

Rabbit MQ



Sumário

Rabbit
MQ



- RabbitMQ: Principais conceitos
- Preparação de ambiente
- Exemplo de aplicação simples
- Servidor de três tarefas



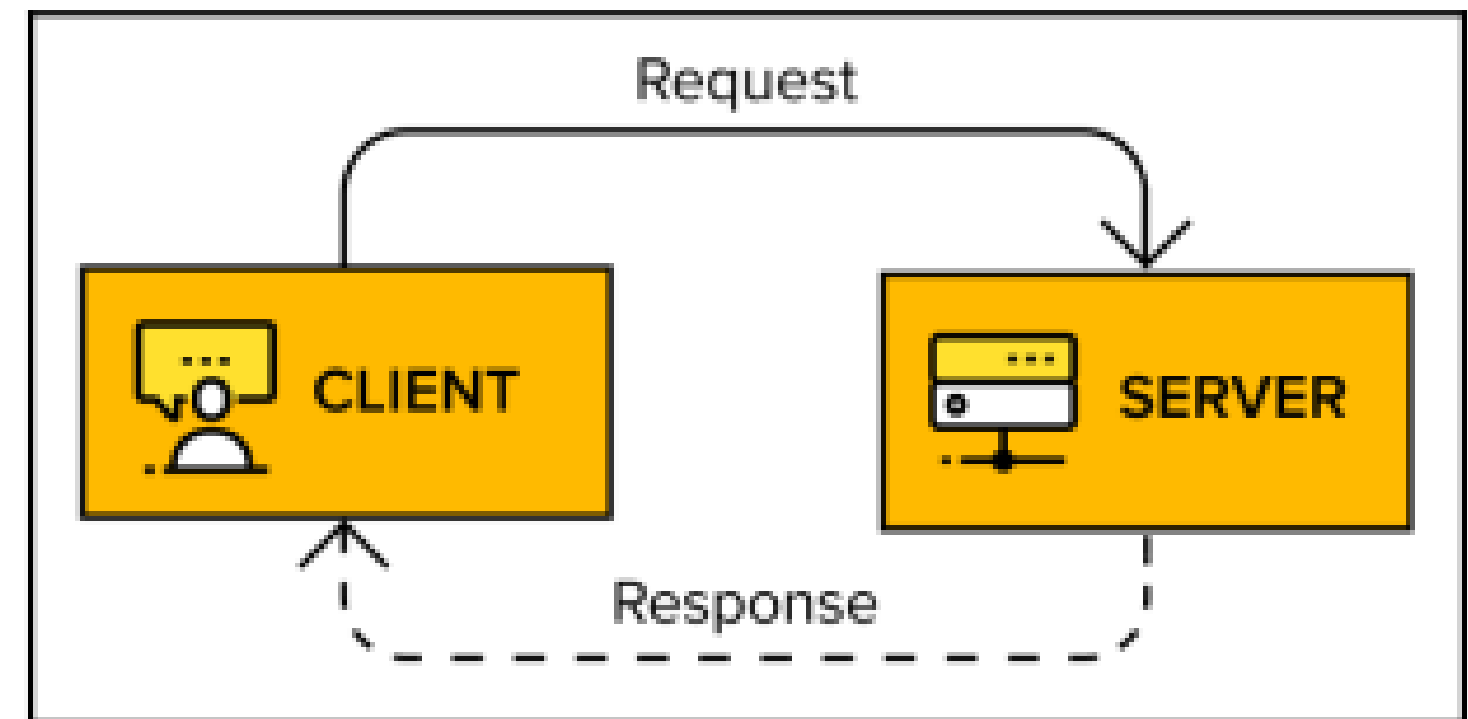
INTRODUÇÃO

- Middleware de mensagens
- Open-source
- Implementação **baseada em Erlang** do **Advanced Message Queuing Protocol (AMQP)** - *fila de mensagens*
- Suporta recursos avançados como clustering e complexo roteamento de mensagens



