

Topic

Topic exchange

Mensagens enviadas para uma exchange do tipo **topic** não podem ter uma lista arbitrária de **routing_key**, deve ser uma lista de palavras, separadas por **pontos(.)**. Podem ser utilizadas quaisquer palavras, mas usualmente é utilizado algumas palavras que tenham relação com as mensagens.

Alguns exemplos de **routing-keys** são: "**stock.usd.nyse**", "**nyse.vmw**", "**quick.orange.rabbit**", pode-se utilizar várias palavras como **routing-key** mas respeitando o limite de 256 bytes.

A **binding-key** também deve seguir o mesmo formato. A lógica por trás da exchange do tipo **topic** é similar a do tipo **direct**, ou seja, a mensagem enviada com uma **routing-key** em particular será entregue para todas as **queues** que estão vinculadas com a **binding-key** correspondente.

Contudo existe 2 casos especiais sobre **binding-keys**

- ***** (**star**) pode substituir exatamente **uma palavra**.
- **#** (**hash**) pode substituir **zero ou mais palavras**.

Exemplo

Para gerar o projeto podem acessar a [URL](#), nessa url iremos utilizar o **Spring Initializr** para gerar a estrutura padrão do projeto já adicionando a biblioteca do RabbitMQ.

The image shows the Spring Initializr web interface. The 'Project' section has 'Maven Project' selected. The 'Language' section has 'Java' selected. The 'Spring Boot' section has '2.3.3' selected. The 'Project Metadata' section shows the following fields: Group (br.com.facef), Artifact (rabbitmq-topic), Name (rabbitmq-topic), Description (Spring boot application to demonstrate topic in rabbitmq), Package name (br.com.facef.rabbitmqtopic), Packaging (Jar), and Java version (11). The 'Dependencies' section shows 'Spring for RabbitMQ' (Messaging), 'Spring Web' (Web), and 'Lombok' (Developer Tools) as selected dependencies. At the bottom, there are buttons for 'GENERATE', 'EXPLORE', and 'SHARE...'.

Primeiramente precisamos iniciar o RabbitMQ:

- Iniciando diretamente via docker

```
docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3-management
```

- Iniciando via docker-compose

```
docker-compose stop && docker-compose rm -f && docker-compose up -d
```

- Iniciando via helm + microk8s

```
microk8s helm3 install rabbitmq stable/rabbitmq
```

Para acompanhar a inicialização do container pode-se utilizar o comando:

```
docker-compose logs -f
```


Verificando os logs:

```
rabbitmq_1 | 2020-08-25 22:59:22.925 [info] <0.675.0> Ready to start client connection listeners
rabbitmq_1 | 2020-08-25 22:59:22.930 [info] <0.980.0> started TCP listener on [::]:5672
rabbitmq_1 | 2020-08-25 22:59:23.273 [info] <0.675.0> Server startup complete; 4 plugins started.
rabbitmq_1 | * rabbitmq_prometheus
rabbitmq_1 | * rabbitmq_management
rabbitmq_1 | * rabbitmq_web_dispatch
rabbitmq_1 | * rabbitmq_management_agent
rabbitmq_1 | completed with 4 plugins.
```

Após o início do container, podemos acessar a URL do admin através do endereço <http://localhost:15672/>

username: guest

password: guest



Username: *

Password: *

Login

Após o login temos acesso a interface de gerenciamento do servidor do rabbitmq, onde temos as visões referentes há:

- **Overview**
- **Connections**
- **Channels**
- **Exchanges**
- **Queues**
- **Admin**

Ao clicar em **exchanges** podemos visualizar as exchanges padrões que o próprio servidor o rabbitmq cria quando iniciamos o serviço. E temos a opção de criar nossas próprias exchanges através do admin.

Exchanges

▼ All exchanges (7)

Pagination

Page of 1 - Filter: ☐ Regex ?

Name	Type	Features	Message rate in	Message rate out	+/-
(AMQP default)	direct	D			
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			

▶ Add a new exchange

Clicando em **queues** podemos visualizar que por default nenhuma fila é criada no momento da inicialização do servidor.

Queues

▼ All queues (0)

Pagination

Page of 0 - Filter: ☐ Regex ?

... no queues ...

