GRPC

A Brief Introduction

What is GRPC?

- Stands for **g**RPC **R**emote **P**rocedure **C**alls

- Allows you to define a service interface using an IDL
  - Data types - Input / Output types
  - Operations (methods/function signatures)
  - Uses v3 .proto files

- Provides tooling to generate (compile) language specific data types and client/server implementations

- You just implement server-side function bodies (business logic)
  - In Java terminology the generated server is an abstract class you extend

- Currently supports:
  - 10 different languages - including Java, Scala, PHP, Node (JavaScript / TypeScript)
  - Web browser support currently in Beta - general availability expected end of Oct 2018

Who is using it?

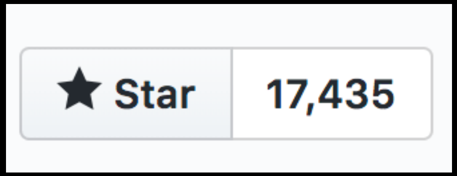
- Originally created inside google, and later open sourced

- Companies using it you will have heard of:
  - Google (obviously)
  - Square
  - Netflix
  - Docker
  - Cisco
  - Juniper Networks

- Google are using it to make *billions* of calls per second
  - Therefore: GRPC == battle tested

- Adopted by the Cloud Native Computing Foundation (CNCF)

- Has over 17,000 stars on GitHub
  - AKKA has ~9,000,



★ Star 17,435

How does it work?

1. First define your service and messages in a .proto
    - This is a Google protobuf (v3) file

2. Generate messages, client and server using build tool (e.g. sbt, mvn)
    - Build tool wraps the “protoc” compiler

3. Extend generated server to implement service functions

4. Instantiate server (create an instance)

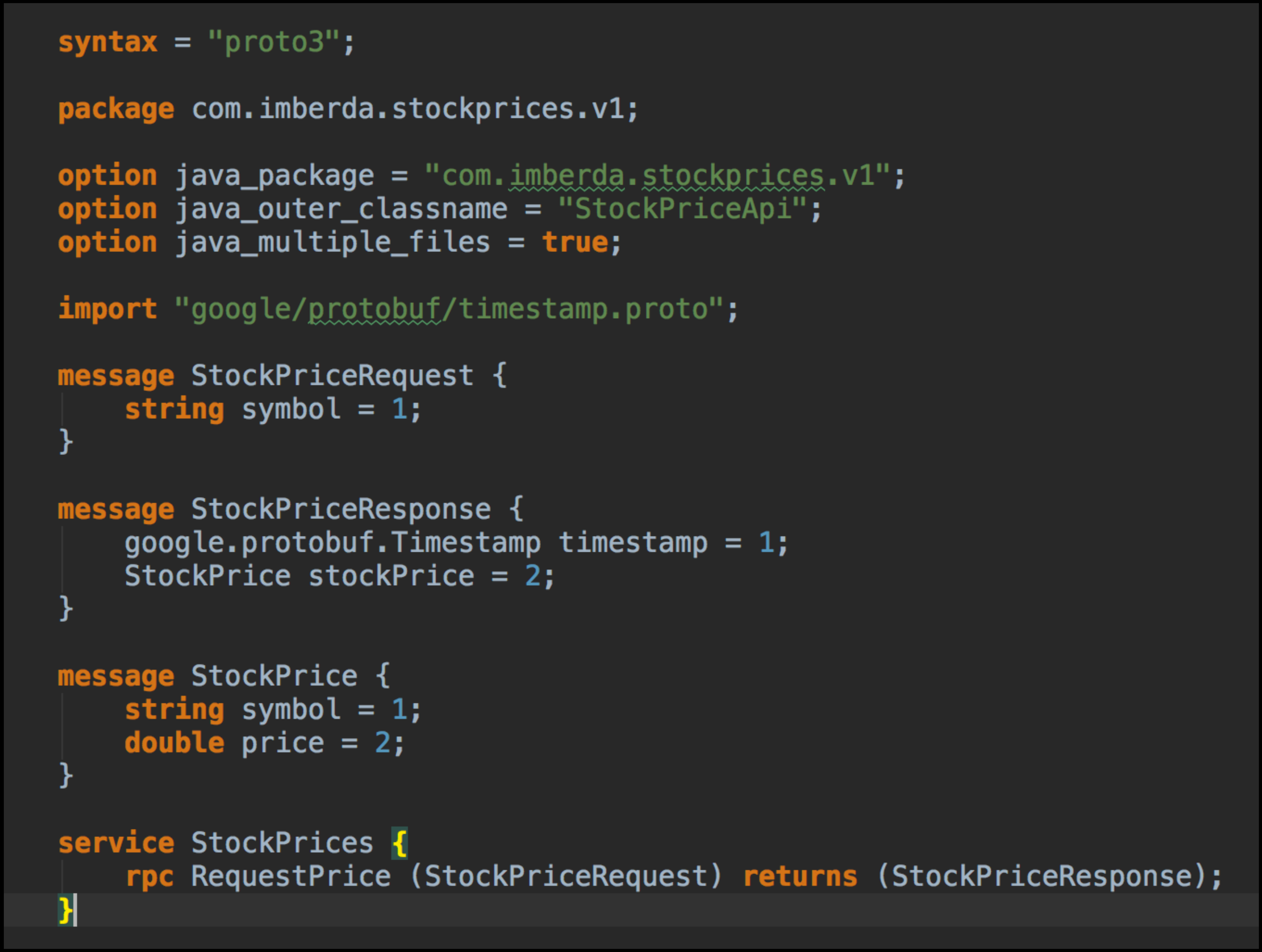
5. Instantiate client (create an instance)

