Google Remote procedure call (gRPC)

Ian Victor

O que é gRPC e como ele funciona?

- **gRPC (Google Remote Procedure Call) é um framework open-source de RPC criado pelo Google.**
- **Utiliza HTTP/2 para comunicação eficiente entre aplicações distribuídas.**
- **Baseia-se em arquivos .proto (Protobuf) para definir serviços e gerar código para várias linguagens.**

PROTOBUF(Protocol Buffers)

O que é Protobuf e como se compara a outros formatos

Protobuf (Protocol Buffers) é um formato de serialização binário, rápido e eficiente.

Comparação com outros formatos:

JSON Mais legível, mas menos eficiente.

XML Maior consumo de banda e processamento.

gRPC X REST

	Protocolo de transporte:	**Formato de dados:**	**Suporte a streaming:**	**Compatibilidade:**
gRPC	Usa HTTP/2 (mais rápido e eficiente).	Usa Protobuf (binário, mais compacto).	Suporta streaming bidirecional.	É ideal para microserviços.
REST	Usa HTTP/1.1 (requisições independentes).	Usa JSON (mais pesado, mas legível).	É baseado apenas em requisição-resposta.	É melhor para APIs públicas.

Vantagens e Limitações do gRPC

- ✓ Desempenho superior (mensagens menores e rápidas).
- ✓ Suporte a streaming bidirecional.
- ✓ Multiplataforma (suporte a diversas linguagens).
- ✓ Otimizações via HTTP/2 (latência reduzida).
- ✗ Curva de aprendizado mais complexa.
- ✗ Depuração difícil (mensagens não são legíveis sem ferramentas).
- ✗ Suporte limitado no navegador (REST é mais compatível).

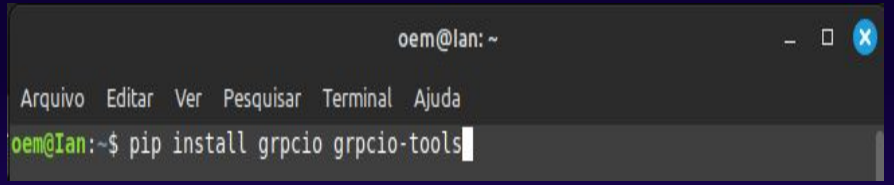
Aplicação Básica utilizando gRPC

Demonstração de uma aplicação simples usando gRPC em Python, onde um cliente envia dois números para um servidor, que realiza a soma e retorna o resultado. Vamos explorar como esse processo funciona, desde a definição do serviço até a comunicação entre cliente e servidor.

1º

Instalar as bibliotecas necessárias para rodar gRPC no python

No terminal, execute:



2º

Criação do arquivo .proto

Este é o arquivo que define o serviço e os tipos de mensagens.

Este é o arquivo que define a estrutura do serviço gRPC usando a linguagem de definição de protocolo (protobuf). Ele contém:

- Um serviço chamado Calculator com um método Add, que recebe uma AddRequest e retorna uma AddResponse.
- Mensagens (AddRequest e AddResponse) que representam a estrutura dos dados enviados e recebidos.

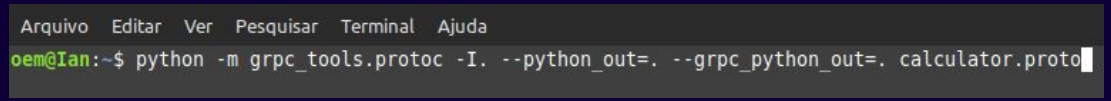
calculator.proto

```
1   syntax = "proto3";
2
3   package calculator;
4
5   service Calculator {
6     rpc Add (AddRequest) returns (AddResponse);
7   }
8
9   message AddRequest {
10    int32 num1 = 1;
11    int32 num2 = 2;
12  }
13
14  message AddResponse {
15    int32 result = 1;
16  }
17
```

3º

Gerar os Arquivos gRPC (calculator_pb2.py e calculator_pb2_grpc.py)

No terminal, execute:



calculator_pb2.py (gerado automaticamente)

Este arquivo é gerado pelo compilador do protocolo (protoc) a partir do calculator.proto. Ele contém:

Classes Python que representam as mensagens definidas no .proto (AddRequest e AddResponse). As definições dos serviços e métodos baseados no protocolo.

calculator_pb2_grpc.py (gerado automaticamente)

Este arquivo contém as classes do cliente e servidor gRPC:

- CalculatorStub → Classe que permite o cliente chamar métodos do servidor.
- CalculatorServicer → Classe base para o servidor implementar o serviço.
- add_CalculatorServicer_to_server → Método para registrar o serviço no servidor.

4º

Criar o servidor (server.py)

server.py (servidor gRPC)

Este é o código do servidor que implementa o serviço Calculator. Ele contém:

- CalculatorServicer → Implementa o método Add, que soma dois números recebidos do cliente e retorna o resultado.
- serve() → Inicializa um servidor gRPC na porta 50051 e aguarda conexões do cliente.