► Add a new queue

HTTP API

Server Docs

Tutorials

Community Support

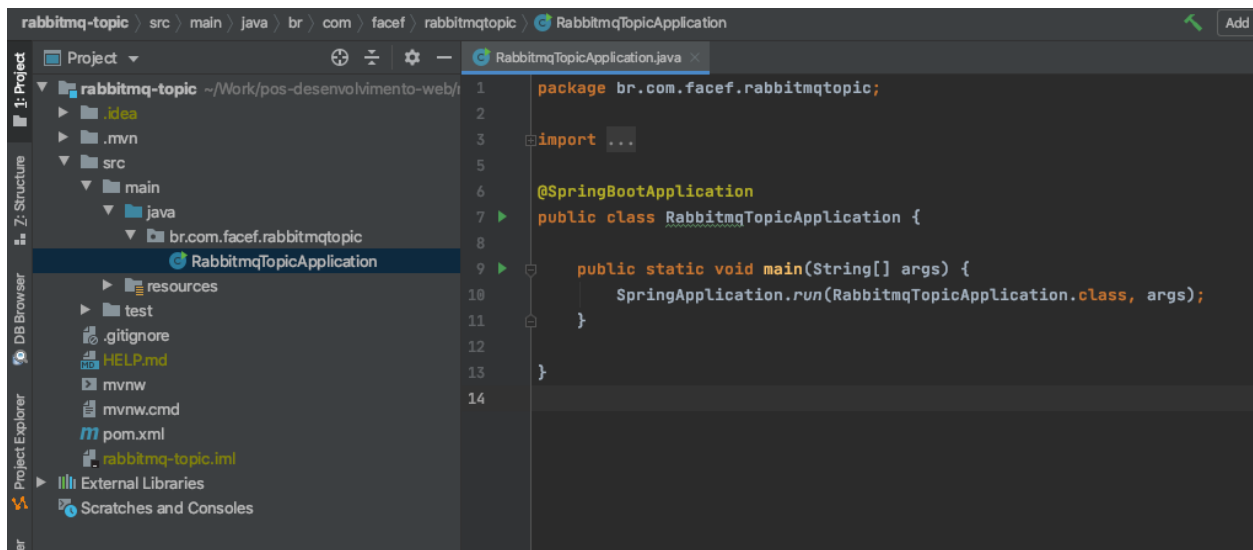
Community Slack

Commei

Para demonstrar o funcionamento, irei fazer um passo a passo com um exemplo funcional, implementando uma API Rest que recebe uma mensagem via POST com um body que será enviado para a exchange criada, e com base no dado enviado no body a mensagem será roteada para a queue em específico.

Passo a passo:

- **Initial project**



- **Include docker-compose**

```

version: '3'
services:
  rabbitmq:
    image: rabbitmq:3-management
    ports:
      - "5672:5672"
      - "15672:15672"

```

- **Include rabbitmq configuration to TopicExchange**

```

package br.com.facef.rabbitmqtopic.configuration;

import org.springframework.amqp.core.Binding;
import org.springframework.amqp.core.BindingBuilder;
import org.springframework.amqp.core.ExchangeBuilder;
import org.springframework.amqp.core.Queue;
import org.springframework.amqp.core.TopicExchange;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class TopicExchangeConfiguration {

    public static final String TOPIC_EXCHANGE_NAME = "city-exchange";
    public static final String SMALL_CITIES_QUEUE_NAME = "small-cities-queue";
    public static final String MEDIUM_BIG_CITIES_QUEUE_NAME = "medium-big-cities-queue";
    public static final String ALL_CITIES_QUEUE_NAME = "all-cities-queue";

    @Bean
    Queue smallCitiesQueue() {
        return new Queue(SMALL_CITIES_QUEUE_NAME);
    }
}

```

```

@Bean
Queue mediumBigCitiesQueue() {
    return new Queue(MEDIUM_BIG_CITIES_QUEUE_NAME);
}

@Bean
Queue allCitiesQueue() {
    return new Queue(ALL_CITIES_QUEUE_NAME);
}

@Bean
TopicExchange exchange() {
    return ExchangeBuilder.topicExchange(TOPIC_EXCHANGE_NAME).durable(true).build();
}

@Bean
Binding bindingSmallCitiesQueue(
    @Qualifier("smallCitiesQueue") Queue queue, TopicExchange exchange) {
    return BindingBuilder.bind(queue).to(exchange).with("*.small");
}

@Bean
Binding bindingMediumCitiesQueue(
    @Qualifier("mediumBigCitiesQueue") Queue queue, TopicExchange exchange) {
    return BindingBuilder.bind(queue).to(exchange).with("*.medium");
}

@Bean
Binding bindingBigCitiesQueue(
    @Qualifier("mediumBigCitiesQueue") Queue queue, TopicExchange exchange) {
    return BindingBuilder.bind(queue).to(exchange).with("*.big");
}

@Bean
Binding bindingAllCitiesQueue(@Qualifier("allCitiesQueue") Queue queue, TopicExchange exchange) {
    return BindingBuilder.bind(queue).to(exchange).with("City.#");
}
}

