INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

SERVIÇO DE MENSAGERIA

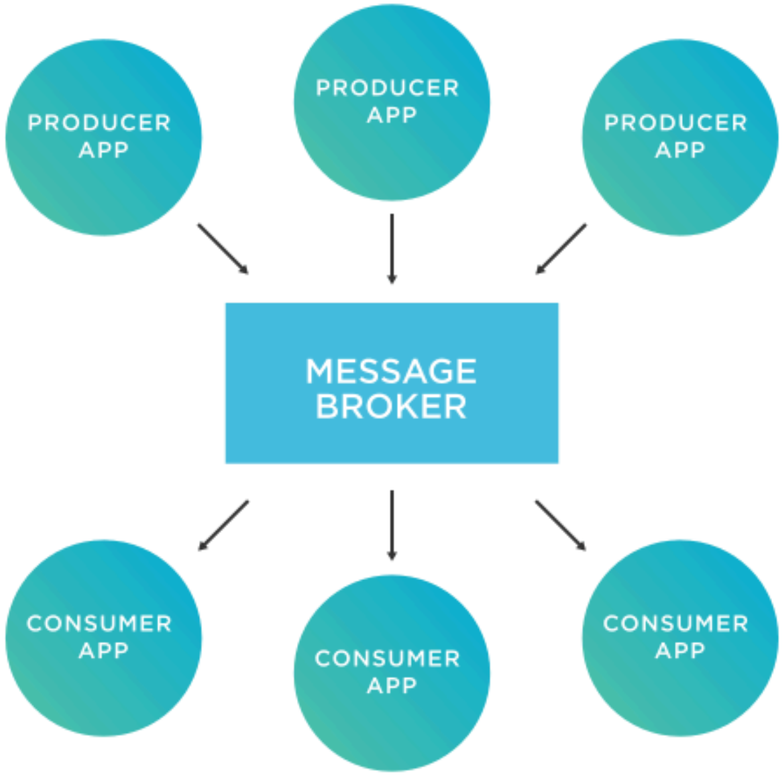
IFNMG CAMPUS JANUÁRIA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO - SISTEMAS DISTRIBUÍDOS
PROF: MSC. ADRIANO ANTUNES PRATES
ALUNA: GABRIELA VITÓRIA AQUINO PEREIRA

MENSAGENS QUEUES DISTRIBUÍDAS

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

As filas de mensagens distribuídas são um padrão de arquitetura que permite comunicação assíncrona e desacoplada entre componentes de um sistema distribuído.

Em vez de se comunicarem diretamente, esses componentes utilizam um intermediário chamado **message broker.**