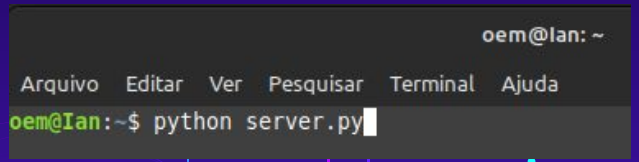
```python
import grpc
from concurrent import futures
import calculator_pb2
import calculator_pb2_grpc

class CalculatorServicer(calculator_pb2_grpc.CalculatorServicer):
    def Add(self, request, context):
        result = request.num1 + request.num2
        return calculator_pb2.AddResponse(result=result)

def serve():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    calculator_pb2_grpc.add_CalculatorServicer_to_server(CalculatorServicer(), server)
    server.add_insecure_port('[::]:50051')
    server.start()
    server.wait_for_termination()

if __name__ == '__main__':
    serve()
```

Inicie o servidor com o seguinte comando:



5º

Criar o cliente (cliente.py)

client.py (cliente gRPC)

Este é o código do cliente que se conecta ao servidor e envia dois números para serem somados. Ele:

- Cria um canal gRPC para comunicação com o servidor (localhost:50051).
- Chama o método Add no servidor enviando dois números (num1 e num2).
- Imprime o resultado da soma recebido do servidor.

client.py > ...

```python
import grpc
import calculator_pb2
import calculator_pb2_grpc

def run(num1, num2):
    with grpc.insecure_channel('localhost:50051') as channel:
        stub = calculator_pb2_grpc.CalculatorStub(channel)
        response = stub.Add(calculator_pb2.AddRequest(num1=num1, num2=num2))
    print(f"Result: {response.result}")

if __name__ == '__main__':
    # Get user Input
    num1 = int(input("Please input num1: "))
    num2 = int(input("Please input num2: "))
    run(num1, num2)
```

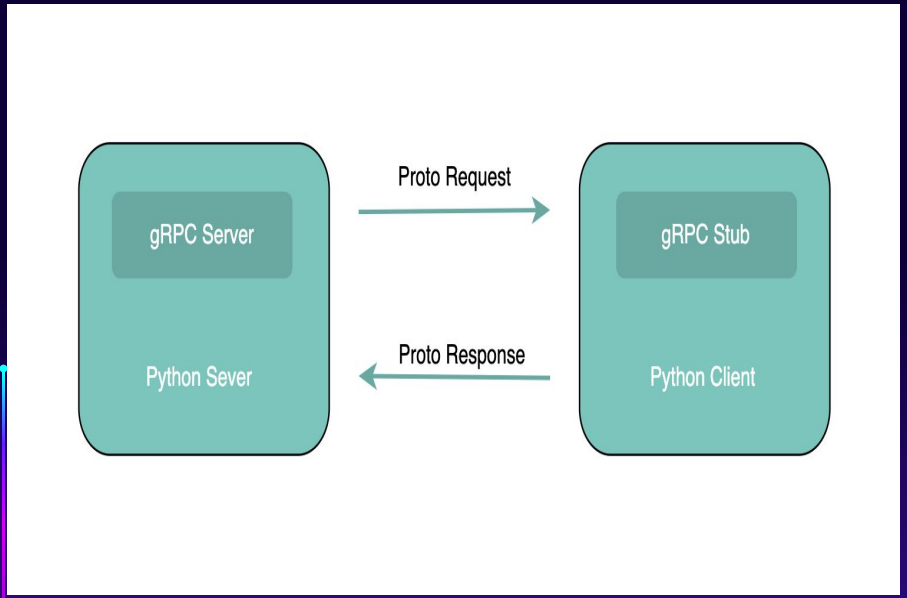
Agora, com o servidor rodando, execute em um novo terminal o seguinte comando:

```
oem@Ian: ~
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
oem@Ian:~$ python client.py
```

Resultado esperado:

```
sh-5.2$ python client.py
Please input num1: 45
Please input num2: 45
Result: 90
```

Contextualizando...



1. O servidor (server.py) inicia e fica esperando conexões.
2. O cliente (client.py) solicita ao usuário dois números e os envia ao servidor.
3. O servidor (server.py) soma os números e envia o resultado de volta.
4. O cliente (client.py) exibe o resultado na tela.

CONCLUSÃO

gRPC é uma excelente alternativa ao REST para comunicação eficiente entre microservices, oferecendo vantagens significativas em desempenho, suporte a streaming e compatibilidade com diferentes linguagens. No entanto, pode não ser a melhor escolha para aplicações web tradicionais devido à falta de suporte direto nos navegadores.

Referências

https://aws.amazon.com/pt/compare/the-difference-between-grpc-and-rest/#:~:text=Leia%20sobre%20APIs%20%C2%BB-,O%20que%20%C3%A9%20gRPC%3F,gRPC%20%C3%A9%20uma%20implementa%C3%A7%C3%A3o%20espec%C3%ADfica.

https://github.com/barkhayot/grpc-python-example/blob/main/calculator_pb2_grpc.py

Obrigado pela atenção!