gRPC zastosowanie w pythonie

Tworzenie pliku .proto

W pliku .proto deklarujemy naszą usługę która będzie zawierała wsztkie serwisy

```
syntax="proto3";

// pierwszy serwis do przywitania
service Greeter{
  // pozdrowienia
  // unary service
  rpc SayHello(MessageRequest) returns (MessageResponse) {}
}

// na wejściu będę miał imię i nazwisko
message MessageRequest{
  string name = 1;
  string surname = 2;
}

// na wyjściu będę miał odpowiedz
message MessageResponse{
  string message = 1;
}
```

Jak widzimy w przykładzie powyżej zadeklarowaliśmy jedną usługę o nazwie ***Unary*** która zawiera tylko jeden serwis ***GetServerResponse*** który ma na wejściu typ Message i na wyjściu MessageResponse.
Plik .proto zawiera także protocol buffer message type definition oraz response type użyte w naszym serwisie

Po stworzeniu pliku .proto musimy przejść do stworznie stubs.

```
python3 -m grpc_tools.protoc -I protos --python_out=. --grpc_python_out=.
protos/greet.proto
```

utworzą się nam wtedy 2 pliki:

- _pb2.py

- _pb2_grpc.py
  które tworzą kod zawierający:
- klasy wiadomości zdefiniowane w .proto
- klasy dla serwisu zdefiniowanego w .proto:
  - Stubs - które mogą być uzyte przez klienta żeby wywołać RPCs
  - Servicer - który definiuje interfejs do implementowania serwisu
- funkcje dla serwisun zdefiniowanego w .proto
  - add"..." Servicer_to_server - który dodaje Servicer do grpc.server

Tworzenie serwera

Tworzenie serwera sprowadza się do pracy w 2 rzeczami:

- Implementacji naszego interfejsu serwisu generowanego z definicji serwisu z wykorzystaniem funkcji która przeprowadza faktyczną "prace" w serwisie
- Działania serwera gRPCtak żeby nasłuchiwał zapytania i przekazywał odpowiedzi

1. Importujemy do naszego serwera moduł gRPC oraz 2 pliki które stworzyliśmy

```
import grpc
import greet_pb2
import greet_pb2_grpc
from concurrent import futures
```

2. Implementacja serwisu
   Tworzymy klase która zapewni implementacją metod z serwisu greeter

```
class GreeterServicer(greeter_pb2_grpc.GreeterServicer)
```

3. Tworzenie funkcji odpowiedzialnych za zapytania jakie tworzyliśmy w pliku .proto

```
# unary service
# serwer odpowiada na klienta
class GreeterServicer(greet_pb2_grpc.GreeterServicer):
   def SayHello(self, request, context):
       return greet_pb2.MessageResponse(message=f"{request.name},
{request.surname}")
```

zwracamy do messageresponse imię i nazwisko które z zapytania do serwera

4. uruchamianie naszego serwera:

```
def
```

```
def serve():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    greet_pb2_grpc.add_GreeterServicer_to_server(GreeterServicer(), server)
    server.add_insecure_port("localhost:50051")
    server.start()
    server.wait_for_termination()

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

Startujemy nasz serwer

Klient

1. Importujemy potrzebne biblioteki i pliki

```
import grpc
from protos import greet_pb2_grpc, greet_pb2
```

2. Tworzymy fynkcję odpowiedzialną za wysłanie zapytania do serwera

```
def run():
    with grpc.insecure_channel("localhost:50051") as channel:
        stub = greet_pb2_grpc.GreeterStub(channel)
        response = stub.SayHello(greet_pb2.MessageRequest(name="Filip",
surname="Walkowicz"))
        print("hello" + response.message)
```

Podajemy tutaj dane jakie chcemy przesłać oraz ustabiamy stub

Uruchamiamy serwer i klienta

```
python3 server.py
python3 client.py
```

Wynikiem będzie:

```
hello Filip, Walkowicz
```

Źródła

- https://grpc.io/docs/languages/python/basics/

- https://developers.google.com/protocol-buffers/docs/proto3
- https://realpython.com/introduction-to-python-generators/
- https://realpython.com/python-microservices-grpc/
- https://www.cloudbees.com/blog/using-grpc-in-python