PROGRAMAÇÃO CONCORRENTE E DISTRIBUÍDA

Mensageria e AMQP

Aula 12 — Padrões de mensagens e RabbitMQ na prática

Prof. Newton S. B. Miyoshi
newton.miyoshi@baraodemaua.br
Centro Universitário Barão de Mauá — Ribeirão Preto

PARTE 1

Comunicação Orientada a Mensagens

Sockets, padrões de troca de mensagens e pipelines.

Padrões de Mensagens

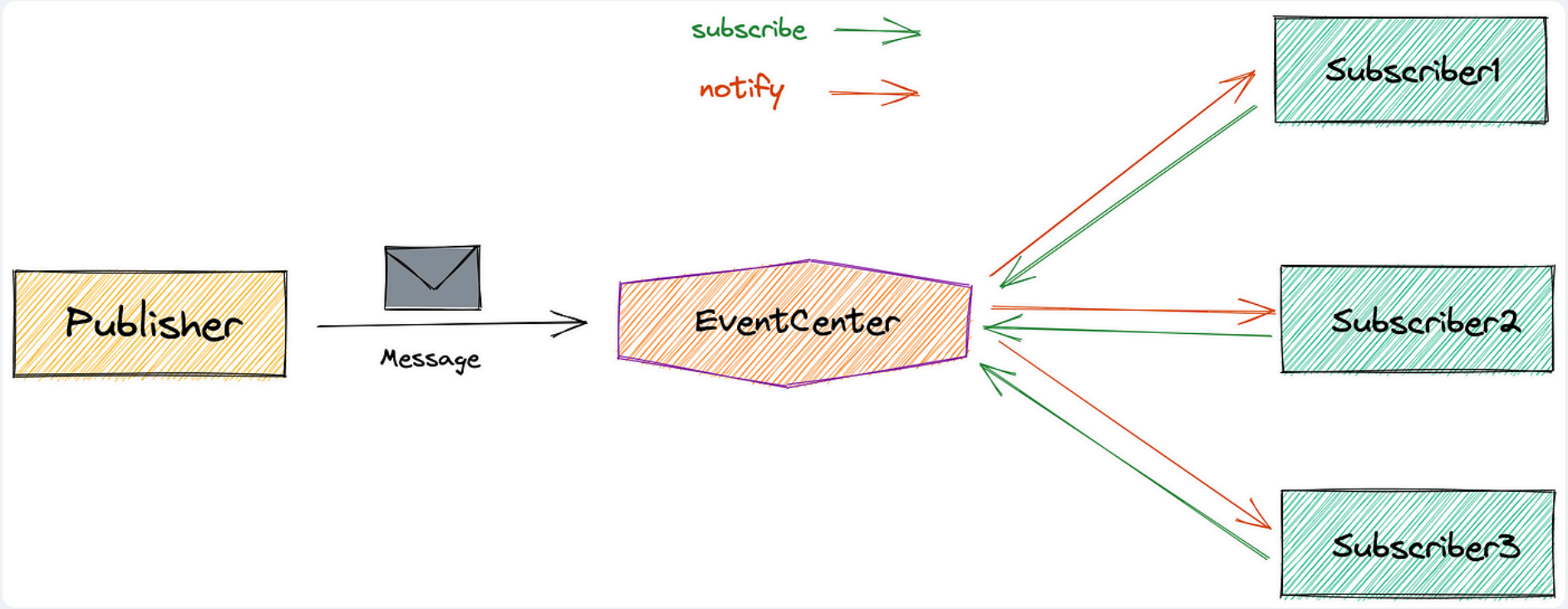
- A comunicação ocorre pelo pareamento de sockets: um tipo específico de socket para envio é pareado com o tipo correspondente para recebimento.
- Cada par de tipos de socket corresponde a um padrão de comunicação.
- Três padrões mais comuns:
  - request-reply
  - publish-subscribe
  - pipeline

Padrão Request-Reply

- Usado em comunicações tradicionais cliente-servidor / RPC.
- Cliente utiliza um request socket (REQ).
- Servidor utiliza um reply socket (REP).
- Não é necessário chamar as operações `listen` nem `accept`.