TECNOLOGIAS POPULARES QUE IMPLEMENTAM ESSE CONCEITO

ACTIVEMQ

kafka

amazon SQS

Azure Service Bus
Managing Connection Lifetime

RabbitMQ

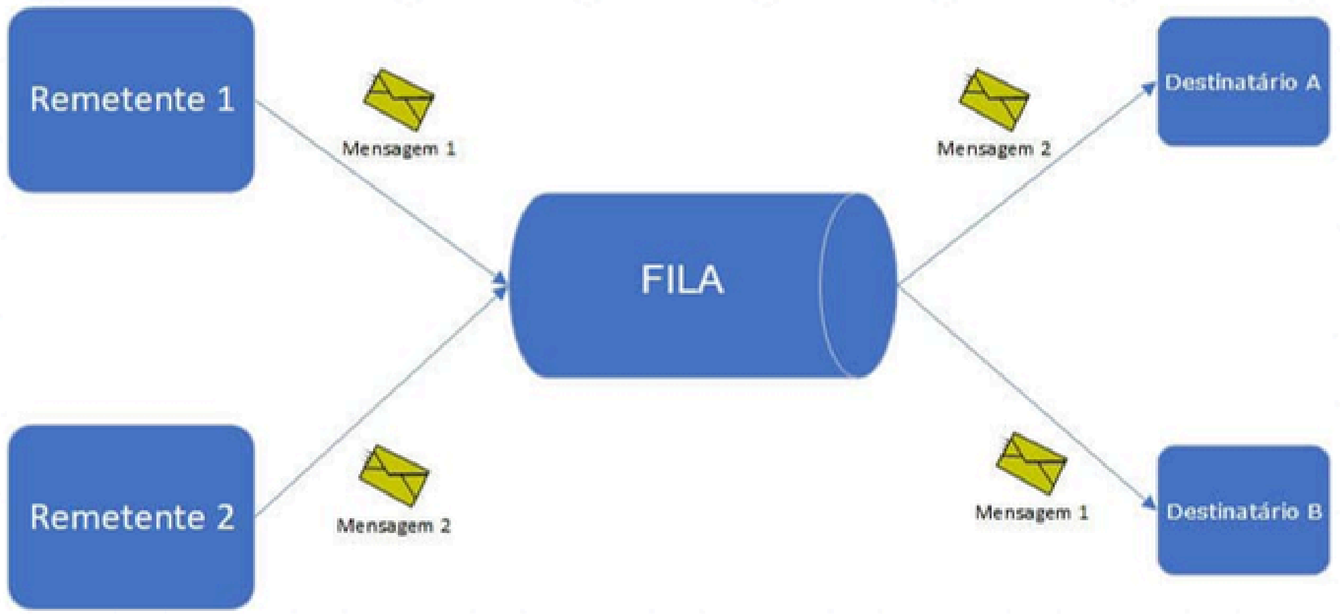
MENSAGENS QUEUES DISTRIBUÍDAS

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Funcionamento

O funcionamento baseia-se em três elementos principais:

1. Producer: Um componente que cria e envia mensagens para a fila.
2. Queue: Uma estrutura temporária onde as mensagens são armazenadas até serem processadas.
3. Consumer: Um componente que recupera e processa as mensagens da fila.



MENSAGENS QUEUES DISTRIBUÍDAS

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Filas de mensagens distribuídas são úteis em uma variedade de cenários de sistemas distribuídos, principalmente onde é necessária comunicação **assíncrona, desacoplamento de serviços, resiliência e escalabilidade.**

Principais cenários e casos de uso incluem:

Processamento Assíncrono

Desacoplamento de Serviços

Balanceamento de Carga

Tarefas em Segundo Plano

Tolerância a Falhas e Resiliência

Auditoria e Log Centralizado

Comunicação entre Sistemas Heterogêneos

MENSAGENS QUEUES DISTRIBUÍDAS

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Vantagens

Desacoplamento e flexibilidade

Os produtores e consumidores de mensagens operam de forma independente, sem precisar estar online simultaneamente. Isso permite que os serviços evoluam e sejam implantados de forma autônoma.

Escalabilidade

O sistema pode lidar com cargas de trabalho flutuantes escalando produtores e consumidores dinamicamente. Uma única máquina pode não ser suficiente para um grande volume de pedidos, e a arquitetura distribuída resolve essa limitação.

Tolerância a Falhas

Se um serviço consumidor falhar, as mensagens permanecem na fila até que o serviço se recupere e possa processá-las. Isso evita falhas em cascata em todo o sistema.

MENSAGENS QUEUES DISTRIBUÍDAS

Vantagens

Processamento assíncrono

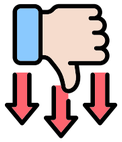
Permite operações não bloqueantes, melhorando o desempenho geral do sistema, pois o produtor não precisa esperar uma resposta imediata do consumidor.

Confiabilidade

Com mecanismos de persistência e confirmação (acknowledgment), as filas de mensagens garantem que as mensagens não sejam perdidas durante falhas do sistema.

MENSAGENS QUEUES DISTRIBUÍDAS

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária



Desvantagens

Complexidade Adicional

Projetar, implementar e manter um sistema de mensagens distribuído aumenta significativamente a complexidade geral do sistema e requer experiência técnica.

