# Documentação Técnica Detalhada
# `bigfs_pb2_grpc.py`

wender13

1 de julho de 2025

## Sumário

# 1 Introdução e Papel na Arquitetura

O arquivo `bigfs_pb2_grpc.py` é um dos dois arquivos gerados automaticamente pelo compilador gRPC a partir do `bigfs.proto`. É fundamental entender que este arquivo **não deve ser editado manualmente**.

Se o `bigfs_pb2.py` representa os "tijolos"(as estruturas de dados), este arquivo representa o **"encanamento"e a "fiação"** do nosso sistema. Ele contém todo o código de baixo nível que permite que o cliente e o servidor conversem através da rede, executando as funções remotas (RPCs) que definimos.

Para cada `service` no arquivo `.proto`, o compilador gera três componentes principais neste arquivo. Usaremos o `GatewayService` como exemplo para explicar o padrão.

# 2 O Lado do Cliente - A Classe `Stub`

```python
class GatewayServiceStub(object):
    def __init__(self, channel):
        self.UploadFile = channel.stream_unary(...)
        self.GetDownloadMap = channel.unary_unary(...)
        self.ListFiles = channel.unary_unary(...)
```

Listing 1: Exemplo de Stub gerado para GatewayService

**Propósito:** Funcionar como o **"Controle Remoto"para o Cliente**.

**Como Funciona:** O código do nosso `client.py` cria uma instância desta classe, passando para ela um canal de comunicação gRPC (‘channel‘). O objeto `Stub` resultante possui métodos que correspondem exatamente aos RPCs definidos no serviço (‘UploadFile‘, ‘GetDownloadMap‘, etc.).

**Detalhes da Implementação:** A linha `self.GetDownloadMap = channel.unary_unary(...)` faz o trabalho pesado. Ela "conecta"a chamada do método a um tipo específico de RPC (neste caso, ‘unary$_u nary‘, quesignifica”umpedido, umaresposta”).Elatambmdefineoendpointde

# 3 O Lado do Servidor - A Classe `Servicer`

```python
class GatewayServiceServicer(object):
    def UploadFile(self, request_iterator, context):
        context.set_code(grpc.StatusCode.UNIMPLEMENTED)
        context.set_details('Method not implemented!')
        raise NotImplementedError('Method not implemented!')
    # ... outros métodos com a mesma implementação ...
```

Listing 2: Exemplo de Servicer gerado para GatewayService

**Propósito:** Servir como o **"Esqueleto"ou "Template"para o Servidor**.

**Como Funciona:** É uma classe base abstrata. Nossas próprias classes de servidor (como a classe `GatewayService` no arquivo `gateway_server.py`) devem **herdar** desta classe gerada.

**Implementação Obrigatória:** O código gerado vem com implementações vazias que imediatamente lançam um erro `NotImplementedError` e retornam um status gRPC `UNIMPLEMENTED`. Isso nos força, como desenvolvedores, a **sobrescrever** esses métodos em nossa própria classe para implementar a lógica de negócio real. É assim que o gRPC garante que nosso servidor cumpre o contrato definido no `.proto`.

# 4 A Conexão - A Função `add_*_to_server`

```
def add_GatewayServiceServicer_to_server(servicer, server):
    rpc_method_handlers = {
        'UploadFile': grpc.stream_unary_rpc_method_handler(...),
        'GetDownloadMap': grpc.unary_unary_rpc_method_handler(...),
        # ...
    }
    generic_handler = grpc.method_handlers_generic_handler(...)
    server.add_generic_rpc_handlers((generic_handler,))
```

Listing 3: Exemplo de função de registro para GatewayService

**Propósito:** Atuar como a **"Cola"** que conecta a nossa lógica de servidor ao servidor gRPC principal.

**Como Funciona:** Esta função é chamada uma vez durante a inicialização do servidor. Ela pega a nossa classe de implementação (o esqueleto que preenchemos) e a registra no objeto principal do servidor gRPC.

**Roteamento Interno:** Essencialmente, ela constrói uma tabela de roteamento interna. Ela diz ao servidor: "Quando uma requisição de rede chegar para o endpoint '/bigfs.GatewayService/UploadFile', execute o método 'UploadFile' do objeto 'servicer' que foi registrado".

# 5 Conclusão

Este arquivo gerado, embora complexo, é o que torna o gRPC poderoso. Ele automatiza toda a infraestrutura de comunicação de baixo nível — serialização de dados, roteamento de rede e invocação de métodos — permitindo que o desenvolvedor se concentre apenas na lógica de negócio da aplicação, tanto no cliente quanto no servidor.