SOFT8023

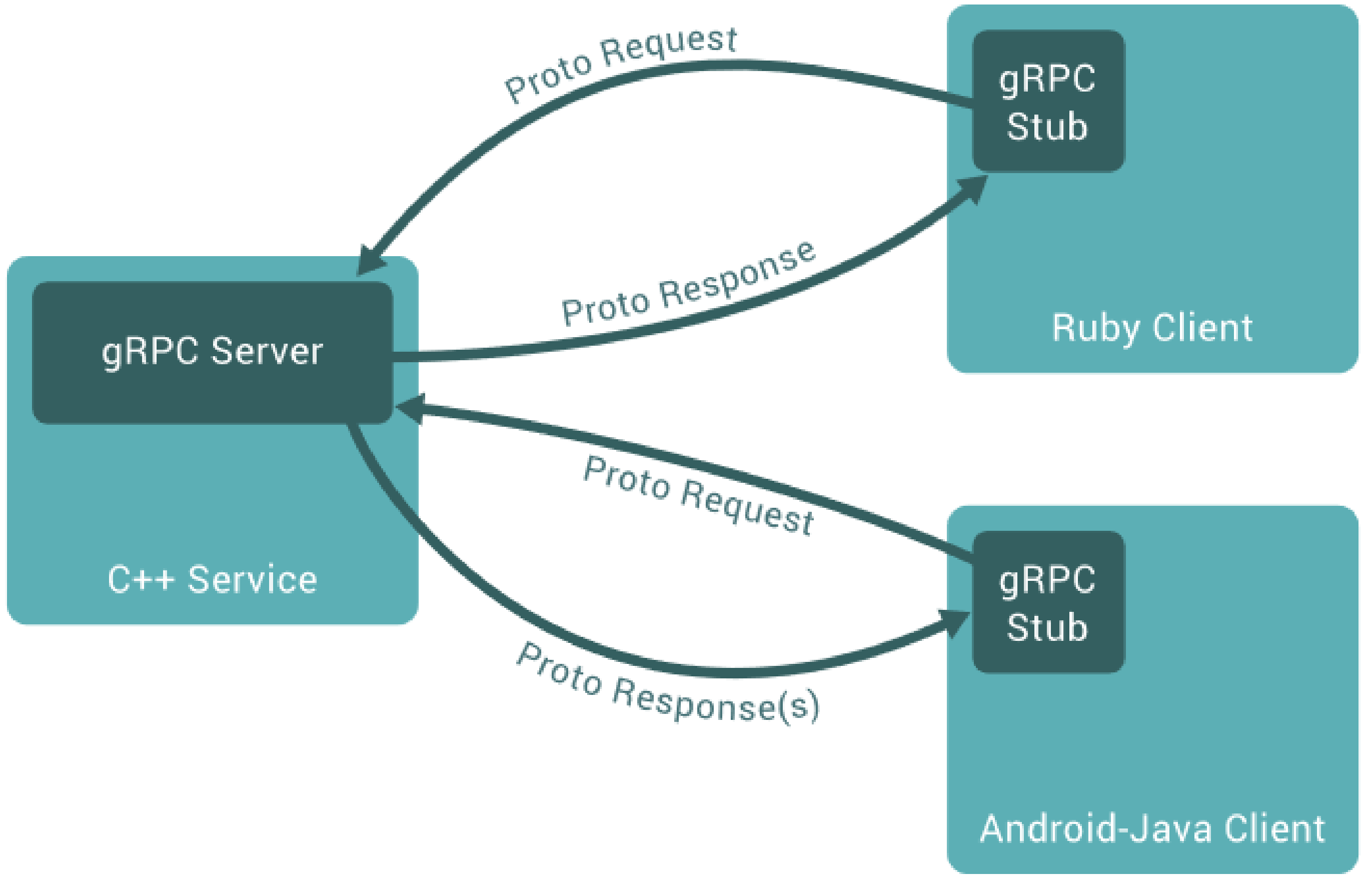
Protocol Buffers

gRPC

- Provides remote procedure calls (RPC) with data serialization
- Is agnostic and can work with Python, Go, C++, Node, PHP and more
- Uses HTTP/2 (modern network transfer protocol with streaming support) – faster than REST with HTTP/1.1
  - Bi-directional communication supported, not just request-response
- Support authentication mechanisms, such as SSL/TLS
- See https://www.infoq.com/presentations/api-pb-grpc for a presentation about an advanced case study
- Alternatives to gRPC: Apache Hadoop's Avro, Apache Thrift, Twitter's Finagle (JVM-based)

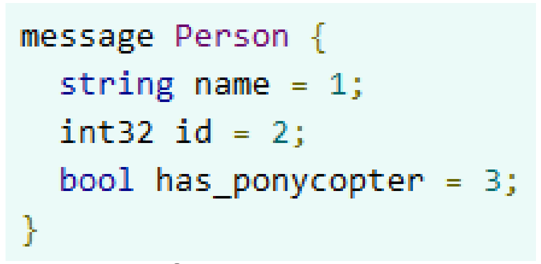
