Caio Silas de Araujo Amaro
21.1.4111

Sistemas Distribuídos / Segurança
2023-1 TP3

cfmcc@ufop.edu.br
Prof. Dr. Carlos Frederico MC Cavalcanti

Introdução:[1]

Neste trabalho foi proposto que dois alunos de SD trabalhem juntos para implementar um sistema básico de troca de mensagens utilizando o RabbitMQ e o Docker, a fim de compreender os princípios fundamentais dessas tecnologias.

O'Que é o RabbitMQ:

O RabbitMQ é um sistema de mensageria de código aberto (open-source) amplamente utilizado para a troca de mensagens entre aplicativos e sistemas distribuídos. Ele implementa o padrão de mensageria AMQP (Advanced Message Queuing Protocol) e fornece uma plataforma robusta e escalável para o envio e recebimento de mensagens assíncronas.

O RabbitMQ atua como um intermediário entre os produtores de mensagens (aplicativos que enviam mensagens) e os consumidores de mensagens (aplicativos que recebem e processam as mensagens). Ele oferece recursos avançados de enfileiramento, roteamento e gerenciamento de mensagens, permitindo que os aplicativos se comuniquem de forma assíncrona e confiável.

1. Produtores: São os aplicativos que enviam as mensagens para o RabbitMQ. Eles publicam as mensagens em filas ou trocas (exchanges) para que possam ser entregues aos consumidores.
2. Consumidores: São os aplicativos que recebem e processam as mensagens enviadas pelo RabbitMQ. Eles se conectam às filas ou trocas e consomem as mensagens disponíveis para processamento.
3. Filas: São as estruturas de armazenamento no RabbitMQ, onde as mensagens são temporariamente armazenadas até que sejam consumidas por um consumidor. As filas garantem a entrega confiável das mensagens e permitem que os consumidores processem as mensagens de forma assíncrona.
4. Trocas (Exchanges): São os pontos de entrada para as mensagens no RabbitMQ. As trocas recebem as mensagens dos produtores e as encaminham para as filas com base em regras de roteamento específicas. Existem diferentes tipos de trocas, como direta,

[1]

tópico, cabeçalho e fanout, que determinam como as mensagens são roteadas para as filas.
5. Mensagens: São os pacotes de dados que são trocados entre os produtores e consumidores. As mensagens podem conter qualquer tipo de informação que seja relevante para a aplicação, como eventos, comandos, dados estruturados, etc.

O RabbitMQ é amplamente utilizado em arquiteturas de microsserviços, sistemas distribuídos e integração de aplicativos, onde a comunicação assíncrona e confiável é necessária. Ele fornece recursos avançados de fila, roteamento e garantias de entrega, tornando-se uma escolha popular para implementar padrões de mensageria e fluxos de trabalho em sistemas complexos.

Dockers utilizados:

```
$ docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672
rabbitmq:3.12-management
$ docker run -it --network="host" caiosilas/produtor:bcc362
$ docker run -it --network="host" caiosilas/consumidor:bcc362
```

Como executar:

abra 3 terminais e digite as seguintes linhas:

1. $ docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3.12-management
    a. Esse será o terminal responsável pelo RabbitMQ.
2. $ docker run -it --network="host" caiosilas/produtor:bcc362
    a. Esse será o terminal que estará rodando o contêiner do produtor,para executar o código siga a seguinte instrução:
    b. $ cd caiosilas
    c. $ python3 produtor.py

3. $ docker run -it --network="host" caiosilas/consumidor:bcc362
    a. Esse será o terminal que estará rodando o contêiner do consumidor, para executar o código siga a seguinte instrução:
    b. $ python3 consumidor.py

Link Para o PPT:
https://docs.google.com/presentation/d/1KDi8j0hHikfhEuKNz9tSpF5A1RdAt9-Cjh9YSG3Joig/edit?usp=sharing

Códigos utilizados :

Os códigos utilizados para o trabalho foram feitos na linguagem python

1. Produtor

```python
import pika
import time

# Função para enviar mensagens para a fila
def enviar_mensagens(qtd_mensagens, tamanho_mensagem, intervalo_segundos):
    connection = pika.BlockingConnection(pika.ConnectionParameters('172.0.3'))
    channel = connection.channel()

    # Declara a fila no servidor RabbitMQ
    channel.queue_declare(queue='minha_fila')

    for i in range(qtd_mensagens):
        mensagem = 'Mensagem ' + str(i + 1) + ' ' + 'X' * tamanho_mensagem
        channel.basic_publish(exchange='', routing_key='minha_fila', body=mensagem)
        print("Mensagem enviada:", mensagem)
        time.sleep(intervalo_segundos)

    connection.close()

# Variáveis para definir as configurações do produtor
qtd_mensagens = 10
tamanho_mensagem = 100
intervalo_segundos = 2

# Envia as mensagens para a fila
enviar_mensagens(qtd_mensagens, tamanho_mensagem, intervalo_segundos)
```

2. Consumidor

```python
import pika
import time

# Variáveis para estatísticas do consumidor
mensagens_recebidas = 0
mensagem_esperadas = 10
inicio_segundos = time.time()

# Função para processar as mensagens recebidas
def processar_mensagem(ch, method, properties, body):
    global mensagens_recebidas
    mensagens_recebidas += 1
    print("Mensagem recebida:", body)
    if mensagens_recebidas >= mensagem_esperadas:
            ch.stop_consuming()

# Estabelece a conexão com o servidor RabbitMQ
connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()

# Declara a fila no servidor RabbitMQ
channel.queue_declare(queue='minha_fila')

# Registra a função de callback para receber mensagens
channel.basic_consume(queue='minha_fila', on_message_callback=processar_mensagem, auto_ack=True)

print('Aguardando mensagens. Pressione CTRL+C para sair.')

# Inicia o consumo de mensagens
channel.start_consuming()

# Calcula o tempo decorrido e exibe o número de mensagens por segundo
tempo_decorrido = time.time() - inicio_segundos
mensagens_por_segundo = mensagens_recebidas / tempo_decorrido
print("Mensagens por segundo:", mensagens_por_segundo)
```