Request-response

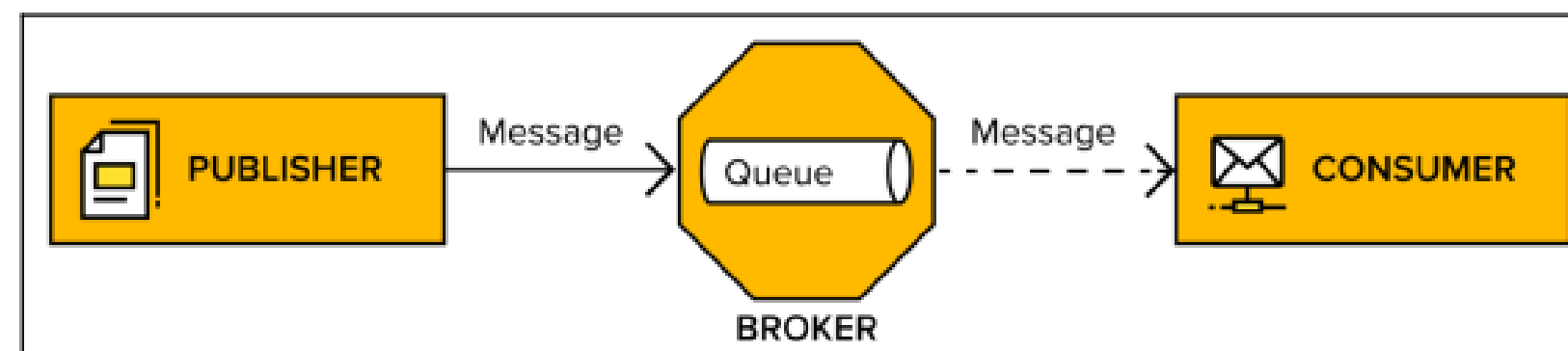
- Um sistema, agindo como um cliente, interage com outro sistema remoto, que está atuando como um servidor
- O cliente envia uma solicitação de dados, e o servidor responde à solicitação
- Seja na forma de RPC, uma invocação de web service, ou consumo de um recurso, o modelo é o mesmo: um sistema envia uma mensagem para outro e espera que a parte remota responda
- A interação entre o cliente e o servidor é **síncrona**