6. Use client to call the server

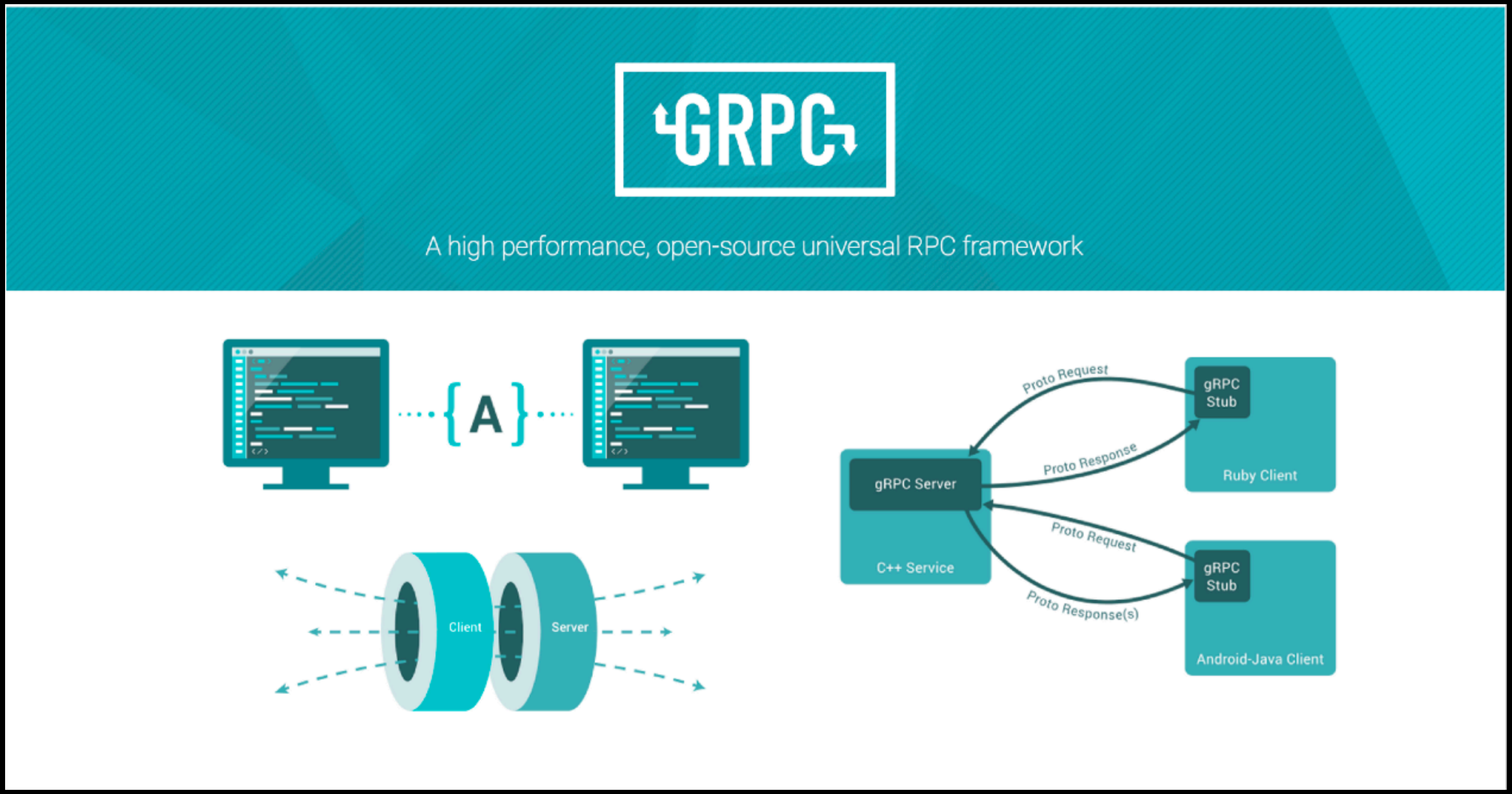
Note: You *can* use the generated (protobuf) messages without GRPC

