deti universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

CD 2022/2023
LEI, 10/06/2023 - P2

PROJETO FINAL CD

Diana Miranda (107457)
Bárbara Galiza (105937)

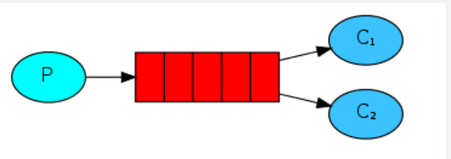
Arquitetura Cliente-Servidor

Decidimos utilizar esta arquitetura porque os clientes fazem pedidos ao servidor, que, por sua vez, processa esses pedidos e retorna os resultados aos clientes.

Esta arquitetura permite uma separação clara de responsabilidades, onde o cliente inicia o processo e o servidor coordena todas as etapas necessárias para fornecer o resultado desejado.

A API foi feita com a framework FastAPI, por já termos trabalhado com ela em outros projetos.

Arquitetura Broker



Optamos por usar o RabbitMQ, um message broker, para criar um sistema de message queueing entre o servidor e os workers. Essa arquitetura permite-nos distribuir tarefas entre vários workers que trabalham em paralelo, acelerando o processamento das músicas.

Dividimos as músicas em partes de 10 segundos e as enviamos para a fila "music_parts". Cada worker ativo consome uma parte de cada vez. Após o processamento, cada worker envia mensagens para a fila "processed_parts", contendo as partes que correspondem ao instrumentos selecionados pelo utilizador. O servidor consome as mensagens da fila "processed_parts", juntando as partes de cada instrumento e combinando os instrumentos.

Escolhemos o RabbitMQ porque ele permite-nos evitar a criação de um message broker e de um protocolo do zero, além de fornecer funcionalidades úteis para lidar com falhas. Para tratar das falhas, utilizamos o método "basic_nack" com a tag "requeue=True". Desta forma, caso ocorra uma falha, a tarefa é colocada de volta na fila para uma nova tentativa de processamento.

Protocolo

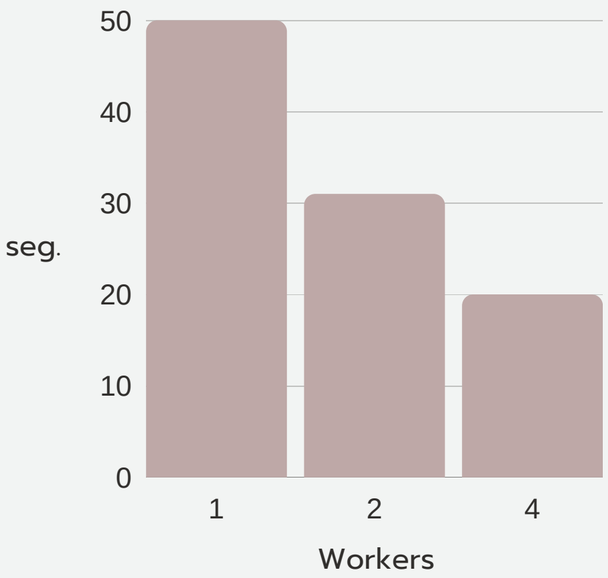
No nosso projeto, utilizamos o RabbitMQ como sistema de mensagens assíncrono, pelo que não foi preciso defenir um protocolo extenso, como fizemos em projetos anteriores. Aproveitámos as funcionalidades fornecidas pelo RabbitMQ, como as funções basic_publish e basic_consume.

A função basic_publish é responsável por enviar mensagens para a fila "music_parts" do RabbitMQ. Essas mensagens contêm as informações relevantes e são serializadas em formato JSON e incluídas no "body".

Já a função basic_consume é usada para chamar uma função de retorno (callback) sempre que uma nova mensagem é encontrada na fila.

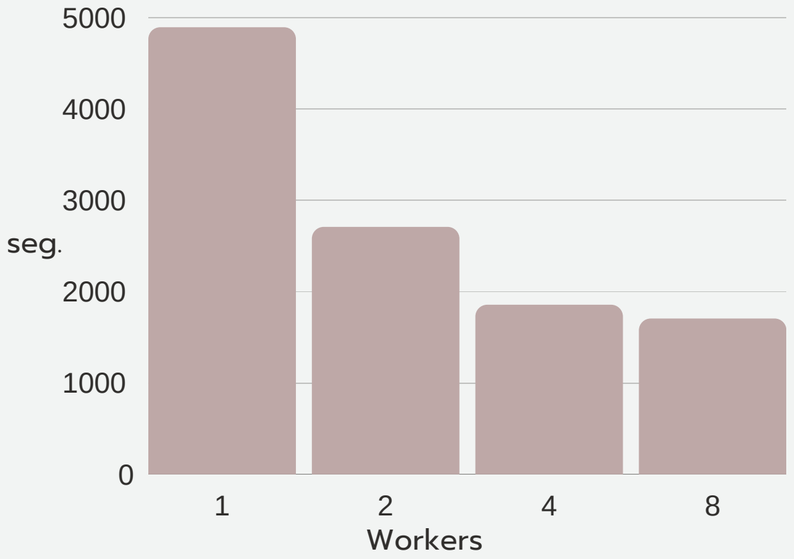
Dentro da função receive_music_parts, ocorre a decodificação da mensagem recebida. As informações relevantes, como o ID da música, o índice da parte, as faixas escolhidas e o áudio da parte da música, são extraídas. No campo part_audio, os bytes do arquivo de áudio são enviados em formato de string, e a conversão é realizada usando a função decode('latin1').

Benchmarks



Música de 34 seg

Música de 2:51 min



Música de 59:04 min

Resultados: Ao analisar os gráficos, constatamos que a diferença de desempenho entre utilizar apenas um worker e o número máximo, possível de testar, de workers é de aproximadamente 60-65% no tempo de processamento da música.

Conclusões

Com base nos resultados obtidos, concluímos que conseguimos desenvolver com sucesso um serviço web com API REST para o processamento eficiente de ficheiros de áudio. Através da separação em partes menores e identificação de instrumentos, conseguimos criar novos ficheiros completos.

A solução distribuída demonstrou ter a capacidade de atender a múltiplos clientes simultaneamente, oferecendo rapidez e escalabilidade. Além disso, a implementação de mecanismos de gestão de falhas contribuiu para a confiabilidade do serviço.

Em resumo, alcançamos plenamente os objetivos propostos, proporcionando um serviço eficaz.