Message Queuing / Publish-subscribe

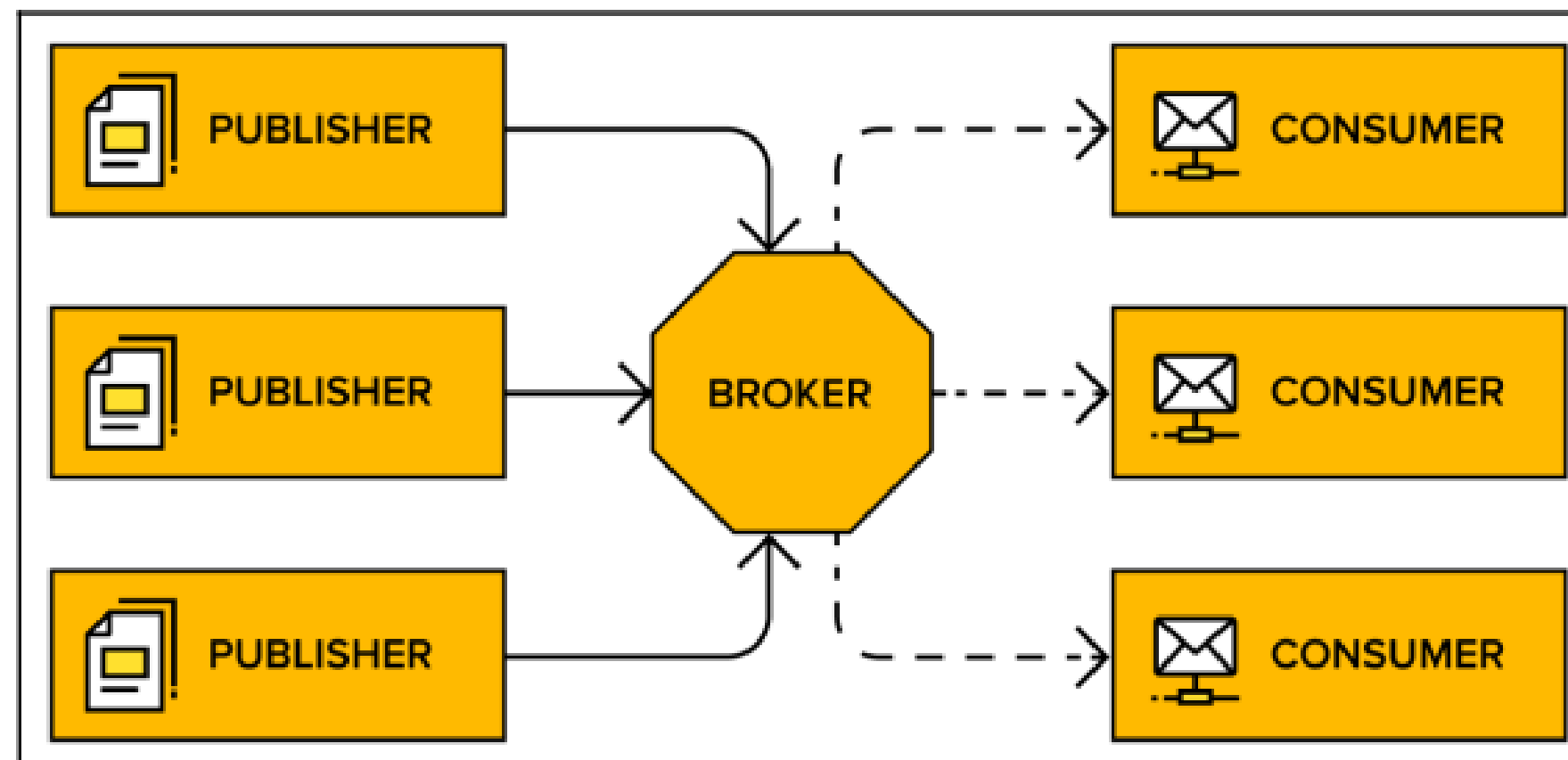
- Estilo de comunicação unidirecional que fornece interação **assíncrona** entre sistemas
- Geralmente através de um **broker**
- Sistemas e aplicativos desempenham tanto o papel de **publishers** quanto de **subscribers**
- Um publisher envia uma mensagem para um broker no qual ele confia para entregar os dados ao subscriber pretendido
- Se uma resposta for necessária, utiliza-se o mesmo mecanismo





