

Sistema de Streaming Distribuído

Arquitetura de Microsserviços com RabbitMQ

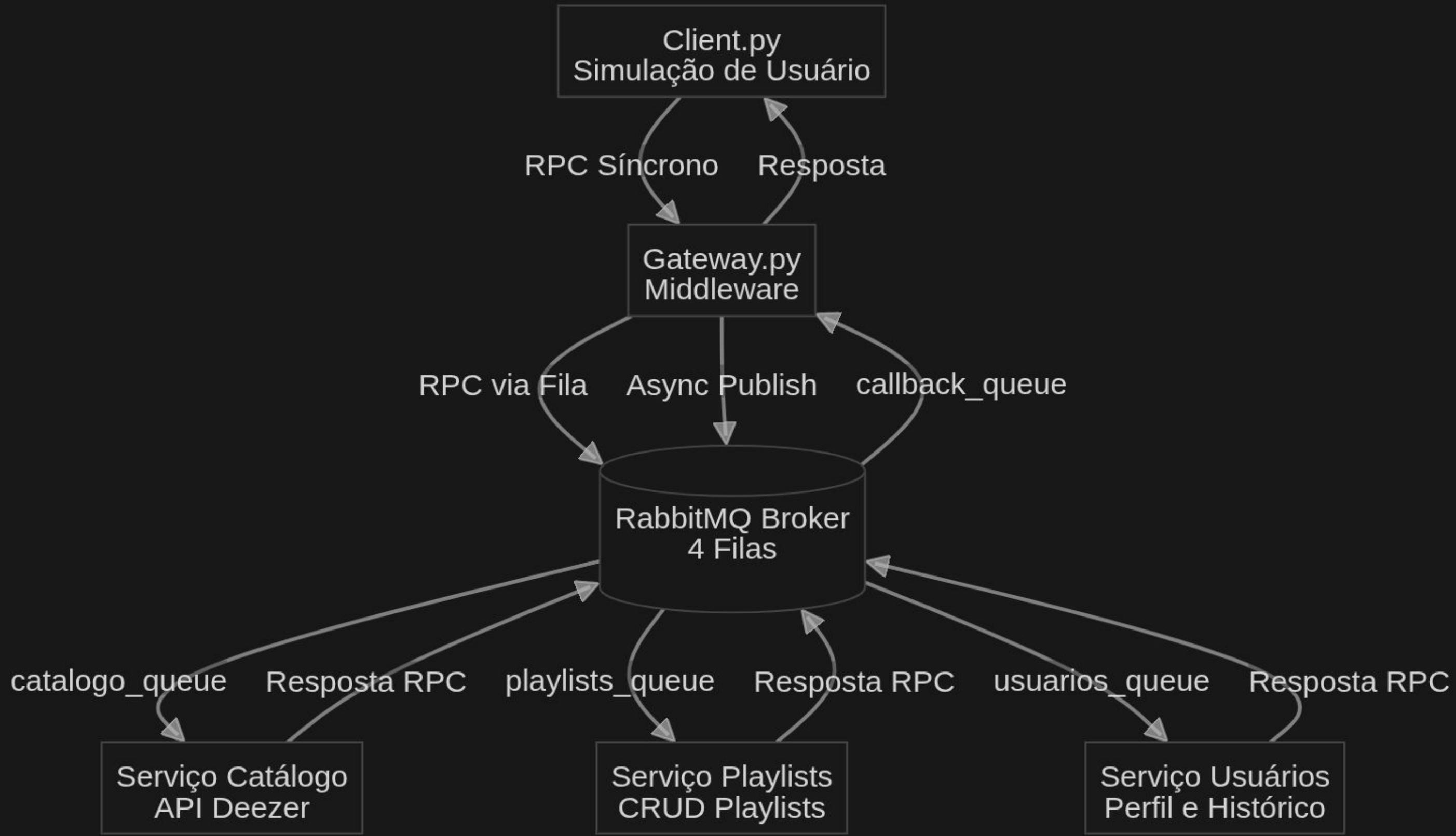
Uma implementação prática de conceitos de sistemas distribuídos utilizando Python, demonstrando comunicação assíncrona, RPC e mensageria através de um serviço de streaming de música escalável e resiliente.

Visão Geral do Sistema

Nossa arquitetura é baseada em microsserviços independentes que se comunicam através de filas de mensagens, garantindo desacoplamento, escalabilidade e tolerância a falhas.



Cada componente executa em processo separado, permitindo deploy, escala e manutenção independentes. O RabbitMQ atua como broker central, gerenciando todas as mensagens entre serviços.



Padrões de Comunicação Distribuída

Implementamos três paradigmas essenciais de sistemas distribuídos, cada um otimizado para diferentes cenários de uso:

RPC (Remote Procedure Call)

Comunicação Síncrona

Cliente aguarda resposta imediata do serviço remoto, como se fosse uma chamada local.

- Listar catálogo de músicas
- Buscar informações de perfil
- Consultar detalhes de playlist

Comunicação Assíncrona

Fire-and-Forget

Cliente envia mensagem e continua execução sem esperar confirmação.

