PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS

218063 - FUNDAMENTOS DE REDES DE COMPUTADORES (NOT)

VICTOR KYOTO KURATA - 22011995
GUSTAVO VIEIRA BIANCHI - 22020827
BRUNO DE SOUZA SEYFART - 23029740
PATRICK PIMENTEL CORRÊA LEITE - 21007850

RELATÓRIO DE PROJETO:
Desenvolvimento de Aplicação Utilizando Protocolos de Mensageria RabbitMQ (AMQP) ou Mosquitto (MQTT)

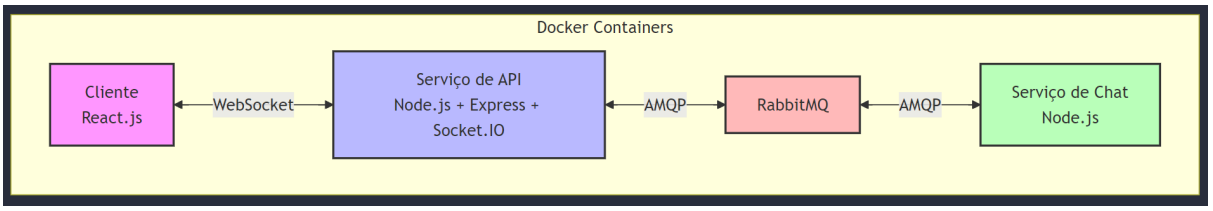
CAMPINAS
2024

Arquitetura do Chat App

Visão Geral

Nossa aplicação de chat é uma aplicação web em tempo real, construída usando uma arquitetura de microsserviços. Ela consiste em quatro componentes principais, cada um rodando em seu próprio container Docker:

1. **Cliente (Frontend)**
2. **Serviço de API**
3. **Serviço de Chat**
4. **RabbitMQ (Message Broker)**



Link do diagrama:
https://mermaid.live/edit#pako:eNqNUktuwjAQvYpllg0oED7BQkiQdMGCfqBSpZIubDIhLsGOHKeFAufpQXqxGmKqdlN1Fp4Z-X08tvd4KWPABK8UzVP0MI4EMjFaBBkHoWHA1HAGdKkbL8UzGtTrQ3R4BDaXyzXoAxov5qBe-eeHRDGg0d3khL8xigaOrtD1NldQnKqK0JjcPlcGY6s1mt7fHVC4mFHGuJ7e2-3w13bwyyVIqf5hYxnVWpSsGiQ8-SkUSKEpF6AKO5d1r1JgzaoEIo6EldG7zIyDEp5lpJb0E6fQSq6B1DzPs3X9jcc6Ja18-5MzthzG_s8JLpyE_ZsTXs7G_uZgB29AbSiPzRvvTwoR1ilsIMLElG8p16aMxNEAaanlfCeWmGhVgoOVLFcpJgnNCtOVeUw1hJyaC95cIDkVT1J-txBzLdW0-lHnj3WGYLLHW0xazYbf9dtuz-12mr2u6zl4h0mz1Wn0Tfhu2_fart9rHR38fhZtHr8A9MzUzA

Componentes

1. Cliente (Frontend)

1. Tecnologia: React.js

2. Responsabilidades:
3. Fornece a interface do usuário para o chat
4. Conecta-se ao Serviço de API via WebSocket (Socket.IO)
5. Envia e recebe mensagens em tempo real

2. Serviço de API

1. Tecnologia: Node.js com Express e Socket.IO
2. Responsabilidades:
3. Gerencia conexões WebSocket com os clientes
4. Encaminha mensagens dos clientes para o RabbitMQ
5. Consome mensagens do RabbitMQ e as envia para os clientes conectados

3. Serviço de Chat

1. Tecnologia: Node.js
2. Responsabilidades:
3. Processa lógica de negócios relacionada ao chat
4. Consome mensagens do RabbitMQ
5. Gera respostas e as envia de volta para o RabbitMQ

4. RabbitMQ (Message Broker)

RabbitMQ é um software de message broker (intermediário de mensagens). Ele atua como um intermediário para mensagens, aceitando e encaminhando mensagens entre diferentes partes de um sistema.

1. Tecnologia: RabbitMQ
2. Responsabilidades:
    a. Atua como um intermediário de mensagens entre os serviços
    b. Gerencia filas de mensagens para garantir a entrega confiável

Como o RabbitMQ está sendo usado em nosso projeto?