Arquitetura de baixo acoplamento

- Sistemas não precisam saber a localização de outros nós na rede
- Os sistemas podem evoluir de uma maneira independente, sem impacto um no outro, pois a confiabilidade da entrega de mensagens é confiada a um **broker**
- Se um sistema estiver inativo, a outra parte do sistema ainda pode operar, e as mensagens que deveriam ser enviadas entre eles aguardam na fila





Arquitetura de baixo acoplamento

- Principais pontos da arquitetura de message queuing:
 - Publisher e subscriber podem ser atualizados um por um, sem que eles impactem uns aos outros
 - O desempenho de cada lado deixa o outro lado inalterado
 - Os publicadores ou consumidores podem falhar sem impactar uns aos outros
 - O número de instâncias de publicadores e consumidores para escalar e acomodar sua carga de trabalho em completa independência
 - Mistura de tecnologia entre consumidor e publicadores



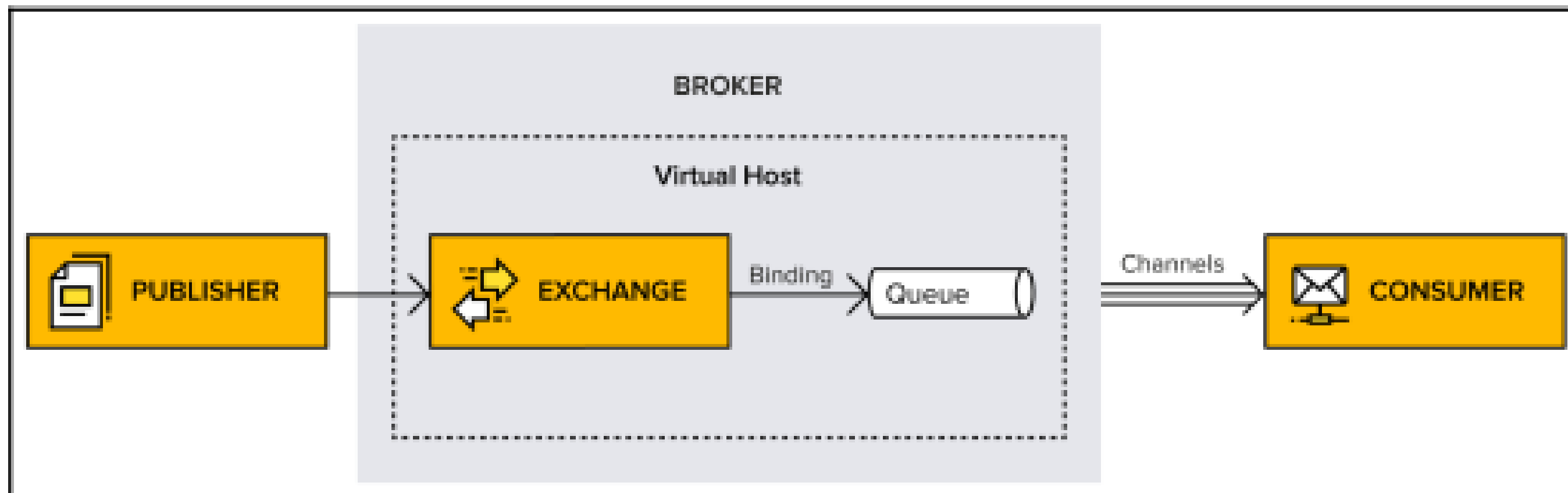
AMQP - Introdução

- Protocolo padrão aberto que define como um sistema pode trocar mensagens
- Define um conjunto de regras a serem seguidas pelos sistemas que vão se comunicar entre si
- Além de definir a interação que acontece entre um publisher/subscriber e um broker, também define a representação das mensagens e comandos que estão sendo trocados



AMQP - Principais conceitos

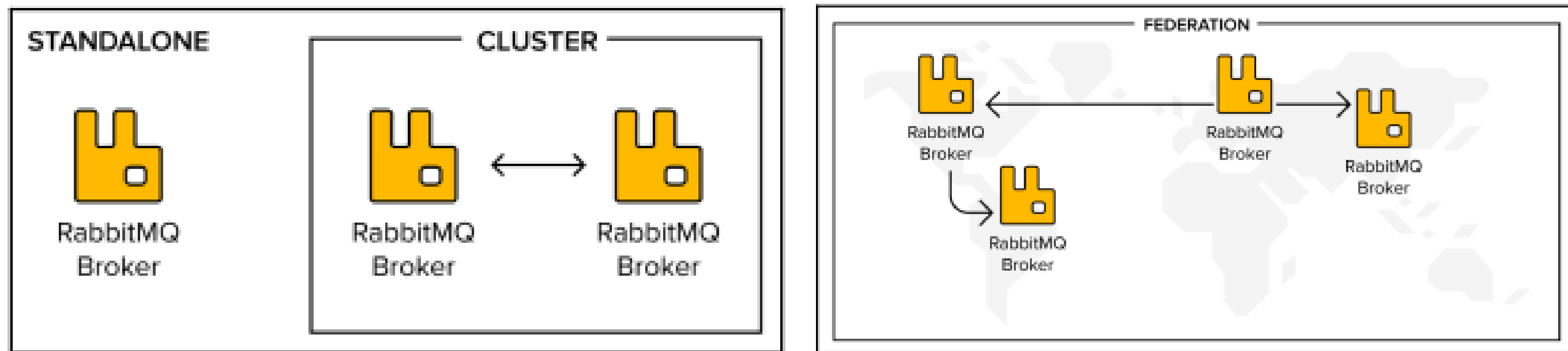
- Message Broker
- Virtual host (vhost)
- Connection (TCP)
- Channel
- Exchange
- Queue
- Binding





RabbitMQ Broker

- RabbitMQ é uma implementação **Erlang** de um **AMQP broker**
- Configurado em instância autônoma ou cluster de servidores
- Brokers podem ser conectados usando diferentes técnicas, como federação e shovels, para formar topologias de mensagens com roteamento inteligente de mensagens entre corretores e capacidade de abranger vários data centers.

