## Design de APIs para Microsserviços

Um bom design de API é crucial em uma arquitetura de microsserviços, já que toda a comunicação entre os serviços ocorre por meio de chamadas de API. As APIs precisam ser eficientes para evitar o excesso de tráfego de rede e devem ter esquemas e versionamento bem definidos para que as atualizações não interrompam outros serviços.

## Dois Tipos de API

É importante fazer a distinção entre os dois principais casos de uso de APIs em uma arquitetura de microsserviços:

- **APIs Públicas:** São utilizadas pelos aplicativos do cliente, como navegadores ou aplicativos móveis. Para garantir a compatibilidade, a abordagem mais comum é o uso de rest sobre http.
- **APIs de Backend:** Usadas para a comunicação interna entre os microsserviços. Nesse cenário, o desempenho é um fator crítico, e alternativas como **gRPC, Apache Avro e Apache Thrift** (open sources) são recomendadas por serem, em geral, mais eficientes que o HTTP.

## Considerações de Design: REST vs. RPC

A escolha do estilo da API envolve a análise de algumas questões.

- **REST:** Seu foco é em recursos, o que pode ser uma forma natural de representar o modelo de domínio. Ele utiliza uma interface uniforme baseada nos verbos HTTP, o que promove a escalabilidade e a evolução do sistema.
- **RPC:** É mais orientado a operações ou comandos. Como suas interfaces se assemelham a chamadas de método locais, é preciso ter cuidado para não criar APIs com excesso de chamadas.

## A Recomendação Principal

A recomendação geral é:

Escolha REST sobre HTTP, a menos que você precise dos benefícios de desempenho de um protocolo binário.

O REST sobre HTTP não requer bibliotecas especiais, tem baixo acoplamento e é compatível com navegadores, além de possuir um vasto ecossistema de ferramentas de suporte. No entanto, é importante realizar testes de carga e desempenho no início do desenvolvimento para garantir que ele atenda aos requisitos do seu cenário.

## Pilares do Design da API

- **Modelagem de Domínio:** A API deve ser um contrato que representa o negócio, e não um reflexo dos detalhes internos do banco de dados ou do código-fonte.
- **Versionamento:** Se for necessário fazer uma alteração que quebre a compatibilidade da API, é preciso introduzir uma **nova versão**. É recomendado continuar a dar suporte à versão anterior e permitir que os clientes escolham qual versão utilizar.
- **Idempotência:** Uma operação é considerada idempotente se puder ser chamada várias vezes sem gerar efeitos colaterais adicionais após a primeira chamada. Isso é útil para aumentar a resiliência do sistema, pois permite que uma operação seja repetida com segurança.
    - Os métodos

GET, PUT e DELETE devem ser idempotentes.

- Não há garantia de que os métodos

POST sejam idempotentes.