- i.e. if you want to serialise them to JSON or binary to send or store somewhere (file/DB)

Example Service Definition



```
syntax = "proto3";

package com.imberda.stockprices.v1;

option java_package = "com.imberda.stockprices.v1";
option java_outer_classname = "StockPriceApi";
option java_multiple_files = true;

import "google/protobuf/timestamp.proto";

message StockPriceRequest {
    string symbol = 1;
}

message StockPriceResponse {
    google.protobuf.Timestamp timestamp = 1;
    StockPrice stockPrice = 2;
}

message StockPrice {
    string symbol = 1;
    double price = 2;
}

service StockPrices {
    rpc RequestPrice (StockPriceRequest) returns (StockPriceResponse);
}
```



A high performance, open-source universal RPC framework

Very Quick Demo…

Performance

- Compared to REST-JSON - benchmarks often report ++
  - More than x5 ops/sec
  - Reduction in CPU of over 25%

- How?
  - Uses HTTP/2.0
    - Compressed binary headers
    - Multiplexing of requests within single connection
  - Uses long-live connections between client and server
  - Uses binary compact serialisation format (avoid string lexing/parsing)

++ Warning: Your Milage May Vary

- “*If you torture the data long enough, it will confess to anything*”
- Run your own tests using reflecting your own use cases

REST is Best!

"*A presumptive architecture is a software architecture that is dominant in a particular domain. Rather than justifying their choice to use it, developers in that domain, may have to justify a choice that differs from their presumptive architecture. Non-curious developers may not even seriously consider other architectures, or have the apprehension that all software should conform to the presumptive architecture.*”

(George Fairbanks)

- Some of the great things about REST are that its ubiquitous:
  - Supported by all languages
  - Excellent tooling (practically no tooling required)
  - Simple to understand
  - We’ve been doing it for years and understand its dark corners

- However….
  - Lots of boilerplate when writing a REST service
  - Typically clients for *n* languages need *n* implementations
  - The debate around what is “RESTful” is seemingly endless (😡)
  - Need to be creative to handle large amounts of data (pagination, ndjson)
  - Need to assemble your stack (http client, web framework, marshaller, etc.)

Where to use GRPC?

- REST makes total sense to clients we don’t own on the Internet (browsers)

- Does it make sense to use REST between services in the same racks of our data centre, where we can control everything?

- Makes sense to use GRPC service-to-service where:
  - You need (or potentially need) to handle very high load
  - You want client, server or client-and-server streaming
  - Environments where you want to generate clients for *n* languages
  - You want or need well-defined interfaces
  - You want or need well-understood & documented compatibility rules

Side Effects

- Business logic doesn't easily leak into bloated client libraries
  - They're generated so you get what you get (give or take config and interceptors)
- You have DTOs that can't be anything other than DTOs
  - Means you don't mix-up your domain model with your interface API model
- Another option when integration testing
  - If you are testing Service A which depends on Service B
  - You can easily instantiate Server B as GRPC server - just create a subclass
  - Provide mock responses
  - Very realistic as you use the same serialisation and wire protocol handling
- Sane semantics around error handling
  - Just return an exception with the appropriate content
- Different version / location for interface vs. implementation?
  - Your server interface (.proto) can be in a separate module / project / repository
  - Possibly separately versioned