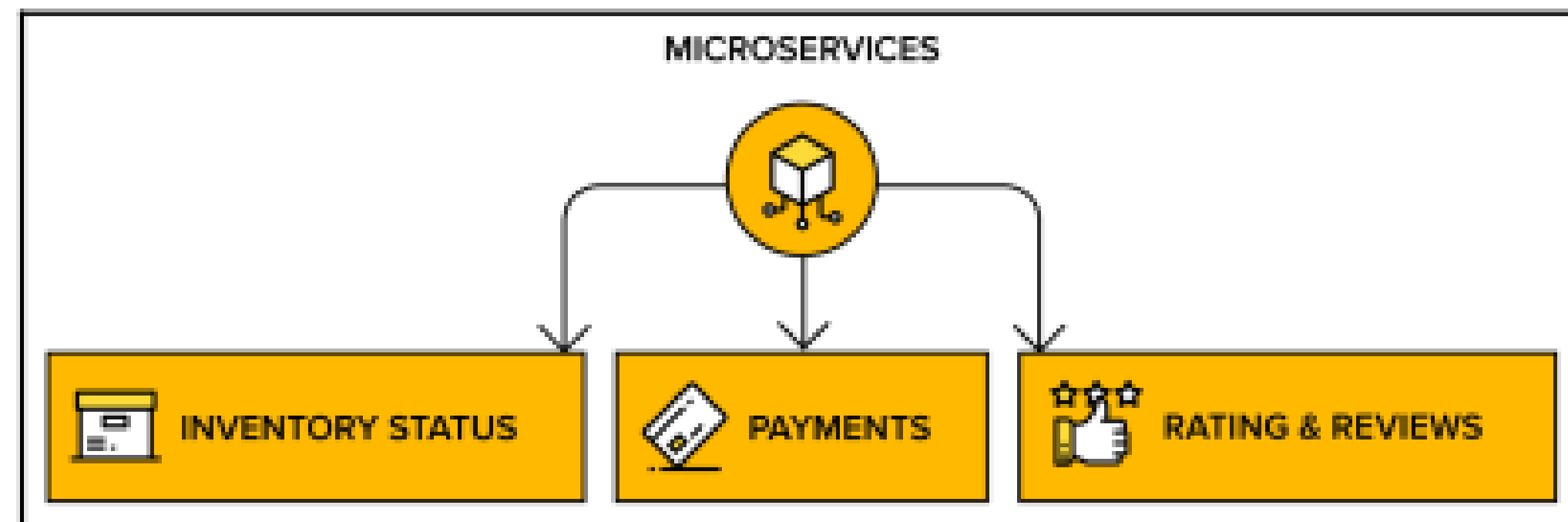


RabbitMQ Broker

Rabbit
MQ



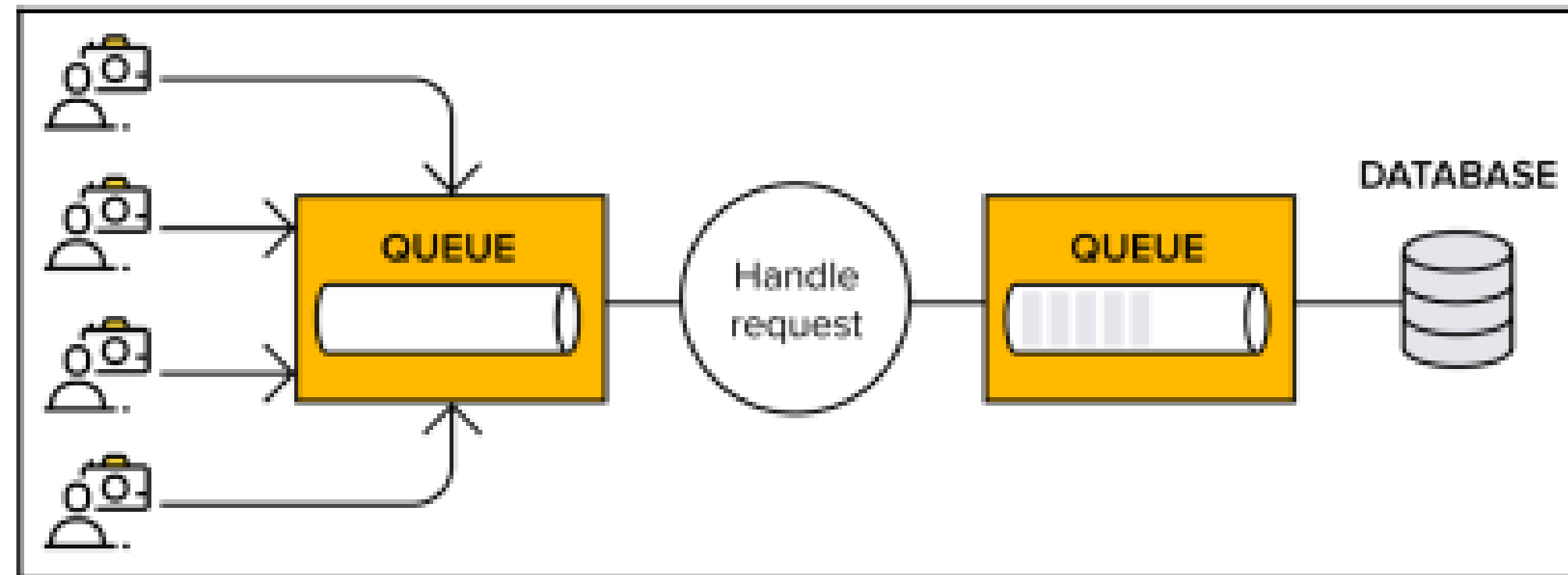
- O caso de uso mais comum é uma fila de um único publisher e subscriber
 - Frequentemente usadas entre **microserviços**
- O estilo arquitetônico de microserviços divide o aplicativo em pequenos serviços, com o aplicativo finalizado sendo a soma de seus microserviços
- Os serviços não são estritamente conectados uns aos outros, são usados message queuings para manter contato