No nosso projeto de chat, o RabbitMQ está sendo usado para facilitar a comunicação entre o Serviço de API e o Serviço de Chat. Ele está gerenciando duas filas principais:

1. `chat_messages`: Para mensagens enviadas pelos clientes
2. `chat_responses`: Para respostas processadas pelo Serviço de Chat

Fluxo de Mensagens

1. **Envio de Mensagens**:
    1.1. Quando um cliente envia uma mensagem, o Serviço de API a recebe.
    1.2. O Serviço de API então publica esta mensagem na fila `chat_messages` do RabbitMQ.
2. **Processamento de Mensagens**:
    2.1. O Serviço de Chat está constantemente "escutando" a fila `chat_messages`.
    2.2. Quando uma nova mensagem chega, o Serviço de Chat a consome (retira da fila).
    2.3. O Serviço de Chat processa a mensagem (por exemplo, formatando, salvando em um banco de dados em um possível cenário, etc.).
3. **Envio de Respostas**:
    3.1. Após processar a mensagem, o Serviço de Chat publica uma resposta na fila `chat_responses`.
4. **Distribuição de Respostas**:
    4.1. O Serviço de API está constantemente "escutando" a fila `chat_responses`.
    4.2. Quando uma nova resposta chega, o Serviço de API a consome.
    4.3. O Serviço de API então envia esta resposta para todos os clientes conectados via WebSocket.

Vantagens do uso do RabbitMQ

1. **Desacoplamento**: Os serviços não precisam conhecer um ao outro diretamente.
2. **Balanceamento de Carga**: Múltiplas instâncias do Serviço de Chat podem consumir mensagens da mesma fila.
3. **Persistência**:

4. **Assincronicidade**: O Serviço de API não precisa esperar o processamento da mensagem para continuar seu trabalho.

Configuração no Projeto

No nosso projeto, estamos usando as seguintes configurações:

1. Duas filas: `chat_messages` e `chat_responses`
2. Conexão AMQP padrão: `amqp://rabbitmq`
3. As filas são declaradas tanto no Serviço de API quanto no Serviço de Chat usando `channel.assertQueue()`

Usamos `channel.assertQueue()` para garantir que as filas existam antes de usá-las.

Mensagens são publicadas usando `channel.sendToQueue()`.

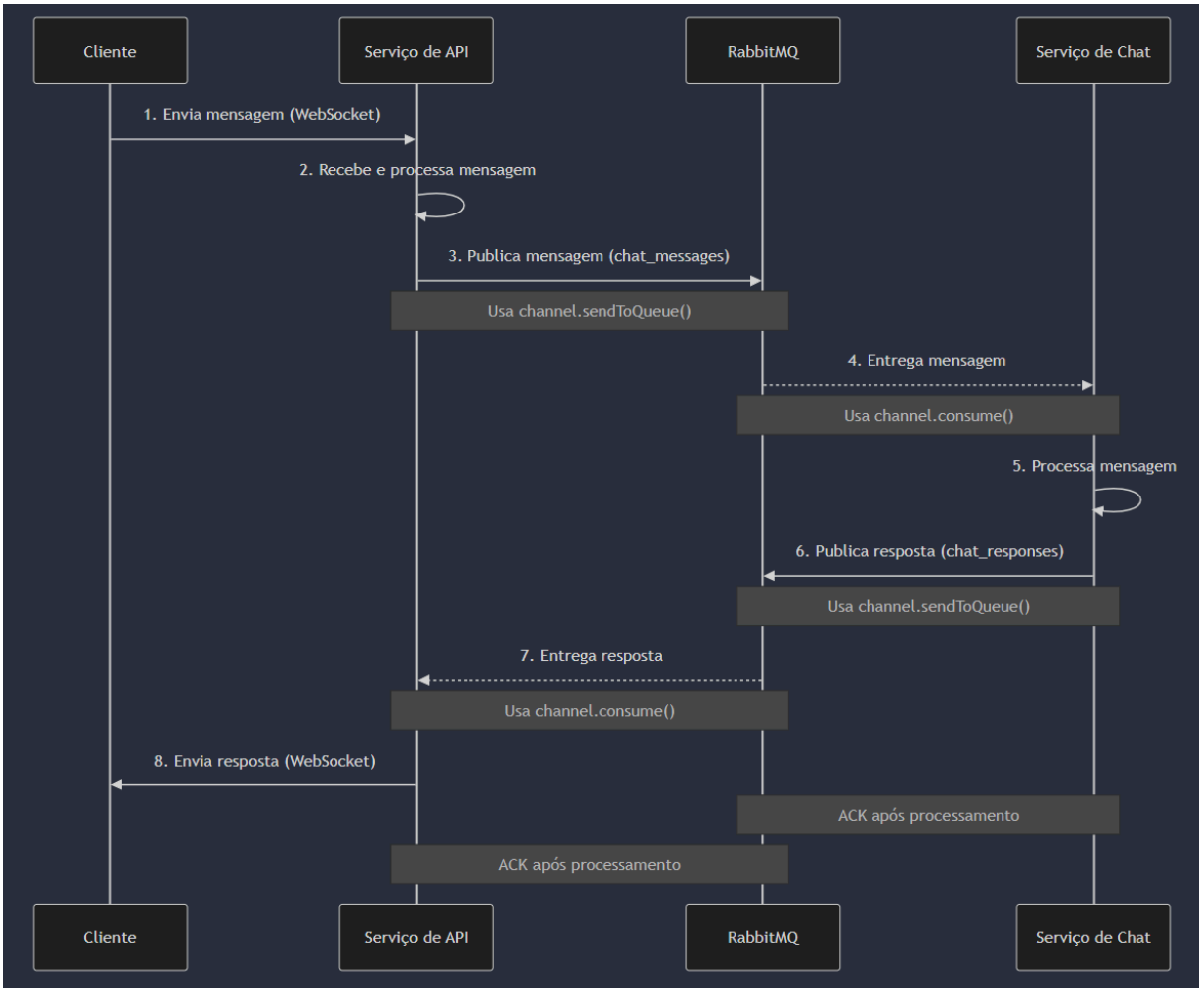
Consumimos mensagens usando `channel.consume()`.