gRPC

- Service is defined, including methods that can be called and what parameters/types to use
- Server side runs gRPC service
- Client side has stub matching service provided by service



Protocol buffers

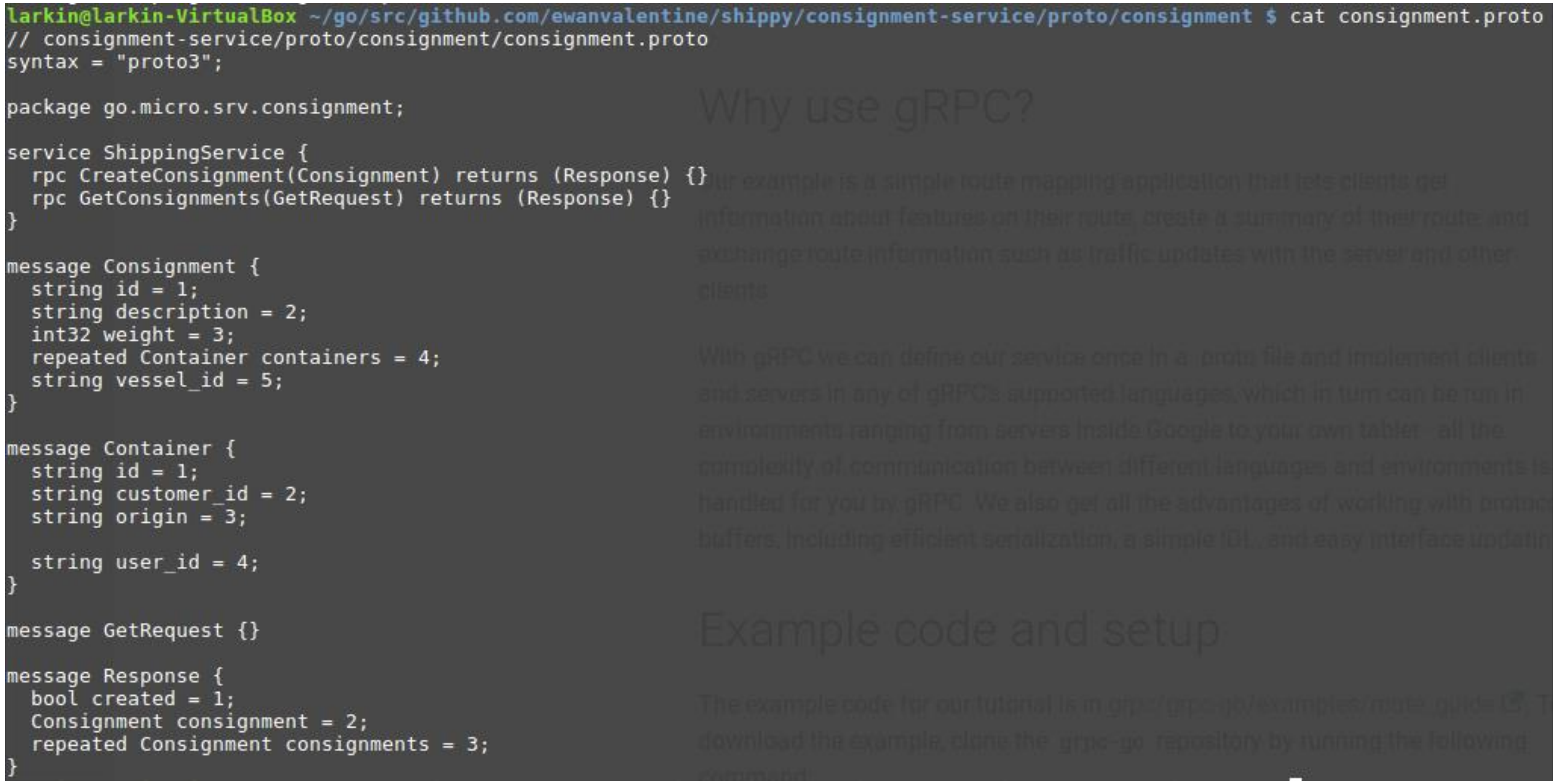
- Or *Protobufs*
- gRPC usually uses protocol buffers (something open-sourced by Google) to serialize structured data
- Message payloads are binary, so compact
- You create a proto file (using an IDL – interface definition language) that defines the service with *messages* containing *fields*, e.g. person.proto



```
message Person {
  string name = 1;
  int32 id = 2;
  bool has_ponycopter = 3;
}
```