```

- **Include DTO City class to store data**

```

package br.com.facef.rabbitmqtopic.dto;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.ToString;

@AllArgsConstructor
@Getter
@ToString
public class City {

    private String name;
    private Long population;
}

```

- **Include service class to send message to rabbitmq**

```
package br.com.facef.rabbitmqtopic.service;

import br.com.facef.rabbitmqtopic.configuration.TopicExchangeConfiguration;
import br.com.facef.rabbitmqtopic.dto.City;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class CityService {

    @Autowired private RabbitTemplate rabbitTemplate;

    public void sendToTopicExchange(City city) {
        try {
            final var messageJson = new ObjectMapper().writeValueAsString(city);

            rabbitTemplate.convertAndSend(
                TopicExchangeConfiguration.TOPIC_EXCHANGE_NAME, getRoutingKey(city), messageJson);
        } catch (JsonProcessingException e) {
            throw new RuntimeException(e);
        }
    }

    private String getRoutingKey(City city) {
        if (city.getPopulation() <= 10000) {
            return "city.small";
        } else if (city.getPopulation() > 10000 && city.getPopulation() < 500000) {
            return "city.medium";
        } else {
            return "city.big";
        }
    }
}
```

- **Create a controller to receive message and send to a rabbitmq**

```
package br.com.facef.rabbitmqtopic.controller;

import br.com.facef.rabbitmqtopic.dto.City;
import br.com.facef.rabbitmqtopic.service.CityService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/routing")
public class CityController {
```



```

@Autowired private CityService cityService;

@PostMapping("/topic")
public ResponseEntity placeOrder(@RequestBody City city) {
    cityService.sendToTopicExchange(city);
    return ResponseEntity.accepted().build();
}
}

```

Após a finalização da implementação do projeto iremos executar o projeto utilizando o próprio plugin do spring.

```
./mvnw clean spring-boot:run
```

E com isso iremos realizar os testes via **Postman** ou **Curl** chamando a API para inclusão das mensagens nas filas e verificar o comportamento do roteamento.

Realizando uma requisição com dados de cidade pequena

```

curl --location --request POST 'http://localhost:8080/routing/topic' \
--header 'Content-Type: application/json' \
--data-raw '{
    "name": "Restinga",
    "population": 7000
}'

```

Realizando uma requisição com dados de cidade média

```

curl --location --request POST 'http://localhost:8080/routing/topic' \
--header 'Content-Type: application/json' \
--data-raw '{
    "name": "Franca",
    "population": 450000
}'

```


Realizando uma requisição com dados de cidade grande

```

curl --location --request POST 'http://localhost:8080/routing/topic' \
--header 'Content-Type: application/json' \
--data-raw '{
    "name": "São Paulo",
    "population": 11000000
}'

```

Após realizarmos a primeira chamada via API, podemos verificar que temos uma nova exchange criada chamada **city-exchange**.

 RabbitMQTM RabbitMQ 3.8.7 Erlang 23.0.3

[Overview](#) [Connections](#) [Channels](#) [Exchanges](#) [Queues](#) [Admin](#)

Exchanges

▼ All exchanges (8)

Page 1 of 1 - Filter: ☐ Regex ?

Name	Type	Features	Message rate in	Message rate out	+/-
(AMQP default)	direct	D			
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			
city-exchange	topic	D	0.00/s	0.00/s	

► Add a new exchange

[HTTP API](#) [Server Docs](#) [Tutorials](#) [Community Support](#) [Community Slack](#) [Commercial Support](#)

E também temos a visão das filas que foram criadas para a exchange **city-exchange** já com as mensagens que enviamos via API.

Queues

▼ All queues (3)

Pagination

Page 1 of 1 - Filter: ☐ Regex ?

Overview				Messages			Message rates			+/-
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
all-cities-queue	classic	D	idle	3	0	3	0.00/s			
medium-big-cities-queue	classic	D	idle	2	0	2	0.00/s			
small-cities-queue	classic	D	idle	1	0	1	0.00/s			

▶ Add a new queue

Para demonstrar o funcionamento do consumo em cada **queue**, iremos criar uma classe reponsável por fazer o consumo das mensagens e imprimir no console o log com o seu conteúdo.