Utilizamos `channel.ack()` para confirmar que uma mensagem foi processada com sucesso.

Esta configuração permite uma comunicação eficiente e confiável entre os diferentes componentes do nosso sistema de chat.

Fluxo de Comunicação

1. O cliente envia uma mensagem através da interface web
2. O Serviço de API recebe a mensagem via WebSocket
3. O Serviço de API publica a mensagem em uma fila do RabbitMQ
4. O Serviço de Chat consome a mensagem da fila do RabbitMQ
5. O Serviço de Chat processa a mensagem e publica uma resposta em outra fila do RabbitMQ
6. O Serviço de API consome a resposta da fila do RabbitMQ
7. O Serviço de API envia a resposta de volta para os clientes conectados via WebSocket



Link mermaid:
https://mermaid.live/edit#pako:eNqVk0tqwzAQhq8yaJWCE-i7eBEIbhelpOTR0o2hyPLUEbEIV49ACTlPD9Aj5GKVbCcOeRTqlSV988__a-wlYTJFEhKNnxYFw3tOM0WLWlB7SqoMZ7ykwkAEVEOUcxQGD08Ho0d_PkW14OtvCSn6rUNuMhx7bkKThJvh-EibGTX7Sn4vFjUbdft9pxzCeQ8exIJTKFBommEBnTdMppLN0ZzVrOM29EUPJsgwQUAolWSodVu5SzuDIVz2YGSTnLNddeZcvBe-MEPddHiWBkEuUPnqoKp9dcIOFQLznkaRvsixRYudpsIxXdfGRwrhymcwCrN9L62u44Ma3hVmUmhbbEU9sBW9duaPJ2ywyuZNG1GhLqU2tlIYLYU-ktHX_ydkdfO3bcZNo2MZK_aPiPV0ohDuNnNvbe_O_bTnQfQEtFz_6O0H4G7HyJODPMGTgBSoCspT99csfXVMzAwLjEnoXlOq5jGJxcpx1Bo5_RKMhEZZDIiSNpuR8IPm2q1smVKz-d9qZPULPV8rSw

Vantagens desta Arquitetura por inteiro:

1. **Escalabilidade**: Cada serviço pode ser escalado independentemente
2. **Resiliência**: Falhas em um serviço não afetam diretamente os outros

3. **Flexibilidade**: Permite fácil adição ou modificação de funcionalidades
4. **Manutenibilidade**: Cada serviço pode ser desenvolvido, testado e implantado separadamente