- Registro de histórico de reprodução
- Atualização de estatísticas
- Logs de atividades do usuário

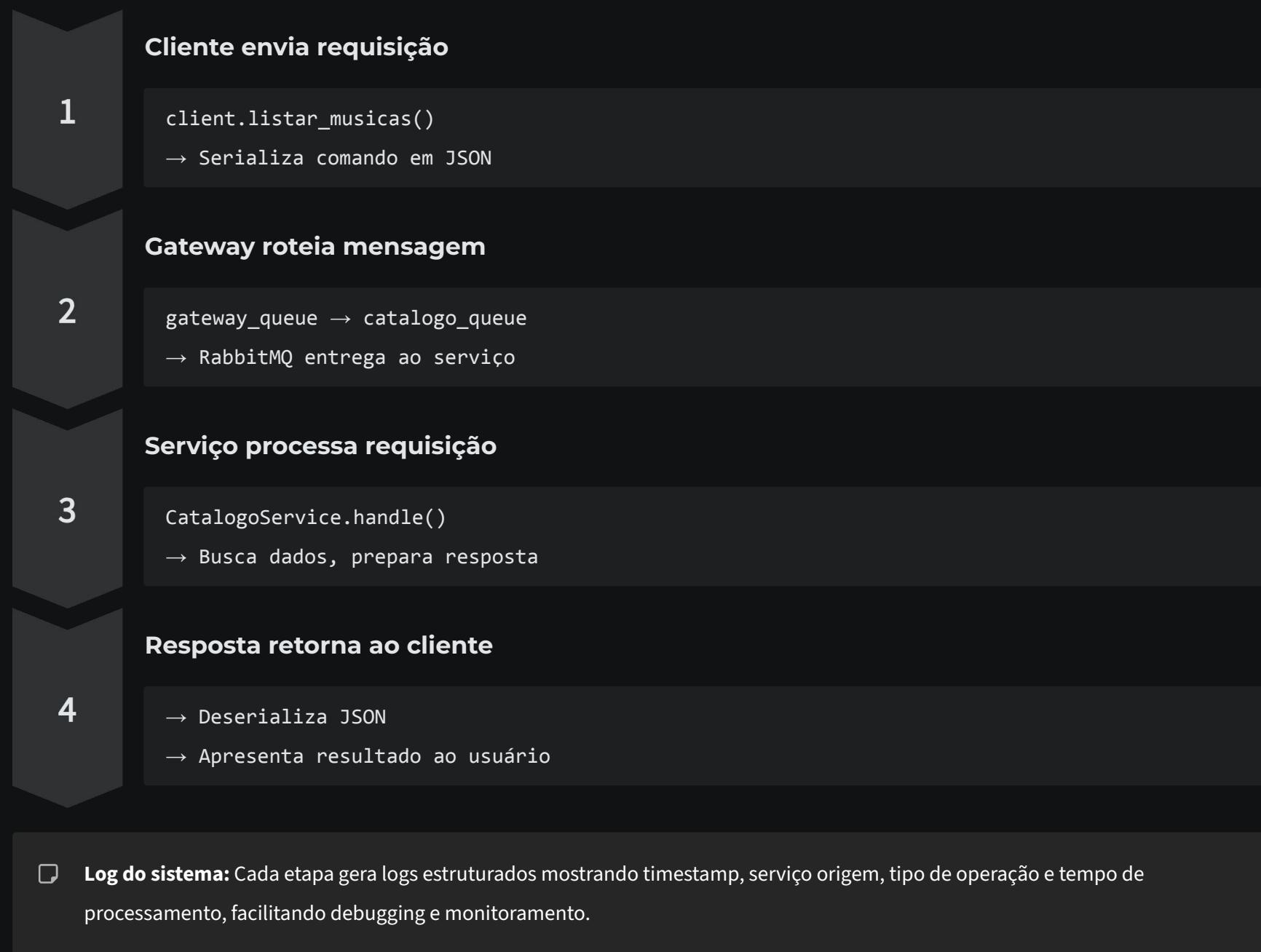
Comunicação Indireta Message Broker Pattern

RabbitMQ gerencia todas as mensagens, desacoplando completamente produtores e consumidores.

- Tolerância a falhas temporárias
- Balanceamento de carga automático
- Persistência de mensagens

Fluxo de Execução na Prática

Veja como uma requisição flui através do sistema distribuído, desde o cliente até o serviço especializado e retorno:



Tecnologias e Ferramentas

Core Technologies

Python 3.10+

Linguagem principal com suporte a type hints, async/await e bibliotecas robustas para sistemas distribuídos.

RabbitMQ + Pika

Message broker AMQP para comunicação confiável entre serviços, com client library Pika para integração Python.

JSON-RPC 2.0

Protocolo leve para chamadas remotas, facilitando serialização e padronização de mensagens.

Organização do Projeto

```
streaming-distribuido/
    ├── gateway/
    │   └── gateway_service.py
    ├── services/
    │   ├── catalogo/
    │   ├── playlists/
    │   └── usuarios/
    ├── client/
    │   └── cliente.py
    ├── common/
    │   └── rabbitmq_rpc.py
    └── start.sh
```

Execução simplificada:

```
chmod +x start.sh
./start.sh
```

Benefícios da Arquitetura Distribuída



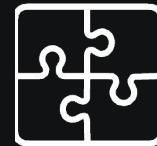
Escalabilidade Horizontal

Adicione mais instâncias de qualquer serviço conforme demanda aumenta. RabbitMQ distribui carga automaticamente entre workers disponíveis.



Resiliência e Tolerância a Falhas

Falha em um serviço não afeta outros componentes. Mensagens persistem até processamento bem-sucedido.



Desacoplamento Total

Serviços não conhecem uns aos outros diretamente. Gateway e broker abstraem toda complexidade de comunicação.



Deploy Independente

Atualize, corrija bugs ou adicione features em serviços individuais sem impactar sistema completo.



Manutenibilidade

Código modular e separado por contextos de negócio facilita entendimento, testes e evolução do sistema.



Flexibilidade Tecnológica

Serviços podem usar diferentes tecnologias e linguagens, desde que respeitem protocolo de mensageria.