gRPC: An alternative to the REST?

GRPC

Pieter Louw

@pieterlouw

JSinSA - 15 July 2017

What are these automotive parts?

Introduction

- Both serve the same purpose
- Fuel Injection was adopted since 1980 because of strict emission regulations (US/Japan)
- Public API's today = REST / SOAP
- RESTful vs REST*ish*
- API playing field today is different and not only for browsers
- Journey with gRPC - a viable for public APIs?

gRPC

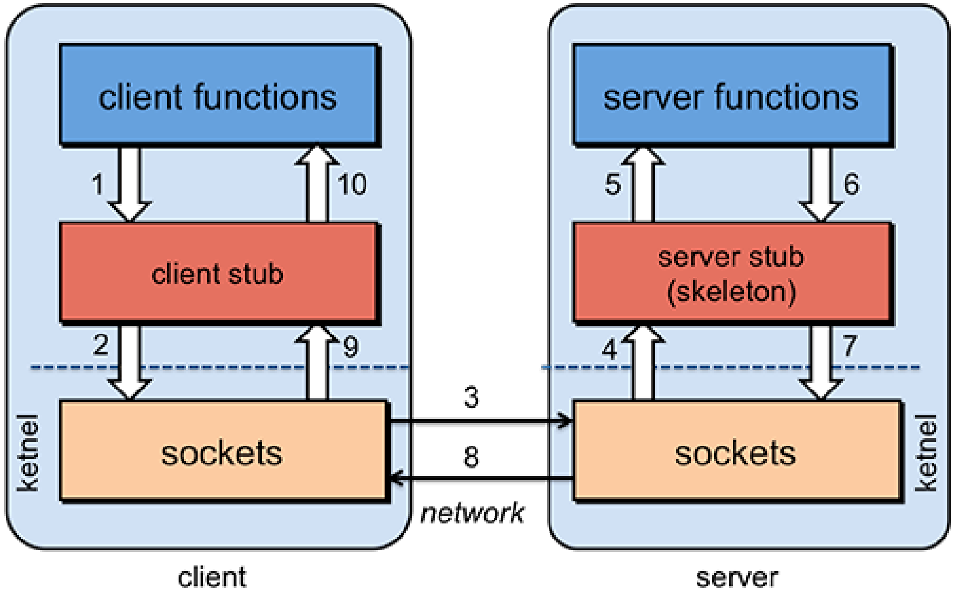
- It stands for "gRPC Remote Procedure Calls"
- Official definition from **grpc.io**: *"A high performance, open-source universal RPC framework"*
- Less formal definition: *"A really cool way to do inter process communication"* - Francesc Campoy (Señor Developer Advocate for Google Cloud)
- Based on Google's internal RPC framework called Stubby
- Open Sourced and v1.0.0 released in August 2016 (current release v1.4)

Trivia:

- 1.0 'g' stands for 'gRPC'
- 1.1 'g' stands for 'good'
- 1.2 'g' stands for 'green'
- 1.3 'g' stands for 'gentle'
- 1.4 'g' stands for 'gregarious'

What is RPC?

Remote Procedure Call (RPC) is an architectural pattern where you call a method in another address space, usually over a network, but it looks like your calling a method inside your local address space.



- Client and server `stubs` need to be created for serialing/unserializing
- Stubs also handle transport over the wire
- Interface Definition Language (IDL)

gRPC - What makes gRPC so effective?

Protocol Buffers:

- Protocol buffers is the encoding format, but is pluggable
- Binary encoded
- Smaller in size and parsed faster than text encoded formats

HTTP/2:

- Faster than HTTP/1.x (Demo -> https://http2.golang.org/)
- Multiplexing (single tcp connection),
- Bidirectional streaming (up and down at the same time and faster than long-polling),
- Smaller packets (Binary format & Header compression)

IDL:

- `.proto` file
- Describes how to serialize the messages to send over wire
- Service definitions are simple, small and concise

gRPC - Steps

Steps to create a gRPC service:

1. Define messages and method calls in a `.proto` file
2. Generate stubs from `.proto` definition (`protoc`)
3. Implement code (server)
4. Write client code

gRPC - Example

gRPC - Example

gRPC - Example (.proto definition)

```
syntax = "proto3";

package cowbell;

// The Cowbell service definition.
service CowbellService {
    // adds a cowbell
     rpc MoreCowbell (MoreCowbellRequest) returns (CowbellResponse) {}
}

// A request message containing the number of cowbells to add
message MoreCowbellRequest {
  int32 qty = 1;
}

// A response message containing the number of cowbells added in total
message CowbellResponse {
  int32 total = 1;
}
```

gRPC - Example (Implementing the service)

```
var PROTO_PATH = __dirname + '/../service/service.proto';

// import grpc library
var grpc = require('grpc');
// load stub
var cowbell = grpc.load(PROTO_PATH).cowbell;

var total = 0;

/**
 * Implements the MoreCowbell RPC method.
 */
function myMoreCowbell(call, callback) {
    var cowbellResponse;

    console.log("moreCowbell qty="+ call.request.qty);

    total += call.request.qty;

    cowbellResponse = {
        total: total
    };

    callback(null, cowbellResponse);
}
```

gRPC - Example (Implementing the service)

```
/**
 * Starts an RPC server that receives requests for the Cowbell service at the
 * sample server port
 */
function main() {
  var server = new grpc.Server();

   server.addService(cowbell.CowbellService.service, {
     moreCowbell: myMoreCowbell,
   });

  server.bind('0.0.0.0:9090', grpc.ServerCredentials.createInsecure());
  server.start();

  console.log("gRPC Server started on 0.0.0.0:9090")
}

main();
```

gRPC - Example (Calling the service)

```
var PROTO_PATH = __dirname + '/../service/service.proto';

var grpc = require('grpc');
var cowbell = grpc.load(PROTO_PATH).cowbell;

function main() {
  var client = new cowbell.CowbellService('localhost:9090',
                                          grpc.credentials.createInsecure());

   client.moreCowbell({qty: 1}, function(err, response) {
     console.log('Total Cowbells:', response.total);
   });
}

main();
```

gRPC - Data and method types

Many different types to use:

- Scalar (int64, bool, bytes)
- Container (repeated, maps)
- Other (enum, oneOf)

Different method types:

- Unary (normal request/response)
- Server side streaming
- Client side streaming
- Bidirectional streaming

Example - Product service

Product domain object

```
  // Our product domain object
  message Product {
      int64 product_id = 1; //primary key
      string code = 2; // unique code
      int64 price = 3;
      string name = 4;
      string description = 5;
      ProductStatus status = 6;
      repeated string tags = 7;
      map<string, string> additionalFields = 8;
  }

// Different statuses a product can have
enum ProductStatus {
    UNKNOWN = 0; //default
    AVAILABLE = 1;
    ONHOLD = 2;
    DISCONTINUED = 3;
    FUTURE = 4;
}
```

Example - Product service

Product service methods

```
// Product service definition.
service ProductService {
    // ------- CRUD operations
    rpc AddProduct (Product) returns (AddProductResponse) {}
    rpc UpdateProductById (UpdateProductByIdRequest) returns (Product) {}
    rpc UpdateProductByCode (UpdateProductByCodeRequest) returns (Product) {}
    rpc GetProduct (GetProductRequest) returns (Product) {}

    // ------- Streaming
     rpc SearchProducts(SearchProductsRequest) returns (stream Product) {} // serv
     rpc UploadProducts (stream Product) returns (stream AddProductResponse) {} //

    // --------- Actions
    rpc BuyProduct(BuyProductRequest) returns (BuyProductResponse) {}
    rpc RequestProduct(RequestProductRequest) returns (RequestProductResponse) {}
}
```

Example - Product service

oneOf type:

```
message SearchProductsRequest {
    oneof searchByFilter {
        ProductStatus status = 1;
        string tag = 2;
        int64 priceLessThan = 3;
        int64 priceGreaterThan = 4;
    }
}
```

Example of search request:

https://github.com/googleapis/googleapis/blob/master/google/datastore/v1/query.proto

Languages supported by gRPC

- 10 languages are supported
- *(Haskell and Swift as well, but only experimental at this stage)*

Client only

- Android Java
- Objective-C
- PHP

Server and client

- C++
- Python
- Ruby
- C#
- **Node.js**
- Java
- Go

grpc-Web

- 10 languages are a lot, but what about javascript in the browser?
- Browser limitations prohibit full implementation of gRPC

grpc-Web - There's hope!

grpc-Web by Improbable

- Improbable migrated from REST to gRPC
- Created while migrated from REST to gRPC created their own grpc-Web implementation as their webapps could not utilise gRPC (https://github.com/grpc/grpc/blob/master/doc/PROTOCOL-WEB.md)
- Limited streaming support (only server-side streaming)
- `ts-protoc-gen`: a TypeScript plugin for the protocol buffers compiler (protoc) that produces TypeScript service definitions and TypeScript declarations for the standard JavaScript objects generated by upstream `protoc`.
- `grpc-web-client` – a TypeScript gRPC-Web client library for browsers that abstracts away the networking (Fetch API or XHR) from users and codegenerated classes.
- Proxy software (written in Go) that do the protocol translation and reverse-proxying.