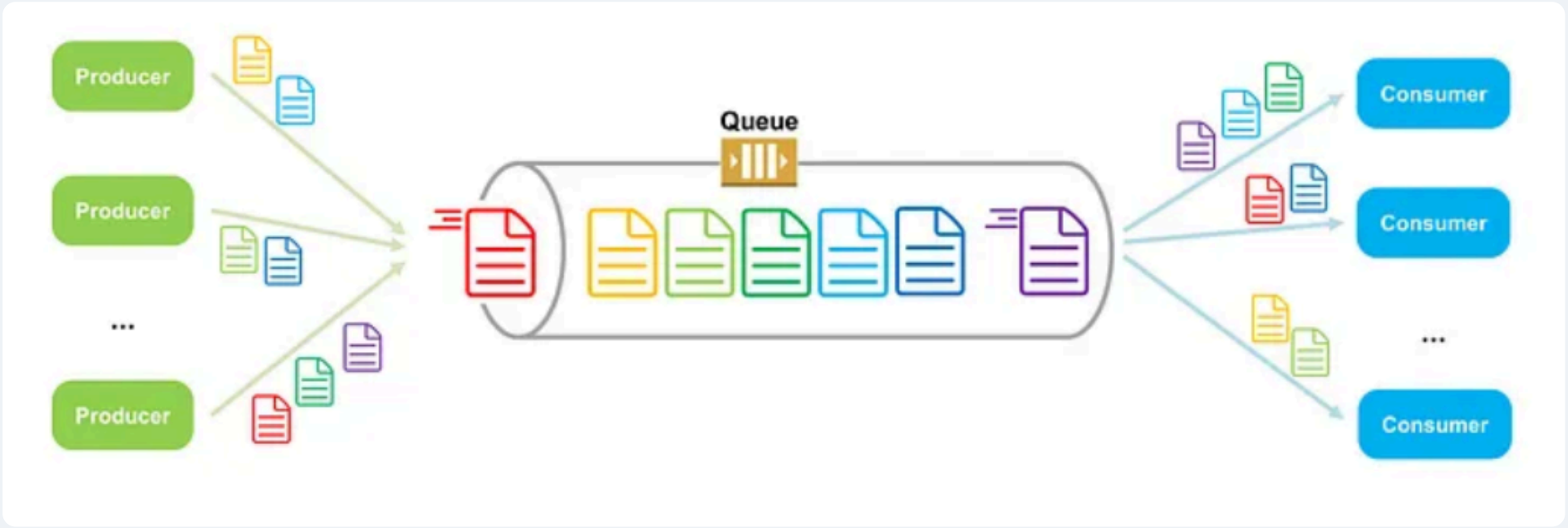
Padrão Publish-Subscribe

- Clientes subscrevem mensagens específicas que são publicadas no servidor.
- Somente as mensagens subscritas são notificadas.
- Base para sistemas orientados a eventos.
- Implementa um modelo multicast de comunicação.


subscribe
notify
Publisher
Message
EventCenter
Subscriber1
Subscriber2
Subscriber3

Padrão Pipeline

- Nós/processos empurram (push) mensagens para um duto.
- Outros nós/processos puxam (pull) mensagens do duto.
- Quem empurra não se importa quem vai puxar a mensagem — e vice-versa.
- Objetivo: performance no fluxo de mensagens.

