Computação Distribuída
Projeto Final

Joao Santos – 110555
Bernardo Pinto - 105926

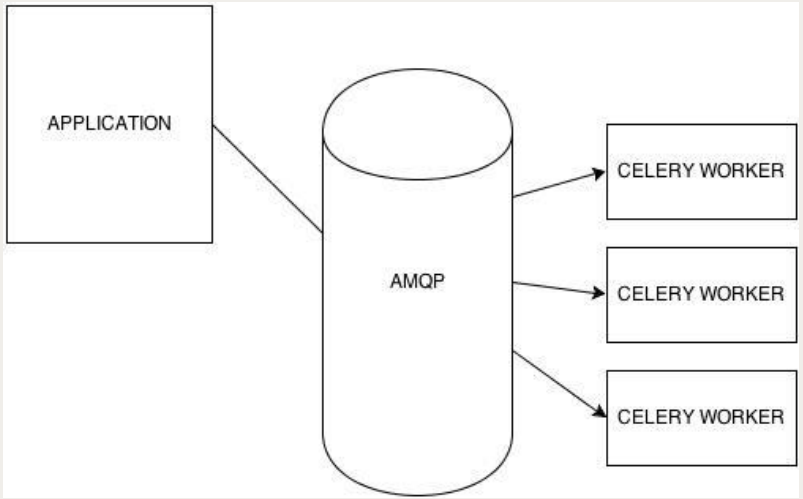
Regente:
Prof. Diogo Gomes

Licenciatura em Engenharia Informática

universidade de aveiro

Arquitetura

A aplicação envia para o broker o pedido de tarefa



O broker envia os pedidos para cada celery worker

Cada celery worker realiza a tarefa recebida

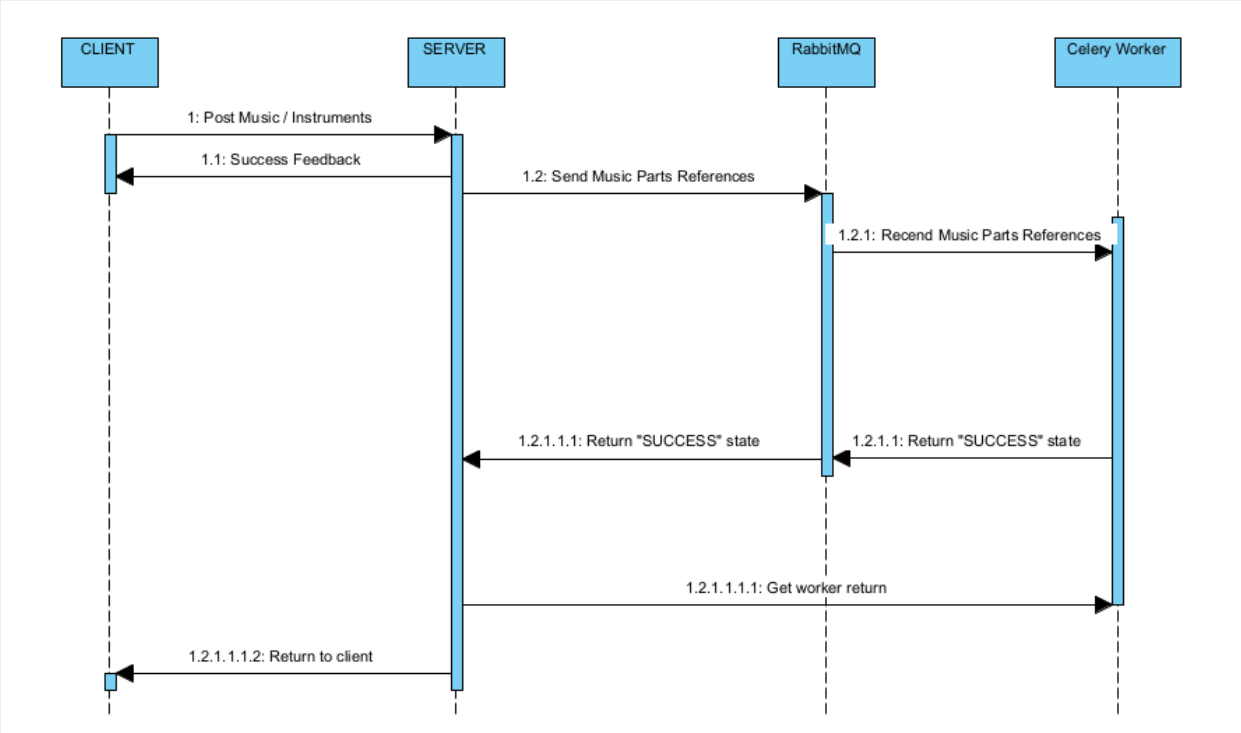
Implementação

Começamos por definir como os clientes iriam comunicar com o servidor, para isso usamos o Flask e o protocolo HTTP em que definimos rotas para direcionar os inputs dos clientes para o servidor de forma a seguir a API REST definida. Em seguida, após o cliente conseguir enviar o ficheiro de musica pretendido para o servidor e selecionar que tipos de tracks quer processar (drums, vocals, bass e other), o sistema divide a música em partes de 2 minutos e envia cada parte separadamente para o celery, com o objetivo de maximizar a tarefa de processamento, ou seja, ao invés de 1 worker processar a musica inteira, temos **n** workers a processar **m** partes menores da mesma musica.

Após partir a musica, o servidor envia, através da função **.delay()**, o pedido de processamento assíncrono de cada parte para a fila do broker, e então é gerado um ID para o pedido e este fica à espera que o worker venha buscar a parte da música para ser processada. Em seguida, após o processamento da musica, os workers guardam o resultado em um "backend" de armazenamento temporário que por fim é acedido pelo servidor usando a função **.get**(), de forma que o servidor não fique a espera ou realizando requisições consecutivas para atualizar o estado do processamento.

Por fim, após receber as tracks processadas, utilizamos a concatenação de **AudioSegment's** para agrupar dados referente a uma mesma track e a funcao **overlay()** do **pydub** para juntar os instrumentos selecionados pelo cliente em um único arquivo para ser disponibilizado ao cliente.

Protocolo


CLIENT
SERVER
RabbitMQ
Celery Worker
1: Post Music / Instruments
1.1: Success Feedback
1.2: Send Music Parts References
1.2.1: Recend Music Parts References
1.2.1.1.1: Return "SUCCESS" state
1.2.1.1: Return "SUCCESS" state
1.2.1.1.1.1: Get worker return
1.2.1.1.1.2: Return to client

Tolerância a falhas

- Usando o próprio Celery e RabbitMQ e definindo a seguinte linha, conseguimos garantir que se um worker falhar, o RabbitMQ envia novamente a mensagem para o fim da queue.

```
celery.conf.task_acks_late = True
```

Resultados

Ao testar o sistema com o ficheiro music.mp3 que tem duração de 59 minutos, o grupo chegou aos seguintes resultados aproximados para o termino do processamento:

1 worker	2 workers	4 workers
2 h 7 m	1 h 5 m	40 m