https://improbable.io/games/blog/grpc-web-moving-past-restjson-towards-type-safe-web-apis

grpc-Web Example (.proto Definition)

```
syntax = "proto3";

message Book {
  int64 isbn = 1;
  string title = 2;
  string author = 3;
}

message GetBookRequest {
  int64 isbn = 1;
}

message QueryBooksRequest {
  string author_prefix = 1;
}

service BookService {
  rpc GetBook(GetBookRequest) returns (Book) {}
  rpc QueryBooks(QueryBooksRequest) returns (stream Book) {}
}
```

grpc-Web Example (Typescript client)

```
import {grpc, BrowserHeaders} from "grpc-web-client";

// Import code-generated data structures.
import {BookService} from "proto/book_service_pb_service";
import {QueryBooksRequest, Book, GetBookRequest} from "proto/book_service_pb";

const queryBooksRequest = new QueryBooksRequest();
queryBooksRequest.setAuthorPrefix("Geor");

grpc.invoke(BookService.QueryBooks, {
  request: queryBooksRequest,
  host: "https://grpc.api.com",

  onMessage: (message: Book) => {
    console.log("got book: ", message.toObject());
  },

  onEnd: (code: grpc.Code, msg: string | undefined, trailers: BrowserHeaders) => {
    console.log(code, message)
  }
});
```

grpc-Web Deployment

- Needs gateway/proxy to translate grpc-Web to native grpc

Two options:

- Improbable's `grpcwebproxy` standalone application (written in Go)
- Caddy webserver `grpc` plugin (from 0.10.4 release)

Other gRPC features

- Canonical error codes
- Interceptors/Middleware (tracing, health-checking etc.)
- Deadlines/Timeouts
- Metadata (i.e authentication details)
- Channels (i.e switching on and off message compression)
- RPC termination
- Cancelling RPCs

Node.js gRPC frameworks

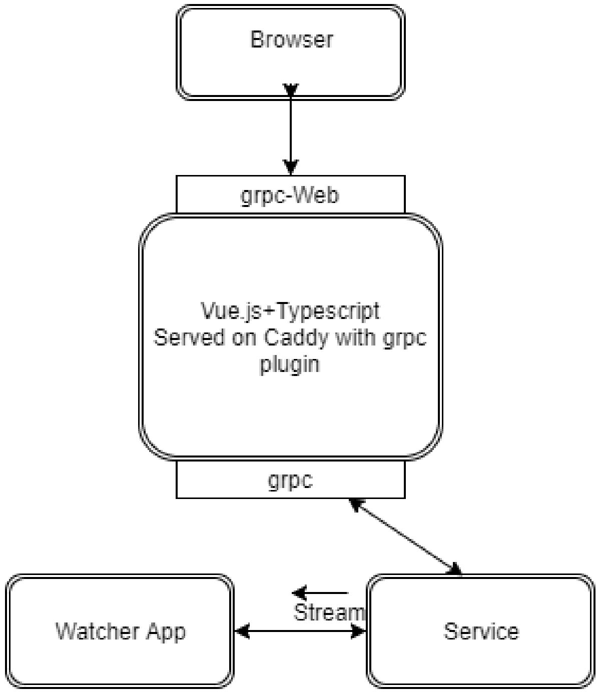
- **CondorJS** (http://condorjs.com): "Condor is a minimalist, fast framework for building GRPC services in Node JS. It’s like express for GRPC"
- **MaliJS** (https://malijs.github.io):"Minimalistic Node.js gRPC microservice framework"
- **grpc-dynamic-gateway** (https://github.com/konsumer/grpc-dynamic-gateway): "Like grpc-gateway, but written in node and dynamic.Provides a REST-like JSON interface for your gRPC protobuf interface"

gRPC Adoption

- Most companies that adopt gRPC to be used internally.
- Netflix, Square, Cisco, Juniper, Arista, Ciena, Yik Yak, Improbable, Vendasta, VSCO, Cockroachdb, CoreDNS, NATS.io, ngrok etc.
- Google Cloud expose APIs like Speech recognition, PubSub, Cloud Storage as gRPC
- Protobuf definitions: https://github.com/googleapis/googleapis
- gRPC has also been selected to be part of the Cloud Native Computing Foundation along with projects like Kubernetes, Prometheus, Linkerd etc.