- Then use the *protoc* compiler to compile the proto file for the language you are using, e.g. compile to a .pb.go or a pb.py file

Example



```
larkin@larkin-VirtualBox ~/go/src/github.com/ewanvalentine/shippy/consignment-service/proto/consignment $ cat consignment.proto
// consignment-service/proto/consignment/consignment.proto
syntax = "proto3";

package go.micro.srv.consignment;

service ShippingService {
  rpc CreateConsignment(Consignment) returns (Response) {}
  rpc GetConsignments(GetRequest) returns (Response) {}
}

message Consignment {
  string id = 1;
  string description = 2;
  int32 weight = 3;
  repeated Container containers = 4;
  string vessel_id = 5;
}

message Container {
  string id = 1;
  string customer_id = 2;
  string origin = 3;

  string user_id = 4;
}

message GetRequest {}

message Response {
  bool created = 1;
  Consignment consignment = 2;
  repeated Consignment consignments = 3;
}
```

Why use gRPC?

Our example is a simple route mapping application that lets clients get information about features on their route, create a summary of their route, and exchange route information such as traffic updates with the server and other clients.

With gRPC we can define our service once in a .proto file and implement clients and servers in any of gRPC's supported languages, which in turn can be run in environments ranging from servers inside Google to your own tablet - all the complexity of communication between different languages and environments is handled for you by gRPC. We also get all the advantages of working with protoc buffers, including efficient serialization, a simple IDL, and easy interface updatin

Example code and setup

The example code for our tutorial is in grpc/grpc-go/examples/route_guide. T download the example, clone the grpc-go repository by running the following command:

pb.go file

- protoc will compile a protocol buffer depending on the language specific, e.g. protoc -I --go_out=plugins=gprc:<insert path to proto file>
- Or with Python:

```
$ python -m grpc_tools.protoc -I../../protos --python_out=. --grpc_python_out=.
../../protos/helloworld.proto
```

- See more here:
  - https://grpc.io/docs/guides/#overview (gRPC)
  - https://developers.google.com/protocol-buffers/docs/overview (Protocol Buffers)
  - https://developers.google.com/protocol-buffers/docs/proto3 (Proto3 language guide)