RabbitMQ Broker

- Outro caso de uso típico é com fila de **eventos e tarefas**
- **Eventos** são notificações que informam aos aplicativos quando algo aconteceu
- Um aplicativo pode assinar eventos de outro aplicativo e responder criando e manipulando **tarefas** para si mesmo





Preparação do Ambiente

RabbitMQ pode rodar em **qualquer plataforma** que ofereça uma versão suportada do Erlang, desde nós de múltiplos núcleos e implantações em nuvem até sistemas embarcados

- Linux
- Windows
- Windows Server
- macOS
- Solaris
- FreeBSD

Preparação do Ambiente

Rabbit
MQ



Instalando o Erlang: Devemos saber primeiro qual a versão correta e suportada pelo RabbitMQ.

Sempre ter somente uma versão instalada do Erlang no computador para que funcione corretamente.

RabbitMQ version	Minimum required Erlang/OTP	Maximum supported Erlang/OTP	Notes
<ul style="list-style-type: none">• 4.0.2• 4.0.1	<ul style="list-style-type: none">• 26.2	<ul style="list-style-type: none">• 26.2.x	<ul style="list-style-type: none">• The 4.0 release series is compatible with Erlang 26.2. <div>⚠ WARNING RabbitMQ 4.0 works with Erlang/OTP 27. However, there are known performance regressions, which will be addressed in patch releases. Running RabbitMQ 4.0 on Erlang 27 is not recommended in production but can be used in development environments.</div>
<ul style="list-style-type: none">• 3.13.7• 3.13.6• 3.13.5• 3.13.4• 3.13.3• 3.13.2• 3.13.1• 3.13.0	<ul style="list-style-type: none">• 26.0	<ul style="list-style-type: none">• 26.2.x	<ul style="list-style-type: none">• The 3.13 release series is compatible with Erlang 26.• Starting with Erlang 26, TLS client peer verification is enabled by default by the TLS implementation. If client TLS certificate and key pair is not configured, TLS-enabled Shovels, Federation links and LDAP server connections will fail. If peer verification is not necessary, it can be disabled.• OpenSSL 3 support in Erlang is considered to be mature and ready for production use.• Erlang 26.1 and later versions supports FIPS mode on OpenSSL 3

Instalação do RabbitMQ

Rabbit
MQ



- Usando a imagem do Docker:

```
# latest RabbitMQ 3.13
docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3.13-management
```

- Usando o chocolatey package:

```
choco install rabbitmq
```

- Usando o instalador do Windows:

Description	Download	Signature
Installer for Windows systems (from GitHub)	rabbitmq-server-4.0.2.exe	Signature

Fonte: <https://www.rabbitmq.com/docs/download>

Instalação do RabbitMQ

Rabbit
MQ



- No Ubuntu:

```
## Install Erlang packages
sudo apt-get install -y erlang-base \
    erlang-asn1 erlang-crypto erlang-eldap erlang-ftp erlang-inets \
    erlang-mnesia erlang-os-mon erlang-parsetools erlang-public-key \
    erlang-runtime-tools erlang-snmp erlang-ssl \
    erlang-syntax-tools erlang-tftp erlang-tools erlang-xmerl

## Install rabbitmq-server and its dependencies
sudo apt-get install rabbitmq-server -y --fix-missing
```