Demo

Use Case: Car Service Booking



Demo (.proto definition)

```
syntax = "proto3";

package proto;

service CarServiceDepartment {
  rpc MakeBooking(Booking) returns (Empty) {}
  rpc Watch (Empty) returns (stream Booking) {}
}

message Booking {
  string reg = 1;
  int32 odo = 2;
  string name = 3;
}

message Empty {}
```

Demo (Service Implementation)

```
var events = require('events');
var bookStream = new events.EventEmitter();

var grpc = require('grpc');
var service = grpc.load('../proto/carservice.proto');
var server = new grpc.Server();

server.addService(service.proto.CarServiceDepartment.service, {
    makeBooking: function(call, callback) {
        var booking = call.request;
        console.log("New Booking received");
        bookStream.emit('new_booking', booking);
        callback(null, {});
    },
    watch: function(stream) {
        bookStream.on('new_booking', function(booking){
            stream.write(booking);
        });
    }
});

console.log("gRPC server started on 0.0.0.0:9090")
server.bind('0.0.0.0:9090', grpc.ServerCredentials.createInsecure());
server.start();
```

Demo (grpc-Web Typescript client using Vue.js)

```
import {grpc, BrowserHeaders} from "grpc-web-client";
import {CarServiceDepartment} from "./_proto/carservice_pb_service";
import {Booking, Empty} from "./_proto/carservice_pb";
import { Component } from 'vue-typed'
import * as Vue from 'vue'

const template = require('./app.jade')();

@Component({
    template
})
class App extends Vue {
    host: string = 'https://grpc.cardealer.com:8443'; // grpc endpoint
    regNo: string =  '';
    odoMeter: number = 0;
    customerName: string = '';

    mounted(){  }

    registerBooking(){
        // code on next slide
    }
}

new App().$mount('#app');
// Code based on https://github.com/b3ntly/vue-gRPC
```

Demo (grpc-Web Typescript client using Vue.js)

```
    registerBooking(){
        //validation
        if (!this.regNo || !this.customerName || this.odoMeter == 0){
            alert("Please populate all fields!");
            return;
        }

        const request = new Booking();
        request.setReg(this.regNo);
        request.setOdo(this.odoMeter);
        request.setName(this.customerName);

        grpc.invoke(CarServiceDepartment.MakeBooking, {
            host: this.host,
            request: request,

            onMessage: (empty: Empty) => {
                alert("Booking successful");
            },

            onEnd(code: grpc.Code, message: string, trailers: BrowserHeaders){
                console.log(code, message, trailers);
            }
        })
    }
```

Demo (grpc Node client - watcher app)

```javascript
var grpc = require('grpc');
var service = grpc.load('../proto/carservice.proto');

var client = new service.proto.CarServiceDepartment('0.0.0.0:9090',
                    grpc.credentials.createInsecure());

console.log("Waiting for new bookings...");
var call = client.watch({});
    call.on('data', function(booking) {
        console.log('Car Service booking made! '+ booking.getName());
        // do lookup to see who the sales person was
        // send notification
    });
```

Demo (Caddyfile)

```
localhost:7777 {
    root demo_vue
}

localhost:8443 {
    tls ./misc/localhost.crt ./misc/localhost.key
    log
    grpc localhost:9090 {
        backend_is_insecure
        backend_tls_noverify
    }
}
```

gRPC - an Alternative to the REST?

Developer Experience (DX):

```
- The API is integrated into your programming language (part of the IDE / Code completion)
- Concise and simple "single source of truth" contract (.proto file)
- Support for many languages - it's a polyglot world we live in
- As a client developer you can mock your server implementations
```

Secure, faster and smaller **by default**

```
- Smaller payloads (binary protobufs) = less data and CPU processing time
- Fast (HTTP/2)
```

gRPC - an Alternative to the REST?

More options

```
- Streaming calls
- Can add complex types to API (*not necessarily a good thing*)
- Mix resource driven (noun) and action driven (verb) calls
```

API integrity

```
- Backwards and forwards compatibility (Protobufs)
- Structured & Strongly typed - you know you're doing something
                        wrong before there's a network call made
- Can't deviate from gRPC
```

gRPC - an Alternative to the REST?

Trade-offs:

- Where to place the .proto file and what if it's updated?
- Generated code
- Comms are abstracted away
- Message are machine readable
- Tooling
- No standards or idiomatic ways (yet as it's still early days)
- Eco systems for different languages evolve seperately
- Documentation can be lacking
- Use the protobuf types or convert to native types?

Conclusion

- Even though our public APIs today consist mainly of REST and SOAP APIs I believe gRPC definitely has a place in the future of the API world.
- It can solve many problems that can't be solved by existing paradigms (performance / streaming needs)
- Next time your creating an API (or consuming one),step back and wonder if it's not RPC you really doing.

Thank You!

> This presentation is made using RemarkJS (https://remarkjs.com)