e-Learning: gRPC

In this e-Learning session, we are going to learn the basics of gRPC, which is a popular standard used for communication between services.

What is gRPC?

Google proposed gRPC as an open-source standard in 2015 to be an evolution of an internal project called Stubby, which had been used for connect its various microservices. In general, gRPC provides a standarized way of defining services using an interface definition language (IDL) called **Protocol Buffers**, which is itself another standard. Protocol Buffers is usually used in gRPC both for defining the interfaces (and generating code for client and server in varios programming languages) and for defining the payloads themselves.

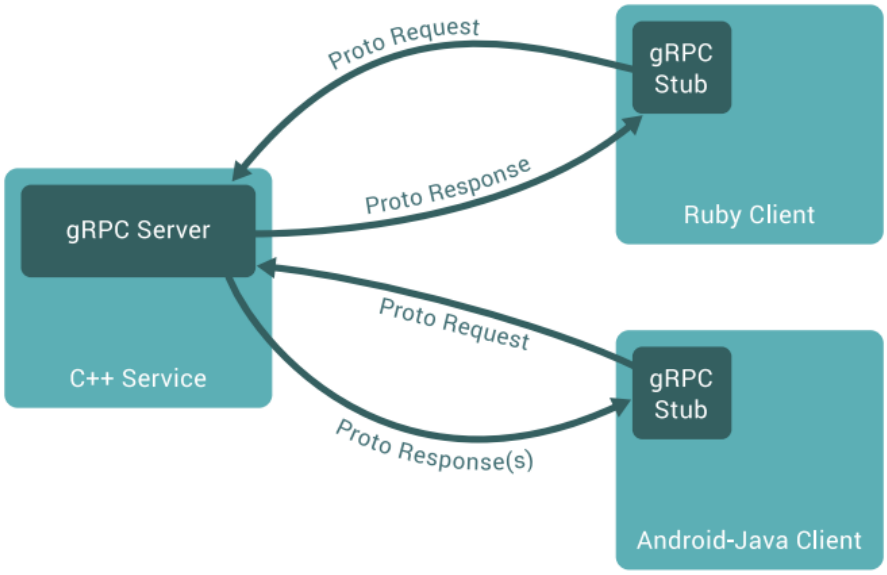

Figure 1: General gRPC workflow (see here for details.)

The main workflow for writing a gRPC service would be:

1. Write the service specification in Protocol Buffers.
2. Generate client and server stubs using the `protoc` compiler.
3. Adapt client and server and implement the business logic.
4. Deploy server.

An example gRPC-based service communication is depicted in Figure 1. Here, a server was implemented in C++ and it communicates with two clients: one written in Ruby (depicted on the top) and another one in Java for Android (bottom). Both requests and responses are in Protocol Buffers format. All messages are transmitted using HTTP/2 as the underlying protocol.

Core gRPC concepts

To learn more about how gRPC works, please visit the following URL: What is gRPC? and go through the pages Introduction to gRPC and Core concepts. Then take a look to this Protocol Buffers overview.

Coding activities

We will use the code in the examples repository (`DS_Examples`) which is located in the `grpc` folder.

1. First take a look at the file `example.proto`. This file defines a service called `CustomerService` that exposes two methods:
   - `AddCustomer`: adds data for a new customer (a customer was defined in the `Customer` message). **This is a unary RPC operation**.
   - `SendPurchases`: sends data for a new `Purchase`. **This is a client streaming RPC**.
2. Now compile this protobuf definition using the command contained in the file `protoc_command.txt`. Two new files will be generated:
   - `example_pb2.py`: specifies how to serialize the data types `Purchase` and `Customer`.
   - `example_pb2_grpc.py`: includes the Python stub classes for the client and the server.
3. The server (and the client) are already implemented to include the generated stubs (by using the `import` statement).
   - Take a closer look at how the input value of `SendPurchases` is implemented using **iterators** both on the server and the client side.

   You can now run the server as a normal Python script in the console. It will just wait for connections. Then run the client. It will send two requests in total:
   - One `AddCustomer` request to add the customer `John Smith`.
   - One `SendPurchases` request to send **two** purchases from that customer.
4. **Exercise 1:** Change the return type of the `SendPurchases` method so that the client receives for each `Purchase` an object indicating if the purchase was successfully added to an existing customer or, when not, an error message.
5. **Exercise 2:** Add a new service method that uses a response stream from the server to receive all Customers with purchases that, in sum, exceed a given amount passed as parameter.

We will take a look at the solutions in the next session!