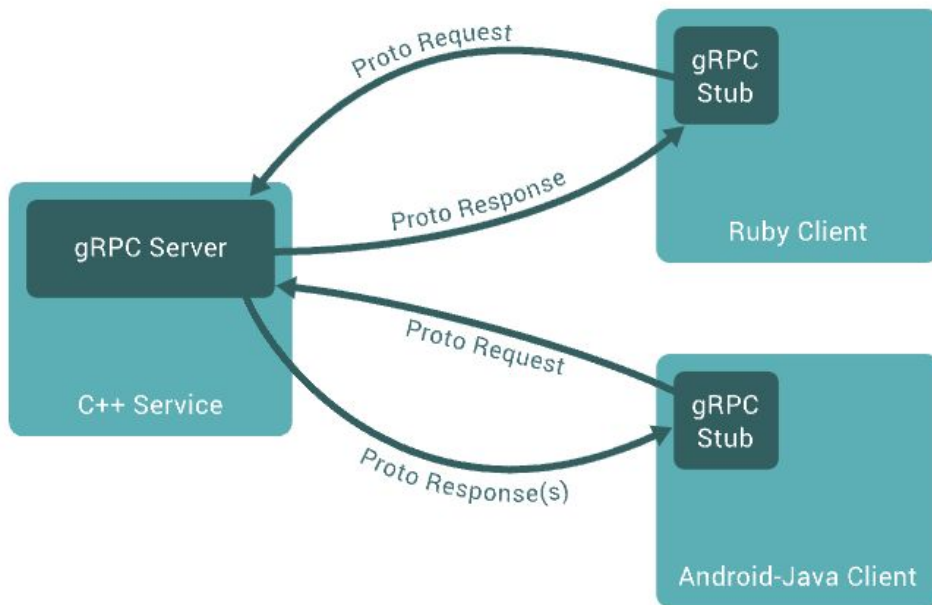# gRPC

remote procedure call framework

How it works

Cross-language tool

- C# / .NET
- C++
- Dart
- Go
- Java
- Kotlin
- Node
- Objective-C
- PHP
- Python
- Ruby

Protocol buffer

.proto files

https://protobuf.dev/programming-guides/proto3/

```
syntax = "proto3";

service Greeter{

  rpc SayHello(Request) returns (Response) {}

}

message Request{
  string name = 1;
}

message Response{
  string message = 1;
}
```

Example service configuration

```python
def serve():
    port = '50051'
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    helloworld_pb2_grpc.add_GreeterServicer_to_server(Greeter(), server)
    server.add_insecure_port('[::]:' + port)
    server.start()
    print("Server started, listening on " + port)
    server.wait_for_termination()
```

Example service class

```
class Greeter(helloworld_pb2_grpc.GreeterServicer):

    👤 Yash Tibrewal

    def SayHello(self, request, context):
        return helloworld_pb2.HelloReply(message='Hello, %s!' % request.name)
```

Client call example

```python
def run():
    print("Will try to greet world ...")
    with grpc.insecure_channel('localhost:50051') as channel:
        stub = helloworld_pb2_grpc.GreeterStub(channel)
        response = stub.SayHello(helloworld_pb2.HelloRequest(name='you'))
        print("Greeter client received: " + response.message)
```

Stream usage in single rpc

```
service Service {

  rpc StreamCall(stream StreamCallRequest) returns (stream StreamCallResponse);
}
```

Bidirectional streaming

Server handle example

```python
def RouteChat(self, request_iterator, context):
    prev_notes = []
    for new_note in request_iterator:
        for prev_note in prev_notes:
            if prev_note.location == new_note.location:
                yield prev_note
        prev_notes.append(new_note)
```

```python
def ListFeatures(self, request, context):
    left = min(request.lo.longitude, request.hi.longitude)
    right = max(request.lo.longitude, request.hi.longitude)
    top = max(request.lo.latitude, request.hi.latitude)
    bottom = min(request.lo.latitude, request.hi.latitude)
    for feature in self.db:
        if (feature.location.longitude >= left and
                feature.location.longitude <= right and
                feature.location.latitude >= bottom and
                feature.location.latitude <= top):
            yield feature
```

Client call example

```python
for feature in stub.ListFeatures(rectangle):
```

```python
for received_route_note in stub.RouteChat(sent_route_note_iterator):
```

Async

```python
import logging
import asyncio
from grpc import aio

import helloworld_pb2
import helloworld_pb2_grpc


class Greeter(helloworld_pb2_grpc.GreeterServicer):

    async def SayHello(self, request, context):
        return helloworld_pb2.HelloReply(message='Hello, %s!' % request.name)


async def serve():
    server = aio.server()
    helloworld_pb2_grpc.add_GreeterServicer_to_server(Greeter(), server)
    listen_addr = '[::]:50051'
    server.add_insecure_port(listen_addr)
    logging.info("Starting server on %s", listen_addr)
    await server.start()
    await server.wait_for_termination()


if __name__ == '__main__':
    logging.basicConfig(level=logging.INFO)
    asyncio.run(serve())
```

First task

Your first task is to implement the base hello world procedure.

Look on helloworld.proto there is all you need to implement

server and client communication

Second task

Your second task is to create server that perform math tasks:

- addition
- subtraction
- multiplication
- division
- exponentiation

Third task

Your last task is to improve the usage of the server from the second task with ping-pong stream type that allow sending more data in a single procedure call. Add some error handling using context.

https://adityamattos.com/grpc-in-python-part-3-implementing-grpc-streaming

Here you can find them all and also presentation.

git: https://github.com/AdrianKarolewski/gRPC-tutorial

https://grpc.io/docs/languages/python/quickstart/