Desafios de consistência de dados

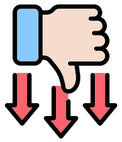
Garantir a integridade e a consistência dos dados em um ambiente distribuído pode ser complexo, especialmente ao coordenar vários serviços envolvidos em uma única operação.

Sobrecarga de Rede

A troca constante de mensagens entre os componentes pode aumentar o tráfego de rede e, em casos de saturação, levar a atrasos na transmissão.

MENSAGENS QUEUES DISTRIBUÍDAS

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária



Desvantagens

Latência Potencial

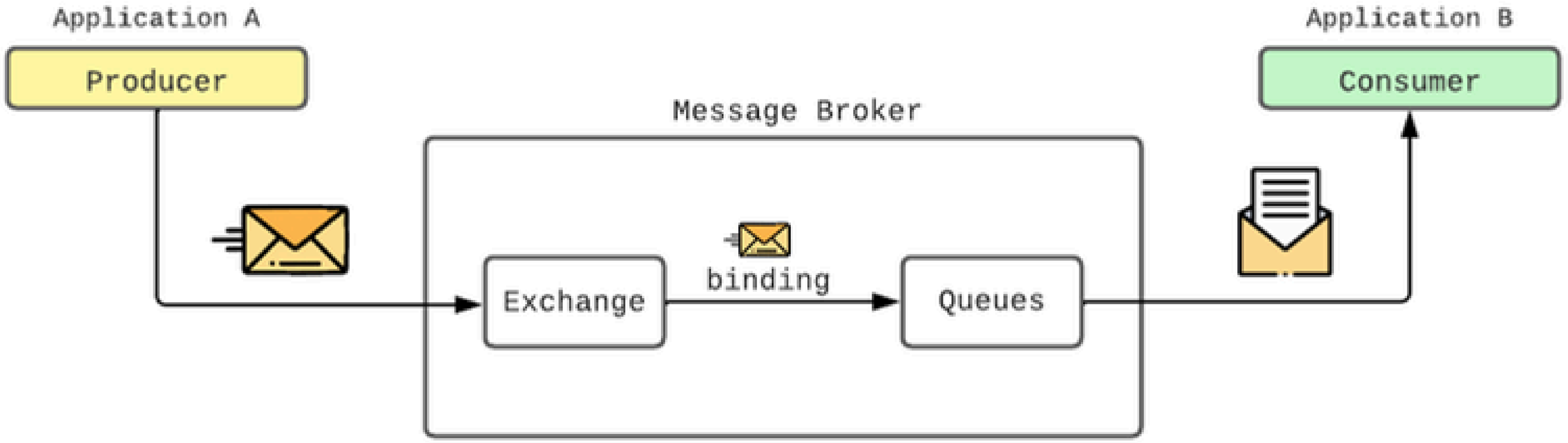
Embora o processamento assíncrono melhore o desempenho geral, pode haver um atraso inerente na entrega e processamento de mensagens, dependendo da carga e da configuração do sistema.

Dificuldades de Monitoramento e Rastreamento

Rastrear o fluxo de uma mensagem através de múltiplos serviços pode ser difícil e requer ferramentas de monitoramento especializadas.

RabbitMQ

O RabbitMQ é um corretor de mensagens que atua como intermediário entre aplicações, permitindo comunicação assíncrona e desacoplada.



Componentes principais

Produtor

A aplicação que envia uma mensagem.

Mensagem

O dado a ser transmitido.

Exchange

Recebe a mensagem do produtor e a roteia para a fila apropriada com base em um tipo e uma chave de roteamento (routing key).

Fila

Um buffer que armazena as mensagens até que sejam processadas por um consumidor.

Binding

A regra que liga um exchange a uma fila.

Binding Key

Uma chave usada no binding para determinar como as mensagens são roteadas de um exchange para uma fila.

Consumidor

A aplicação que recebe e processa as mensagens de uma fila.