Fonte: <https://www.rabbitmq.com/docs/download>



Instalação do RabbitMQ

- **Variáveis de Ambiente**

Foi preciso criar as seguintes variáveis no Windows:

ERL_LIBS	C:\Program Files\Erlang OTP\lib
----------	---------------------------------

ERLANG_HOME	C:\Program Files\Erlang OTP
-------------	-----------------------------

RABBITMQ_HOME	C:\Program Files\rabbitmq_server-3.13.7
---------------	---

E no Path foi necessário acrescentar:

%ERLANG_HOME%\bin

%RABBITMQ_HOME%\sbin



Principais Comandos

```
#Iniciar o RabbitMQ  
rabbitmq-service.bat start
```

```
#Parar o RabbitMQ  
rabbitmq-service.bat stop
```

```
#Ver quais comandos estão disponíveis  
rabbitmqctl help
```

```
#Verificar o Status  
rabbitmq-service.bat status
```

```
#criar um usuario  
rabbitmqctl add_user username password
```

```
#Definir o usuário como administrador  
rabbitmqctl set_user_tags username administrator
```

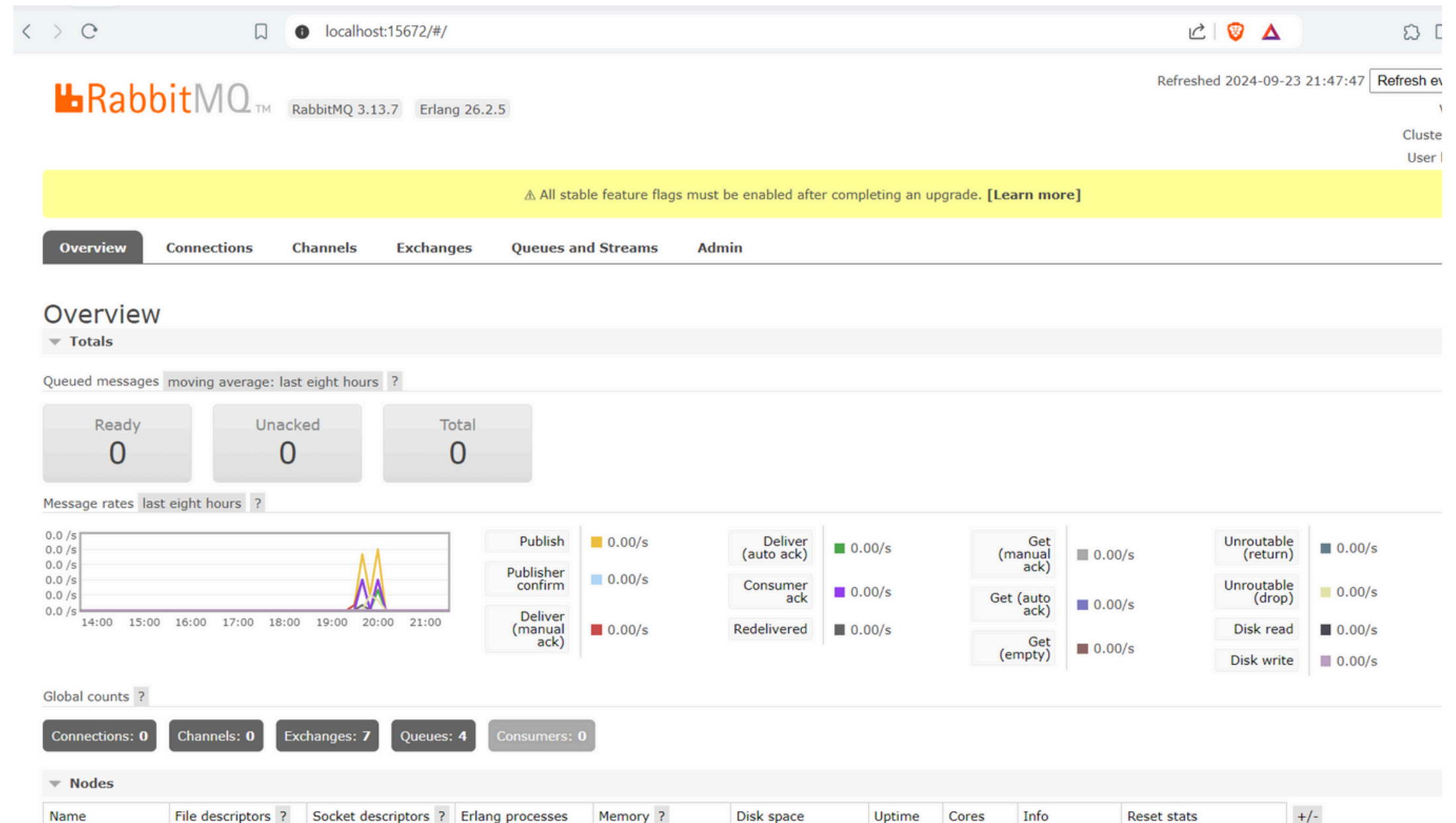
```
#Dar permissões específicas ao usuário  
rabbitmqctl set_permissions -p / username ".*" ".*" ".*"
```

