.exchange.direct' which is highlighted with a red arrow.

Name	Type	Features	Message rate in	Message rate out
(AMQP default)	direct	D	0.00/s	0.00/s
amq.direct	direct	D		
amq.fanout	fanout	D		
amq.headers	headers	D		
amq.match	headers	D		
amq.rabbitmq.trace	topic	D I		
amq.topic	topic	D		
letter.exchange.direct	direct	D	0.00/s	0.00/s



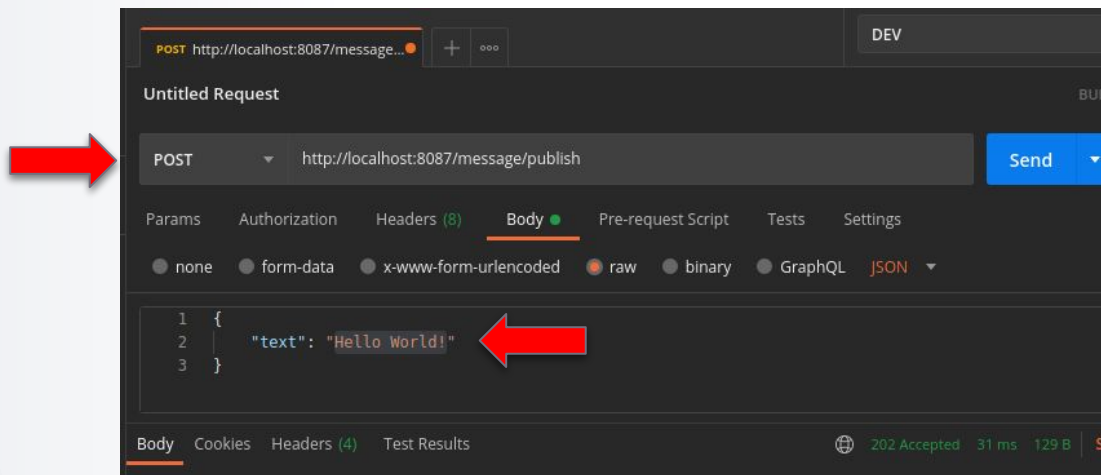
The screenshot shows the RabbitMQ web interface with the 'Queues' tab selected. The page title is 'Queues' and it shows 'All queues (2)'. The pagination is 'Page 1 of 1 - Filter:'. Below the pagination is a table with columns: Name, Type, Features, State, Ready, Unacked, and Tot. The table lists two queues, including 'letter.queue' which is highlighted with a red arrow.

Name	Type	Features	State	Ready	Unacked	Tot
letter.dead.letter.queue	classic	D	idle	1	0	
letter.queue	classic	D DLX DLK	idle	0	0	

Solicitando o envio da mensagem pelo Postman

<https://www.postman.com/>

- ▶ Postman é uma ferramenta que tem como objetivo **testar serviços RESTful (Web APIs)** por meio do envio de requisições HTTP.
- ▶ Configure e envie uma requisição HTTP pela ferramenta “**Postman**”:
 - ▶ Method: **POST**
 - ▶ URL: <http://localhost:8087/message/publish>
 - ▶ Formato: **JSON**
- ▶ O controller receberá a **requisição do envio de mensagem**, via **Postman**, e passará para o serviço de producer.



► O serviço **recebe** a mensagem do RabbitMQ, **pelo consumer “RabbitMQConsumer”**, e **imprime** na tela.

```
> Task :bootRun

  ____  _
 / ___|| | | |
| |___| | | |
 \___ \| | | |
  ___) | | | |
 / ___|| | | |
| |___| | | |
 \___) |_| |_|

=====|_|=====|_|_/|_|_|_|
:: Spring Boot ::                (v2.6.2)

2022-01-25 01:24:07.026 INFO 8747 --- [main] c.e.rabbitmq.RabbitmqApplicationKt : Starting Rab
ing-rabbitmq/src/kotlin/app/build/classes/kotlin/main started by CIANDT\rluna in /home/rluna/github/microlearnin
2022-01-25 01:24:07.028 INFO 8747 --- [main] c.e.rabbitmq.RabbitmqApplicationKt : No active pr
2022-01-25 01:24:07.721 INFO 8747 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initi
2022-01-25 01:24:07.729 INFO 8747 --- [main] o.apache.catalina.core.StandardService : Starting ser
2022-01-25 01:24:07.729 INFO 8747 --- [main] org.apache.catalina.core.StandardEngine : Starting Ser
2022-01-25 01:24:07.793 INFO 8747 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing
2022-01-25 01:24:07.793 INFO 8747 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebAppl
2022-01-25 01:24:08.053 INFO 8747 --- [main] o.s.a.r.c.CachingConnectionFactory : Attempting to
2022-01-25 01:24:08.074 INFO 8747 --- [main] o.s.a.r.c.CachingConnectionFactory : Created new
0.0.1:5672/, localPort= 38532]
2022-01-25 01:24:08.312 INFO 8747 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat start
2022-01-25 01:24:08.333 INFO 8747 --- [main] c.e.rabbitmq.RabbitmqApplicationKt : Started Rabb
2022-01-25 01:24:52.138 INFO 8747 --- [nio-8087-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing
2022-01-25 01:24:52.138 INFO 8747 --- [nio-8087-exec-1] o.s.web.servlet.DispatcherServlet : Initializing
2022-01-25 01:24:52.139 INFO 8747 --- [nio-8087-exec-1] o.s.web.servlet.DispatcherServlet : Completed in

===== LETTER =====
"Hello World!"

=====
<-----> 83% EXECUTING [6m 35s]
> :bootRun
```

TO BE CONTINUED...

<https://github.com/ricardogaldino/microlearning-rabbitmq/tree/main/docs>