Funcionamento

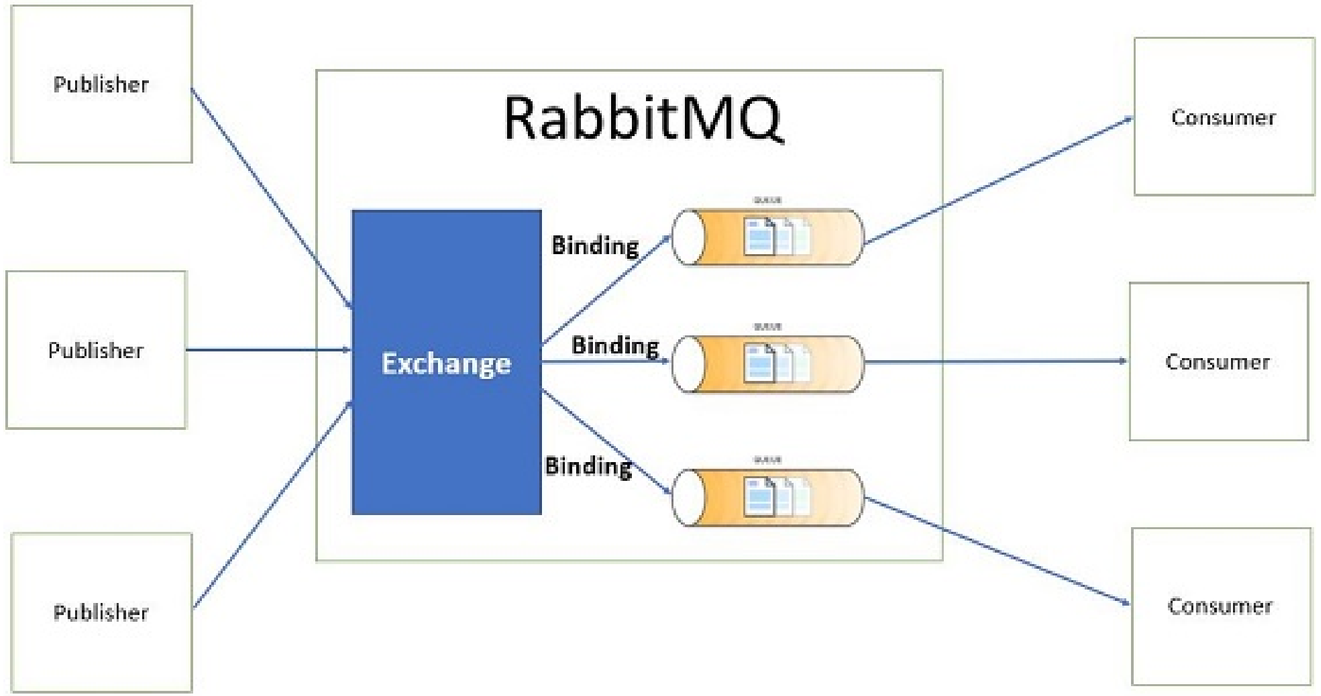
1. O produtor envia uma mensagem para um exchange.

2. O exchange usa um conjunto de regras (bindings) e as chaves de roteamento (routing keys) da mensagem para determinar para quais filas a mensagem deve ser enviada.

3. As mensagens são armazenadas nas filas designadas.

4. Os consumidores se conectam a essas filas e processam as mensagens.

5. Para garantir a entrega, é recomendável usar o acknowledgement (ACK), que permite que o consumidor confirme o processamento. Assim, se um consumidor falhar, a mensagem pode ser reprocessada por outro.



Exchanges

Tipo	Como funciona	Quando usar
Direct exchange	Envia mensagen para uma fila específica baseada na routing key.	Quando se quer enviar para apenas uma fila.
Fanout exchange	Envia a mesma mensagem para todas as filas conectadas.	Quando precisa notificar vários sistemas.
Topic exchange	Encaminha conforme um padrão na routing key (usa * e #).	Quando há diferentes categorias.
Headers exchange	Encaminha a partir de regras nos cabeçalhos.	Quando precisa de mais controle.