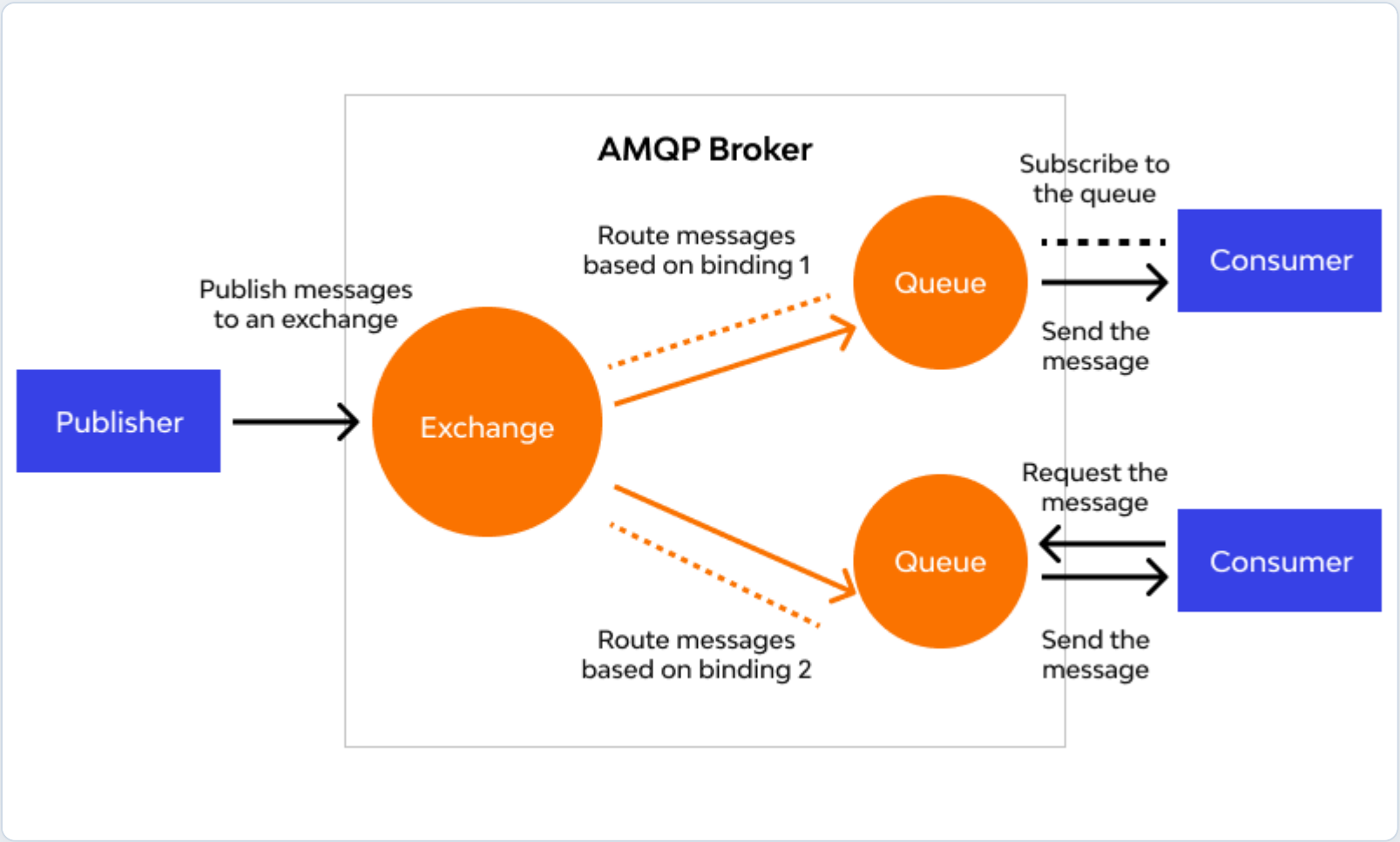

Producer
Producer
...
Producer
Queue
Consumer
Consumer
...
Consumer

PARTE 2

Python com RabbitMQ

AMQP na prática, do Hello World
aos exchanges.

AMQP — visão geral



PARA DISCUTIR

No projeto Café com Pão, quais seriam possíveis eventos gerados?