- **Create a consumer to processing messages**

```
package br.com.facef.rabbitmqtopic.consumer;

import br.com.facef.rabbitmqtopic.configuration.TopicExchangeConfiguration;
import lombok.extern.slf4j.Slf4j;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

@Component
@Slf4j
public class MessageConsumer {

    @RabbitListener(queues = TopicExchangeConfiguration.SMALL_CITIES_QUEUE_NAME)
    public void consumeSmallCitiesQueue(Message message) {
        log.info("Message processed from small-cities-queue {}", new String(message.getBody()));
    }

    @RabbitListener(queues = TopicExchangeConfiguration.MEDIUM_BIG_CITIES_QUEUE_NAME)
    public void consumeMediumBigCitiesQueue(Message message) {
        log.info("Message processed from medium-big-cities-queue {}", new String(message.getBody()));
    }

    @RabbitListener(queues = TopicExchangeConfiguration.ALL_CITIES_QUEUE_NAME)
    public void consumeAllCitiesQueue(Message message) {
        log.info("Message processed from all-cities-queue {}", new String(message.getBody()));
    }
}
```

Na saída do console conseguimos identificar o consumo das mensagens corretamente por fila:

```

Spring
:: Spring Boot :: (v2.3.3.RELEASE)

2020-08-26 23:45:15.604 INFO 3809 --- [main] b.c.f.r.RabbitmqTopicApplication : Starting RabbitmqTopicApplication on Cassios-MacBook-Air.local with PID 3809 (/Users/cassiothadeu/Work/pos-desenvolvimento-web/messengeria-streams/rabbitmq-topic/target/classes started by cassiothadeu in /Users/cassiothadeu/Work/pos-desenvolvimento-web/messengeria-streams/rabbitmq-topic)
2020-08-26 23:45:15.609 INFO 3809 --- [main] b.c.f.r.RabbitmqTopicApplication : No active profile set, falling back to default profiles: default
2020-08-26 23:45:17.782 INFO 3809 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2020-08-26 23:45:17.725 INFO 3809 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2020-08-26 23:45:17.725 INFO 3809 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.37]
2020-08-26 23:45:17.877 INFO 3809 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2020-08-26 23:45:17.878 INFO 3809 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2166 ms
2020-08-26 23:45:18.551 INFO 3809 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2020-08-26 23:45:19.000 INFO 3809 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2020-08-26 23:45:19.004 INFO 3809 --- [main] o.s.a.r.c.CachingConnectionFactory : Attempting to connect to: [localhost:5672]
2020-08-26 23:45:19.192 INFO 3809 --- [main] o.s.a.r.c.CachingConnectionFactory : Created new connection: rabbitConnectionFactory#61f39bb0/SimpleConnection@554c4eaa [delegate=amqp://guest@127.0.0.1:5672/, local
Port: 53071]
2020-08-26 23:45:19.393 INFO 3809 --- [ntContainerR0-1] b.c.f.r.consumer.MessageConsumer : Message processed from small-cities-queue {"name":"Restinga","population":7000}
2020-08-26 23:45:19.400 INFO 3809 --- [ntContainerR1-1] b.c.f.r.consumer.MessageConsumer : Message processed from medium-big-cities-queue {"name":"Franca","population":4500000}
2020-08-26 23:45:19.409 INFO 3809 --- [ntContainerR1-1] b.c.f.r.consumer.MessageConsumer : Message processed from medium-big-cities-queue {"name":"São Paulo","population":11000000}
2020-08-26 23:45:19.477 INFO 3809 --- [ntContainerR2-1] b.c.f.r.consumer.MessageConsumer : Message processed from all-cities-queue {"name":"Restinga","population":7000}
2020-08-26 23:45:19.478 INFO 3809 --- [ntContainerR2-1] b.c.f.r.consumer.MessageConsumer : Message processed from all-cities-queue {"name":"Franca","population":4500000}
2020-08-26 23:45:19.479 INFO 3809 --- [ntContainerR2-1] b.c.f.r.consumer.MessageConsumer : Message processed from all-cities-queue {"name":"São Paulo","population":11000000}
2020-08-26 23:45:19.503 INFO 3809 --- [main] b.c.f.r.RabbitmqTopicApplication : Started RabbitmqTopicApplication in 9.843 seconds (JVM running for 10.843)
```