RabbitMQ Management Plugin

Rabbit
MQ



Ele permite visualizar e controlar a fila de mensagens, trocas, conexões e muito mais através de uma interface web. O plugin oferece recursos como estatísticas em tempo real, gerenciamento de usuários e permissões, além de opções para configurar e testar filas.



Programa emissor



```
#!/usr/bin/env python
import pika

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

channel.queue_declare(queue='hello')

channel.basic_publish(exchange='', routing_key='hello', body='Hello World!')
print(" [x] Sent 'Hello World!'")
connection.close()
```

Programa consumidor



```
#!/usr/bin/env 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(f" [x] Received {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()
```

Servidor das três tarefas ...

```
#!/usr/bin/env python
import pika
...
connection = pika.BlockingConnection(
    pika.ConnectionParameters(host="localhost"),
)
channel = connection.channel()

channel.queue_declare(queue="rpc_queue")
channel.queue_declare(queue="text_queue")
channel.queue_declare(queue="salvar_queue")


channel.basic_qos(prefetch_count=1)
channel.basic_consume(queue="rpc_queue", on_message_callback=on_request)
channel.basic_consume(queue="text_queue", on_message_callback=on_request2)
channel.basic_consume(queue="salvar_queue", on_message_callback=on_request3)

channel.start_consuming()
```

Calculando uma função

```
def fib(n):  
    if n == 0:  
        return 0  
    elif n == 1:  
        return 1  
    else:  
        return fib(n - 1) + fib(n - 2)  
  
def on_request(ch, method, props, body):  
    n = int(body)  
    print(f" [.] fib({n})")  
    response = fib(n)  
    ch.basic_publish(  
        exchange="",  
        routing_key=props.reply_to,  
        properties=pika.BasicProperties(correlation_id=props.correlation_id),  
        body=str(response),  
    )  
    ch.basic_ack(delivery_tag=method.delivery_tag)
```

Respondendo texto concatenado



```
def on_request2(ch, method, props, body):
    texto = body.decode()
    concat = "Hello " + texto + "!"

    ch.basic_publish(
        exchange="",
        routing_key=props.reply_to,
        properties=pika.BasicProperties(correlation_id=props.correlation_id),
        body=str(concat),
    )
    ch.basic_ack(delivery_tag=method.delivery_tag)
```


Respondendo texto concatenado



```
def ativar(texto):  
    f = open("placas_ativadas.txt", "a")  
    f.write("{} \n".format(texto))  
    f.close()  
  
def on_request3(ch, method, props, body):  
    print("chegou")  
    texto = body.decode()  
    ativar(texto)  
    print("Salvo no arquivo placas_ativadas.txt")
```

Materiais adicionais

- Tutoriais

- [**https://www.rabbitmq.com/tutorials**](https://www.rabbitmq.com/tutorials)

- Playlist

- [**https://www.youtube.com/playlist?list=PLalrWAGybpB-UHbRDhFsBgXJM1g6T4lvO**](https://www.youtube.com/playlist?list=PLalrWAGybpB-UHbRDhFsBgXJM1g6T4lvO)

- Livro

- **RabbitMQ Essentials (Lovisa Johansson, David Dossot)**