Notações — RabbitMQ

RabbitMQ — Docker

Quick reference

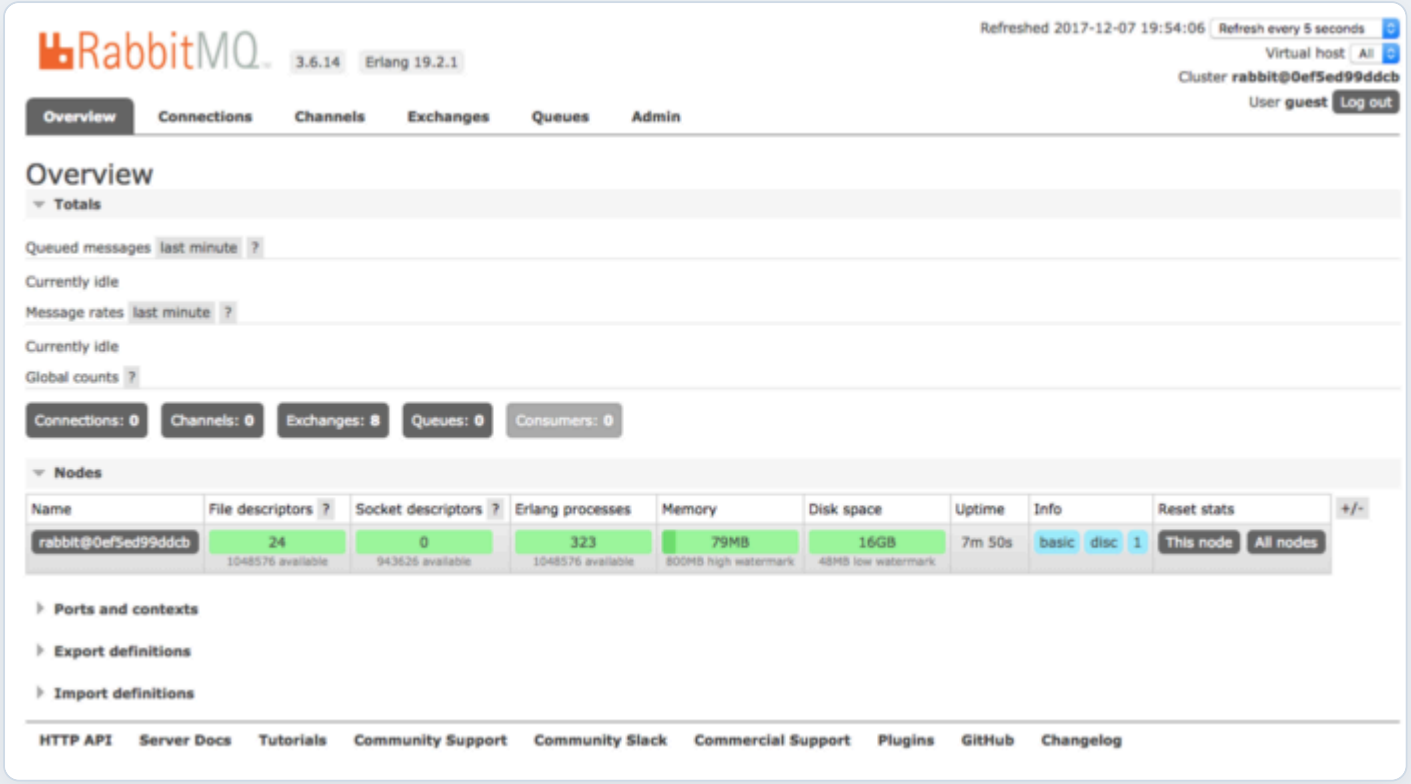
- **Maintained by**: the Docker Community
- **Where to get help**: the Docker Community Forums, the Docker Community Slack, or Stack Overflow

Supported tags and respective `Dockerfile` links

- `3.9.9`, `3.9`, `3`, `latest`
- `3.9.9-management`, `3.9-management`, `3-management`, `management`
- `3.9.9-alpine`, `3.9-alpine`, `3-alpine`, `alpine`

```
$ docker run -d --hostname rabbitmq --name rabbitmq \
  -p 5672:5672 -p 15672:15672 \
  rabbitmq:3-management
```

RabbitMQ Management



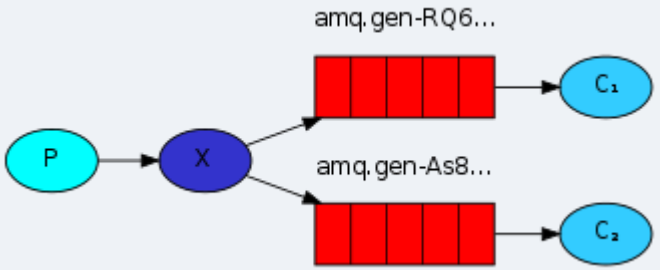
Hello World — enviando

```python
import pika

# Conectando com o rabbit
connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()

# Criando 1 fila
channel.queue_declare(queue='hello')

# Enviando 1 msg para fila
channel.basic_publish(exchange='',
                      routing_key='hello',  # nome da fila
                      body='Hello World!')
print(" [x] Sent 'Hello World!'")
connection.close()
```

Hello World — recebendo

```python
import pika

connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.queue_declare(queue='hello')

def callback(ch, method, properties, body):
    print(" [x] Received %r" % body)

channel.basic_consume(queue='hello', on_message_callback=callback, auto_ack=True)

print(' [*] Waiting for messages. To exit press CTRL+C')
channel.start_consuming()
```

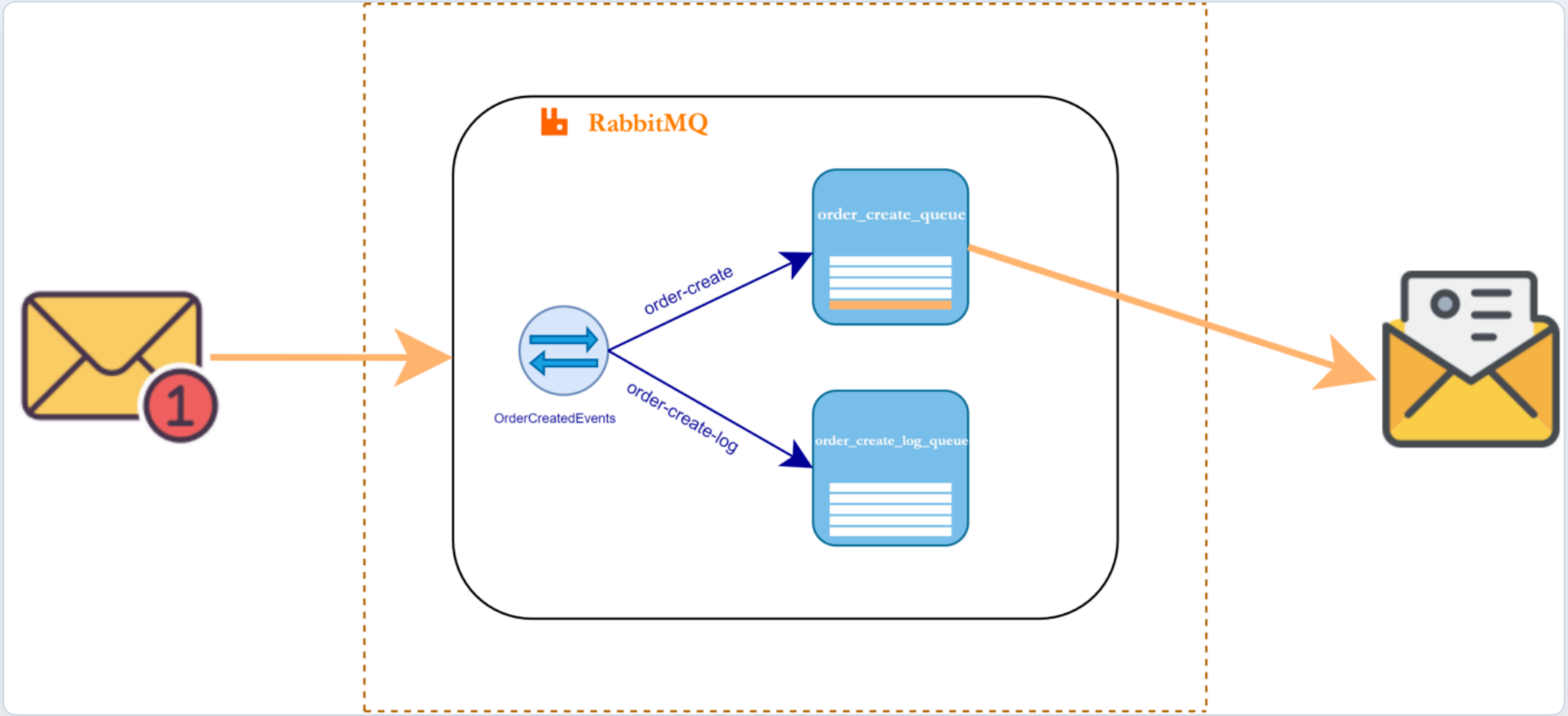
Pub-Sub com RabbitMQ



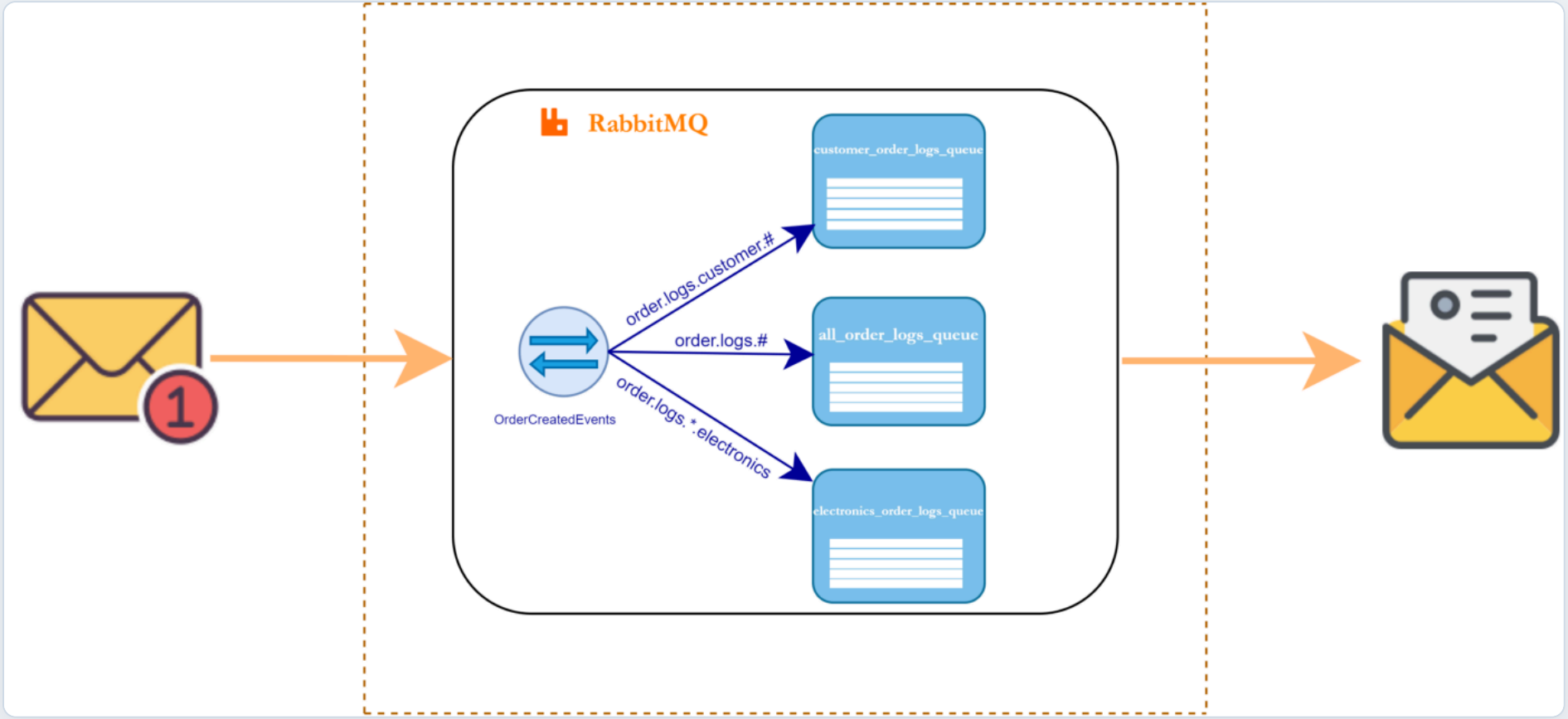
Exchanges

- Producer é a aplicação que envia mensagens; Queue é um buffer que as armazena; Consumer é quem recebe.
- O producer não envia direto para a fila — geralmente ele nem sabe se/quando a mensagem foi entregue.
- O producer envia para uma Exchange, que pode selecionar a fila via `routing_key`.
- Tipos de exchange: direct e fanout (outros: topic e header).

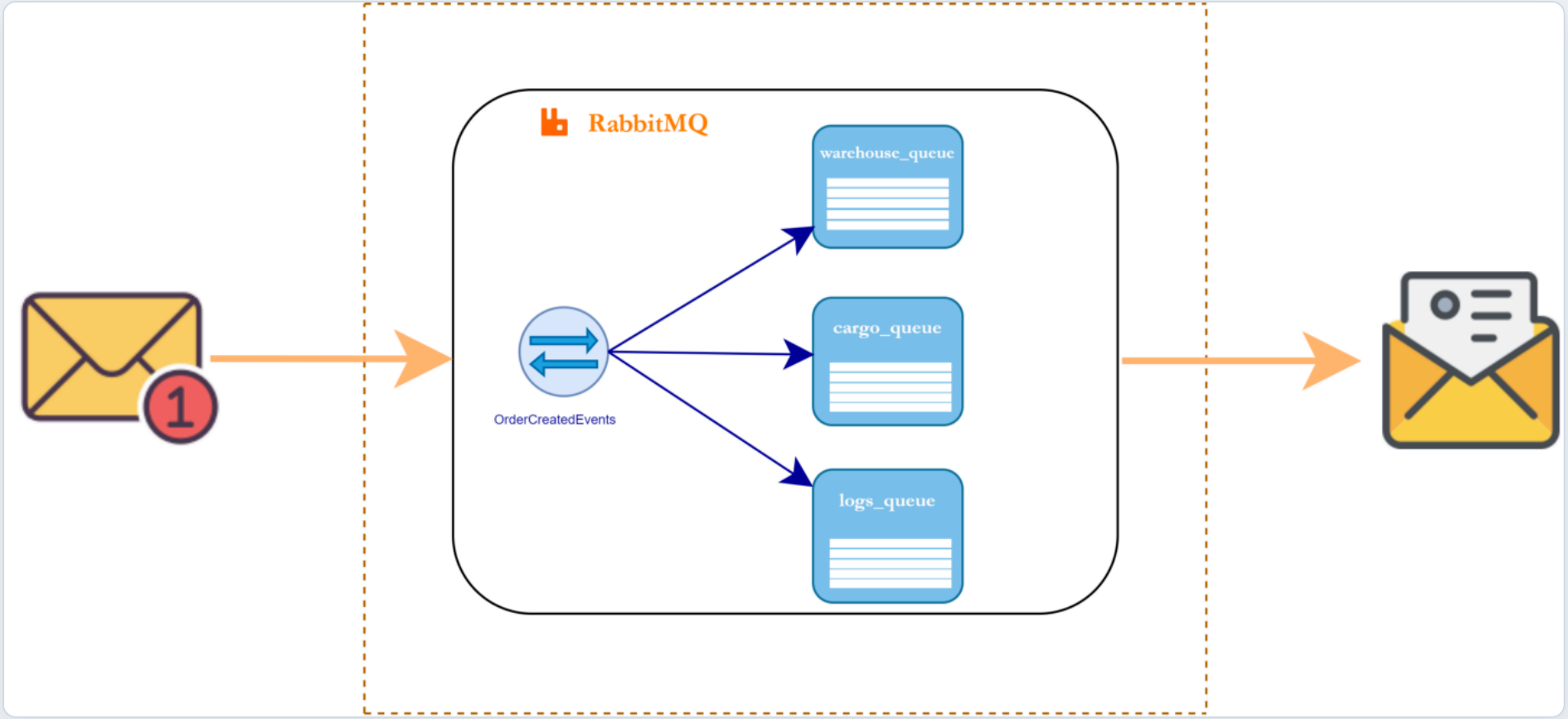
Exchange — Direct



Exchange — Topic

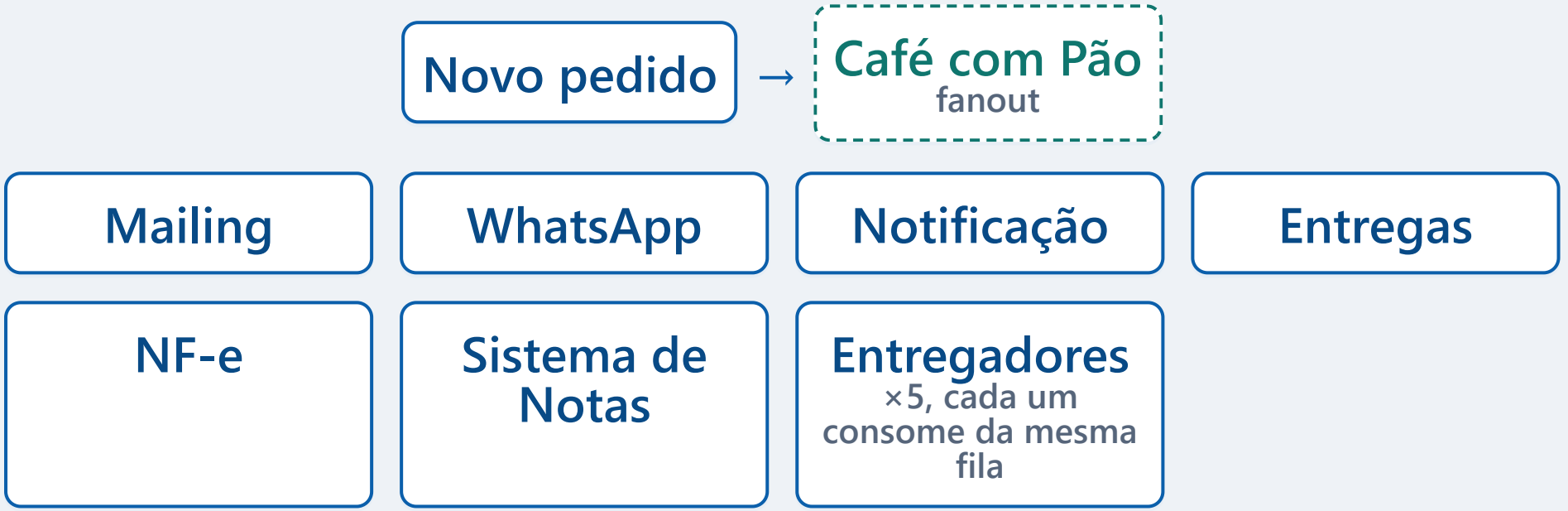


Exchange — Fanout



CAFÉ COM PÃO NA PRÁTICA

Um novo pedido, vários interessados



Um único evento `novo_pedido` publicado no fanout — cada fila decide o que fazer com ele.

Fanout — publicando logs (`emit_log.py`)

```python
import pika
import sys

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.exchange_declare(exchange='logs', exchange_type='fanout')

message = ' '.join(sys.argv[1:]) or "info: Hello World!"
channel.basic_publish(exchange='logs', routing_key='', body=message)
print(" [x] Sent %r" % message)
connection.close()
```

Declara um exchange fanout — ele faz o broadcast da mensagem para todas as filas ligadas a ele.

Fanout — assinando logs (`receive_logs.py`)

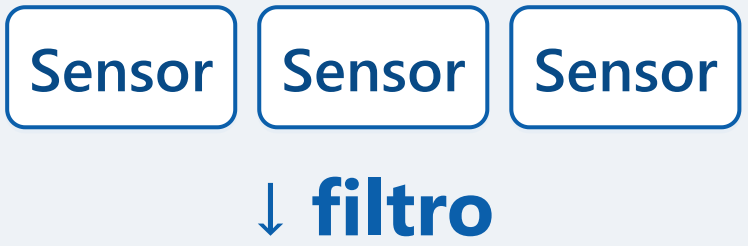
```python
import pika

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.exchange_declare(exchange='logs', exchange_type='fanout')