Benefícios

Desacoplamento	Produtores e consumidores não precisam conhecer um ao outro, apenas o broker.
Confiabilidade	As mensagens são armazenadas no broker até que sejam processadas, evitando perdas em caso de falha temporária de um sistema.
Escalabilidade	É possível adicionar mais consumidores para processar mensagens em paralelo e escalar o sistema.
Assincronismo	O produtor pode enviar uma mensagem e continuar seu trabalho sem esperar que o consumidor a processe imediatamente.

Instalação via docker

Para utilizar o RabbitMQ, é necessário que ele esteja em execução na porta padrão (5672), pois ele é o servidor que gerencia as filas de mensagens. Para estabelecer essa conexão usaremos um arquivo docker-compose.yml com a seguinte estrutura:

```yaml
services:
  rabbitmq:
    image: rabbitmq:3-management
    container_name: rabbitmq
    restart: always
    ports:
      - "5672:5672"
      - "15672:15672"
    environment:
      RABBITMQ_DEFAULT_USER: admin
      RABBITMQ_DEFAULT_PASS: admin
      RABBITMQ_DEFAULT_VHOST: /
```

Obs: Sem o RabbitMQ rodando, não há comunicação possível.

Integração com Python

A integração é feita através da biblioteca **pika** para conectar, criar filas, enviar e consumir mensagens.

Comando para instalação da biblioteca pika.

```
pip install pika
```

Com o Pika instalado, podemos escrever alguns códigos.

Primeiro precisamos nos conectar ao servidor RabbitMQ.

```
import pika
connection= pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel= connection.channel()
```

Em seguida criar uma fila para a qual a mensagem vai ser entregue.

```
channel.queue_declare(queue='nome_queue' )
```

Para nos conectar a um broker em uma máquina diferente, basta especificar seu nome ou endereço IP aqui.

Integração - Exemplo simples

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

```python
import pika

credentials = pika.PlainCredentials('admin', 'admin')

connection = pika.BlockingConnection(pika.ConnectionParameters(
    host='gabrielavitoria.ddns.net',
    credentials=credentials
    )
)
channel = connection.channel()
channel.queue_declare(queue='fila_notificacoes')

channel.basic_publish(
    exchange='',
    routing_key='fila_notificacoes',
    body='Hello World!'
)

print(" [x] Mensagem enviada")
connection.close()
```

O primeiro programa (envia.py) enviará uma única mensagem para a fila.

Integração - Exemplo simples


```python
import pika, sys, os

def main():
    credentials = pika.PlainCredentials('admin', 'admin')

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

    channel.queue_declare(queue='fila_notificacoes')

    def callback(ch, method, properties, body):
        print(" [x] Recebido:", body.decode())

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

    print(" [*] Consumidor ativo, aguardando mensagens... Para sair, pressione CTRL+C")
    channel.start_consuming()

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print(' Consumidor Interrompido')
        try:
            sys.exit(0)
        except SystemExit:
            os._exit(0)
```

O segundo programa (receive.py) receberá mensagens da fila e imprimirá na tela.

Sempre que recebemos uma mensagem, a função callback é chamada pela biblioteca Pika, enquanto o consumidor permanece em execução aguardando novas mensagens.

RabbitMQ

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

```
http://IP_INSTANCIA:15672/
```

É a URL usada para acessar o dashboard do RabbitMQ.

15672 é a porta padrão do painel de administração do RabbitMQ.

Obs:

Neste projeto, o RabbitMQ foi configurado para permitir conexões externas a partir de qualquer local, facilitando os testes.

Em um ambiente real de produção, medidas de segurança adicionais seriam necessárias, como autenticação forte, uso de SSL/TLS, restrição de IPs e redes privadas.

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Referências

https://www.rabbitmq.com/tutorials/tutorial-one-python

OBRIGADA!

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária