gRPC retry mechanisms with Go

Thomas Gosteli @ Bärner Go Talks 2023 no. 3


FIRST EVER TALK ON A MEETUP
NOT NERVOUS AT ALL
imgflip.com

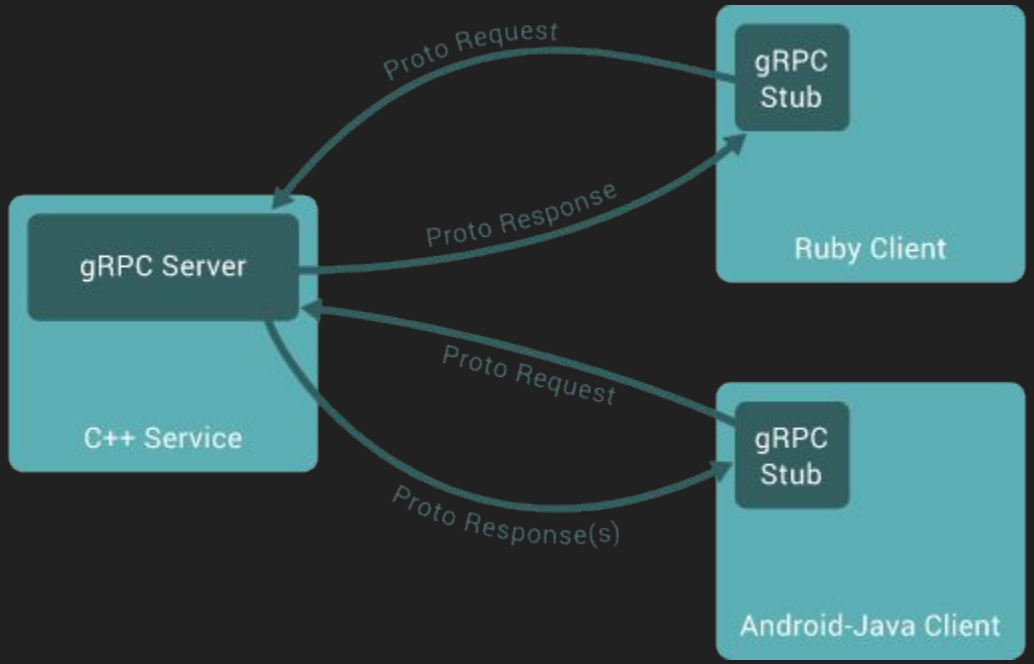
Intro

- Thomas Gosteli, 30 years young
- Backend developer @ swissmarketplace.group working on the backend of tutti.ch which is mostly written in Go (and some C 😬)
  - We use gRPC for internal communication and in some places make use of the automatic retry capabilities I'm going to present
- You can find me on some ("old school") social media and GitHub with handle @ghouscht


LET'S GET STARTED
imgflip.com

very quick intro to gRPC

A high performance, open source universal RPC framework - grpc.io



https://grpc.io/docs/what-is-grpc/introduction/#overview

gRPC: unary RPCs

- Pretty similar to a REST or even SOAP (💩) API, gRPC unary RPC is a request-response model. We send a request and get a response back - that’s it. This is (likely) the most common type of RPC.

```
service UsersService {
    rpc GetUser (GetUserRequest) returns (GetUserResponse) {}
}

message GetUserRequest {
    int64 id = 1;
}

message GetUserResponse {
    User user = 1;
}

message User {
    int64 id = 1;
    string first_name = 2;
    string last_name = 3;
    google.protobuf.Timestamp birthdate = 4;
}
```

gRPC: stream RPCs

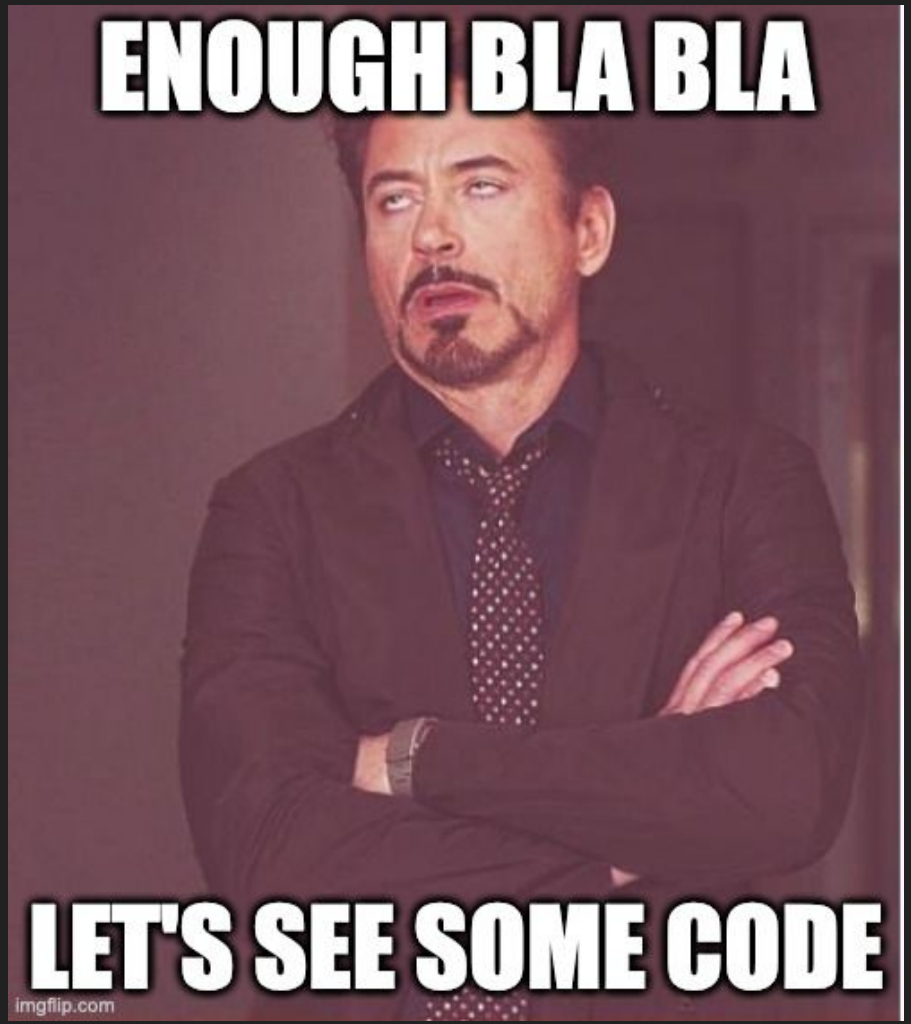
- **server streaming RPC**
  - Similar to a unary RPC, except that the server returns a stream of messages in response to a client's request.
- **client streaming RPC**
  - The client sends a stream of messages and the server responds with a single message once the client sent everything.
- **bidirectional streaming RPC**
  - Client and server can read/write messages in any order so the implementation is application specific.

```
service UsersService {
    rpc GetAllUsers (GetAllUsersRequest) returns (stream GetAllUsersResponse) {}
}

message GetAllUsersRequest {
    int64 offset = 1;
}

message GetAllUsersResponse {
    User user = 1;
}

message User {
    int64 id = 1;
    string first_name = 2;
    string last_name = 3;
    google.protobuf.Timestamp birthdate = 4;
}
```


ENOUGH BLA BLA
LET'S SEE SOME CODE
imgflip.com

https://github.com/ghouscht/gRPC-retry-mechanisms-with-go

Summary

- It is very easy to retry unary RPCs - no code change needed
- Retrying/restarting streaming RPCs is more complex and can't be done automatically
- The configuration (of service configs) is a bit fragile (in my opinion) - unit testing can help

- Go developers are used to implement things on their own and we do it every day - and I think this is a good thing!
- But sometimes we can and should rely on battle proved solutions - like the automatic retires we have at hand from gRPC
- Apparently lots of people (including myself until a few weeks ago) are not aware that such a feature exists in gRPC so this is why I decided to talk about it.

… gRPC can do more, a lot more 😱

- loadbalancing
- reflection
- circuit breaking
- authentication
- compression
- deadlines
- request hedging
- …

Thank you