result = channel.queue_declare(queue='', exclusive=True)
queue_name = result.method.queue

channel.queue_bind(exchange='logs', queue=queue_name)

print(' [*] Waiting for logs. To exit press CTRL+C')
```

Uma fila com nome aleatório e temporário (`exclusive=True`) é criada e ligada (binding) ao exchange.

PRATIQUE SEM INSTALAR NADA

SIMULADOR **RabbitMQ Simulator** tryrabbitmq.com

Direct × Fanout — uma rede de sensores



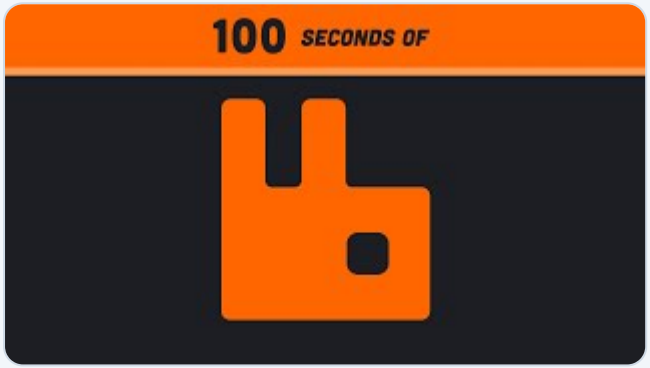
Direct

`routing_key: alerta` — só os alarmes chegam ao Monitor.

Fanout

Toda leitura chega ao Monitor e ao Climatizador.

MATERIAL EM VÍDEO



YOUTUBE **100 Seconds of RabbitMQ — Fireship** youtube.com/watch?v=NQ3fZtyXji0

MATERIAL EM VÍDEO

YOUTUBE **RabbitMQ — Mensageria e Arquitetura Baseada em Eventos (EDA) — Full Cycle** youtube.com/watch?v=yxeMFqIA

Owg

Referências bibliográficas

- TANENBAUM, A. S.; VAN STEEN, M. Distributed Systems. 3. ed. 2018. ISBN 978-90-815406-2-9. distributed-systems.net
- RABBITMQ. RabbitMQ Tutorials. rabbitmq.com/tutorials