A high-performance, open-source universal RPC framework.

Contexto Histórico

Do conceito original à revolução open-source.

1981

Xerox PARC

Primeiro uso comercial. Bruce Jay Nelson cunha o termo **RPC** (Remote Procedure Call).

2000s

Google Stubby

Infraestrutura interna do Google. Conectava microsserviços, mas era muito acoplado à infra proprietária.

2015

gRPC

Google lança o gRPC como projeto **Open Source**. A evolução "desacoplada" do Stubby.

O que é gRPC?

Um framework de chamada de procedimento remoto de alta performance.

- **Função Local:** Chame métodos em outro servidor como se fossem locais.
- **Agnóstico:** Funciona entre diferentes linguagens (Go, Java, Python).
- **Tipagem Forte:** Contratos definidos evitam erros de interpretação.

Performance Crítica

Ideal para comunicação entre microsserviços e sistemas onde "cada milissegundo importa".

Protocol Buffers (Protobuf)

Mecanismo de serialização binária eficiente e neutro.

Contrato .proto: Define a estrutura de dados e serviços.

Code Gen: Compilador gera stubs de cliente e servidor automaticamente.

Binário: Contexto separado do conteúdo. Usa índices numéricos em vez de nomes de campos.

```proto
// Exemplo de definição .proto
syntax = "proto3";

message Produto {
  string partnumber = 1; // Índice 1
  int32 quantidade = 2;  // Índice 2
}
```

Por que Protobuf é mais rápido?

A diferença está no que trafega pela rede.

JSON (Texto)

```
{ "partnumber": "Boinas", "quantidade": 150 }
```

Envia repetidamente os nomes das chaves ("partnumber", "quantidade") e caracteres estruturais ({, ", :}) a cada requisição.

Protobuf (Binário)

Tag 1 | Len 6 | "Boinas"

Envia apenas os **índices** e os **valores**. O contexto (nomes dos campos) já está pré-definido no contrato local.

Como representar a mensagem

JSON (Texto)

```
{
  "partnumber": "Boinas",
  "quantidade": 150
}
```

Protobuf (Binário)

```
0A06426F696E6173109601
```

HTTP/2

Superando as limitações do HTTP/1.1.

Problema (HTTP/1.1)

- ✕ Uma mensagem por conexão (Head-of-line blocking).
- ✕ Handshake repetitivo e lento.
- ✕ Headers repetidos (texto puro).

Solução (HTTP/2)

- ✓ **Multiplexação:** Várias requisições em uma única conexão TCP.
- ✓ **Full Duplex:** Envio e recebimento simultâneo (Streaming).
- ✓ **HPACK:** Compressão de headers (envia apenas o delta/diferença).

💡Isso resolve o limite de conexões simultâneas dos navegadores e reduz drasticamente o consumo de CPU.

Metodos de comunicação

- **Unary**
  Requisição e resposta simples (Tradicional).
- **Server Streaming**
  Cliente pede, servidor envia fluxo contínuo.
- **Client Streaming**
  Cliente envia fluxo, servidor responde no final.
- **Bidirectional**
  Troca simultânea de mensagens via canal único.

gRPC vs. Padrão REST

Característica	REST	gRPC
Eficiência	É mais pesado e lento. Trafega texto (JSON) e o protocolo HTTP 1.1 abre e fecha conexões constantemente.	É leve (binário/Protobuf), economiza CPU e usa uma única conexão para tudo (HTTP/2).
Segurança de contrato	Flexível, você pode mudar um campo no JSON e quebrar o cliente sem aviso.	Garante que o cliente e o servidor obedeçam às mesmas regras. Se alguém quebrar o contrato, o código nem compila.
Integração	Universal, funciona em qualquer lugar que aceita HTTP/JSON.	Gera código automático e tipado para várias linguagens. Garante segurança nos dados, mas cria dependência do arquivo de contrato.

Limitações do gRPC

Quando **não** é a melhor opção?

Suporte no Browser

Browsers não expõem controle total do HTTP/2. Exige um proxy intermediário (gRPC-Web) para funcionar no front-end.

Legibilidade Humana

Por ser binário, você não consegue ler a requisição facilmente. O debug é mais complexo que ler um JSON simples.