**Lab#06 gRPC**

Before start writing a code, today go through the general below documentation

[**Introduction to gRPC**](https://grpc.io/docs/what-is-grpc/introduction/)

[**Core concepts, architecture and lifecycle**](https://grpc.io/docs/what-is-grpc/core-concepts/)

[**Protocol Buffers**](https://developers.google.com/protocol-buffers/docs/overview)

**Starting gRPC with Python**

This guide gets you started with gRPC in Python with a simple working example.

Prerequisites

* Python 3.5 or higher
* pip version 9.0.1 or higher

If necessary, upgrade your version of pip:

$ python -m pip install --upgrade pip

If you cannot upgrade pip due to a system-owned installation, you can run the example in a virtualenv:

$ python -m pip install virtualenv

$ virtualenv venv

$ source venv/bin/activate

$ python -m pip install --upgrade pip

**gRPC**

Install gRPC:

$ python -m pip install grpcio

Or, to install it system wide:

$ sudo python -m pip install grpcio

gRPC tools

Python’s gRPC tools include the protocol buffer compiler protoc and the special plugin for generating server and client code from .proto service definitions. For the first part of our quick-start example, already generated the server and client stubs from [helloworld.proto](https://github.com/grpc/grpc/tree/v1.50.0/examples/protos/helloworld.proto" \t "_blank), but you’ll need the tools for the rest of our quick start, as well as later tutorials and your own projects.

To install gRPC tools, run:

$ python -m pip install grpcio-tools

Download the example

You’ll need a local copy of the example code to work through this quick start. Download the example code from our GitHub repository (the following command clones the entire repository, but you just need the examples for this quick start and other tutorials):

*# Clone the repository to get the example code:*

$ git clone -b v1.50.0 --depth 1 --shallow-submodules https://github.com/grpc/grpc

*# Navigate to the "hello, world" Python example:*

$ cd grpc/examples/python/helloworld

Run a gRPC application

From the examples/python/helloworld directory:

1. Run the server:
2. $ python greeter\_server.py
3. From another terminal, run the client:
4. $ python greeter\_client.py

You’ve just run a client-server application with gRPC.

**Update the gRPC service**

Now let’s look at how to update the application with an extra method on the server for the client to call. Our gRPC service is defined using protocol buffers. You need to know is that both the server and the client “stub” have a SayHello RPC method that takes a HelloRequest parameter from the client and returns a HelloReply from the server, and that this method is defined like this:

*// The greeting service definition.*

**service** Greeter {

*// Sends a greeting*

**rpc** SayHello (HelloRequest) **returns** (HelloReply) {}

}

*// The request message containing the user's name.*

**message** **HelloRequest** {

**string** name = 1;

}

*// The response message containing the greetings*

**message** **HelloReply** {

**string** **message** = 1;

}

Let’s update this so that the Greeter service has two methods. Edit examples/protos/helloworld.proto and update it with a new SayHelloAgain method,

with the same request and response types:

*// The greeting service definition.*

**service** Greeter {

*// Sends a greeting*

**rpc** SayHello (HelloRequest) **returns** (HelloReply) {}

*// Sends another greeting*

**rpc** SayHelloAgain (HelloRequest) **returns** (HelloReply) {}

}

*// The request message containing the user's name.*

**message** **HelloRequest** {

**string** name = 1;

}

*// The response message containing the greetings*

**message** **HelloReply** {

**string** **message** = 1;

}

Remember to save the file!

Generate gRPC code

Next we need to update the gRPC code used by our application to use the new service definition.

From the examples/python/helloworld directory, run:

$ python -m grpc\_tools.protoc -I../../protos --python\_out=. --pyi\_out=. --grpc\_python\_out=. ../../protos/helloworld.proto

This regenerates helloworld\_pb2.py which contains our generated request and response classes and helloworld\_pb2\_grpc.py which contains our generated client and server classes.

Update and run the application

We now have new generated server and client code, but we still need to implement and call the new method in the human-written parts of our example application.

**Update the server**

In the same directory, open greeter\_server.py. Implement the new method like this:

**class** **Greeter**(helloworld\_pb2\_grpc.GreeterServicer):

**def** SayHello(self, request, context):

**return** helloworld\_pb2.HelloReply(message='Hello, %s!' % request.name)

**def** SayHelloAgain(self, request, context):

**return** helloworld\_pb2.HelloReply(message='Hello again, %s!' % request.name)

...

Update the client

In the same directory, open greeter\_client.py. Call the new method like this:

**def** run():

**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)

response = stub.SayHelloAgain(helloworld\_pb2.HelloRequest(name='you'))

print("Greeter client received: " + response.message)

Run!

Just like we did before, from the examples/python/helloworld directory:

1. Run the server:

$ python greeter\_server.py

From another terminal, run the client:

$ python greeter\_client.py