Exercise 10

gRPC / Binary protocols

Pre-reqs
None

Objectives
Demonstrate the use of a fast binary protocol
Demonstrate a python client calling a Java service via ProtoBuf messages

Steps

1. Firstly, we need to checkout the code from Github.
   In a new terminal window, type the following:

   ```
   cd ~
   git clone https://github.com/pzfreo/grpc-sample.git
   cd grpc-sample
   ```

2. There is a Java server project ready to go.
   Firstly build it:

   ```
   cd java
   gradle build shadowJar
   ```

3. Open the Java project in vscode and take a look.

4. Have a look at `src/main/proto/purchase.proto`
   This contains the gRPC service definition in the ProtoBuf format.

   It looks like this:

```
syntax = "proto3";

package freo.me.purchase;

service Purchase {

  rpc purchase (PurchaseRequest) returns (PurchaseReply) {}
}

message PurchaseRequest {
  string poNumber = 1;
  string lineItem = 2;
  int32 quantity = 3;
  Date date = 4;
  string customerNumber= 5;
  string paymentReference = 6;

}

message Date {
       int32 year = 1;
       int32 month = 2;
       int32 day = 3;
}

message PurchaseReply {
  string message = 1;
  int32 returncode = 2;
}
```

5. Now look at the PurchaseServer.java

   This will only work when the .proto file has been compiled into Java. That happens as part of the gradle build. Most of this file is "boilerplate". The real logic happens near the bottom where it says:

```
public void
       purchase(PurchaseRequest req,
              StreamObserver<PurchaseReply> responseObserver) {

      System.out.println("Customer Number: " +req.getCustomerNumber());
      System.out.println("PO Number: " +req.getPoNumber());
      PurchaseReply reply = PurchaseReply.newBuilder().
              setMessage("order accepted").setReturncode(0).build();
      responseObserver.onNext(reply);
      responseObserver.onCompleted();
    }
```

   This code handles any requests against the purchase method (defined as "rpc purchase" in the proto file). It prints out some basic information and then responds with "order accepted" and a returncode of 0.

6. You can run this code as a JAR just like before:

```
java –jar build/libs/PurchaseGRPC–all.jar
```

Now lets create a Python client. In a new terminal window:

```
cd ~/grpc–sample
mkdir python
cd python
cp ../java/src/main/proto/purchase.proto .
```

7. We need to install the python grpc tools:

```
sudo pip install grpcio
sudo pip install grpcio–tools
```

8. Now we can run the client code compiler (all on one line please)

```
python –m grpc_tools.protoc –I .
c
```

9. If you look at the directory (ls), you will see two new files:

```
> ls –l
total 20
–rw–rw–r–– 1 oxsoa oxsoa 1433 Jan 10 19:24 purchase_pb2_grpc.py
–rw–rw–r–– 1 oxsoa oxsoa 8269 Jan 10 19:24 purchase_pb2.py
–rw–rw–r–– 1 oxsoa oxsoa  446 Jan 10 19:21 purchase.proto
```

10. These files will let us call the purchase service. Create and edit a new python program:

```
code purchase–client.py
```

11. Type in the following code:

```
import grpc;
import purchase_pb2;
import purchase_pb2_grpc;
import time;

def run():
  channel = grpc.insecure_channel('localhost:50051')
  stub = purchase_pb2_grpc.PurchaseStub(channel)
  print ("starting")

  response = stub.purchase(purchase_pb2.PurchaseRequest(poNumber="001",quantity=5))
  print("Purchase client received: " + response.message)
  print("Purchase client received: " + str(response.returncode))

run()
```

12. Run the program:

```
python purchase-client.py
```

13. Check the console log on your Java server and you should see it has been called.

Extension

14. Change the python code to call the server 1000 times. Add a timer around the code: e.g.

```
import time

t0 = time.time()
// code
t1 = time.time()

t = t1-t0
print "elapsed seconds: " + str(t)
```

15. See how long it takes. Note that if you run it more than once the Java server may "warm up" and do some JIT compilation, so the numbers should improve and then stabilize.

16. You could also remove all the console logging in the Java and Python code and try again. What is the throughput of requests once the system is warm?

Extension

The performance of the python client <-> Java server seems a little slow. Use another language to create a client and see if it is python that is the problem.