Final Thoughts

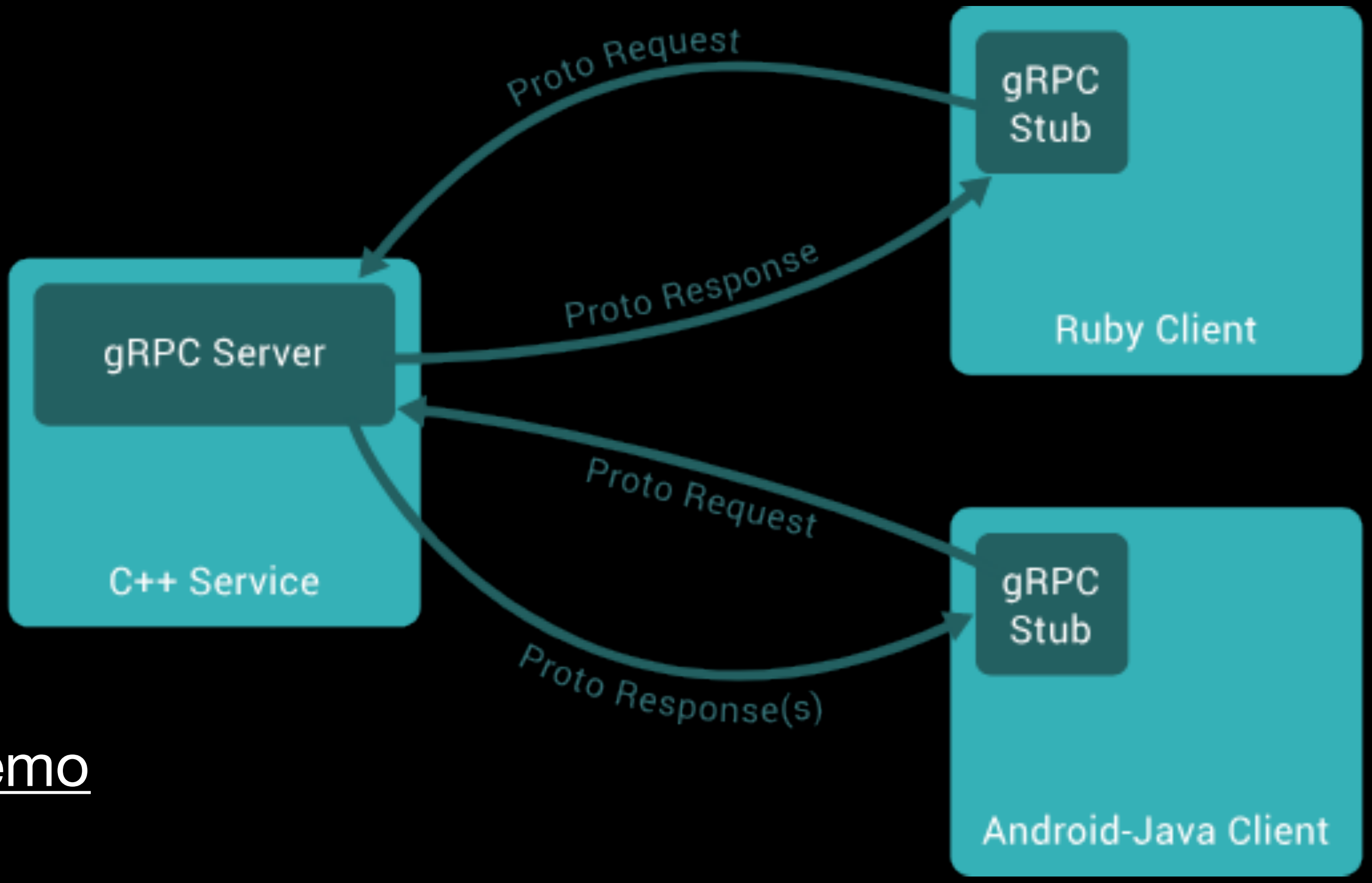
- There is a Scala protoc compiler (scalapb) that generates nice Scala (case) classes. We already use it for protobuf.

- NGINX recently (Q1 2018) added support for reverse proxying GRPC (including load-balancing)

- Istio (service mesh) support GRPC in addition to HTTP
  - Very interesting in a post Kubernetes migrated world

- There are CURL and (poor man’s) Postman type tools available

Want to Know more?



Code used in demo:

- https://github.com/imberda/grpc-demo

Links:

- https://grpc.io/
- https://github.com/grpc-ecosystem/awesome-grpc
- https://www.nginx.com/blog/nginx-1-13-10-grpc/
- https://github.com/fullstorydev/grpcurl
- https://scalapb.github.io/grpc.html