gRPC

boilerplate to high performance scalable APIs

Robert Kubis
Developer Advocate

Google Cloud Platform

About me

Robert Kubis
Developer Advocate
Google Cloud Platform
London, UK

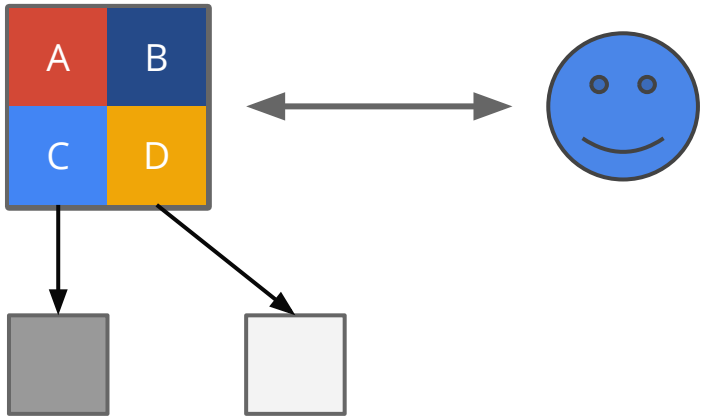
hostirosti
github.com/hostirosti

Agenda

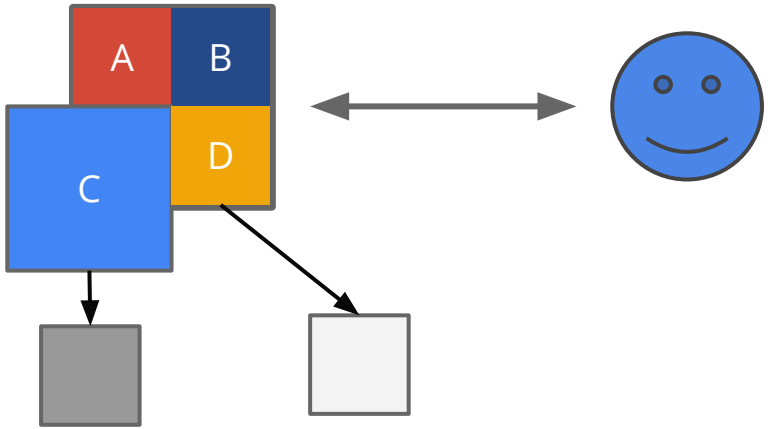
1. Motivation (Microservices)
2. HTTP/2
3. Protobuf
4. gRPC

Microservices

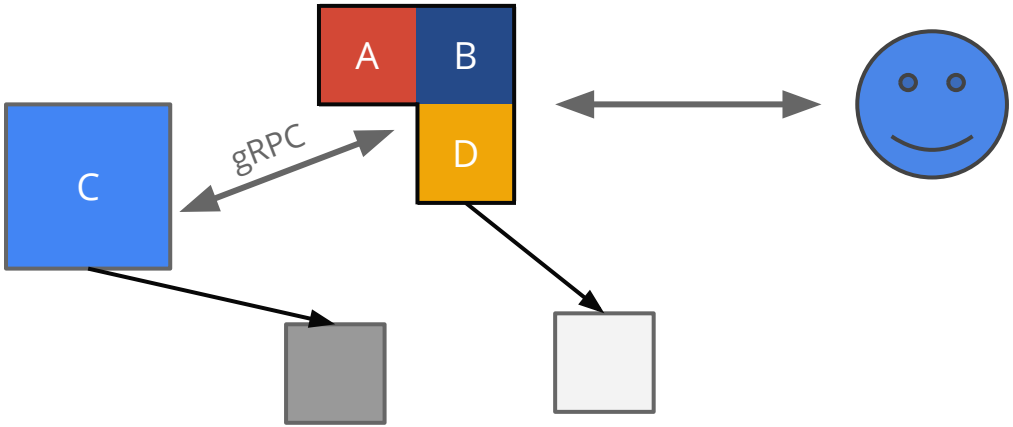
Decomposing Monolithic apps



Decomposing Monolithic apps



Decomposing Monolithic apps



HTTP/2

So what’s up with HTTP/1?

Today statistics show that there are on average:

- 12 distinct hosts per page
- 78 distinct requests per page
- 1,232 KB transferred per page

Resulting in typical render times of **2.6-5.6 seconds** (50th and 90th percentiles).

* Numbers are medians, based on latest HTTP Archive crawl data.

Coping Strategies

HTTP Pipelining

Queue responses not requests

Domain Sharding

Need more than 6 connections per origin?
Add more origins!

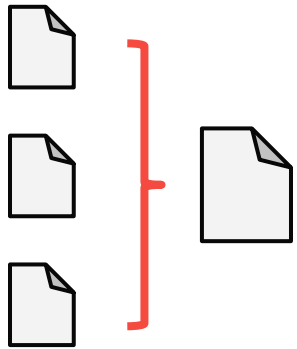
Concatenation and Spriting

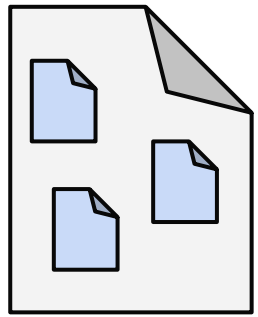
Multiple JavaScript/CSS/Image files combined into a single resource

Resource Inlining

Inline JS/CSS/Images in the page

img1.example.com
img2.example.com
img3.example.com
...





Copying Strategies - The downsides

HTTP Pipelining

Head of Line blocking

Domain Sharding

Over sharding can kill performance

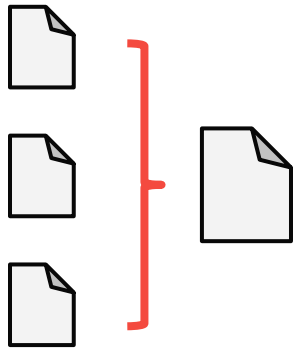
Concatenation and Spriting

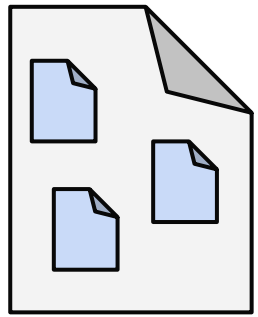
Increased complexity, can hurt cache performance and page load

Resource Inlining

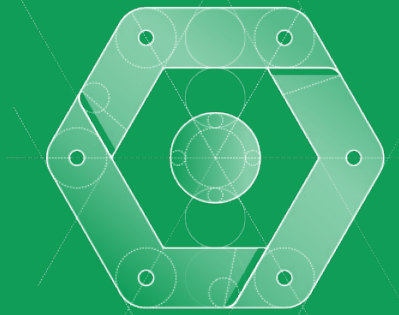
Can hurt cache performance, hard to get granularity right

img1.example.com
img2.example.com
img3.example.com
...





*"**HTTP/2** is a protocol designed for **low-latency transport of content** over the World Wide Web"*



- Improve end-user perceived latency
- Address the "head of line blocking"
- Not require multiple connections
- Retain the semantics of HTTP/1.1

Ilya Grigorik - **hpbn.co/http2 - High Performance Browser Networking**

HTTP/2 in one slide ...

One TCP connection

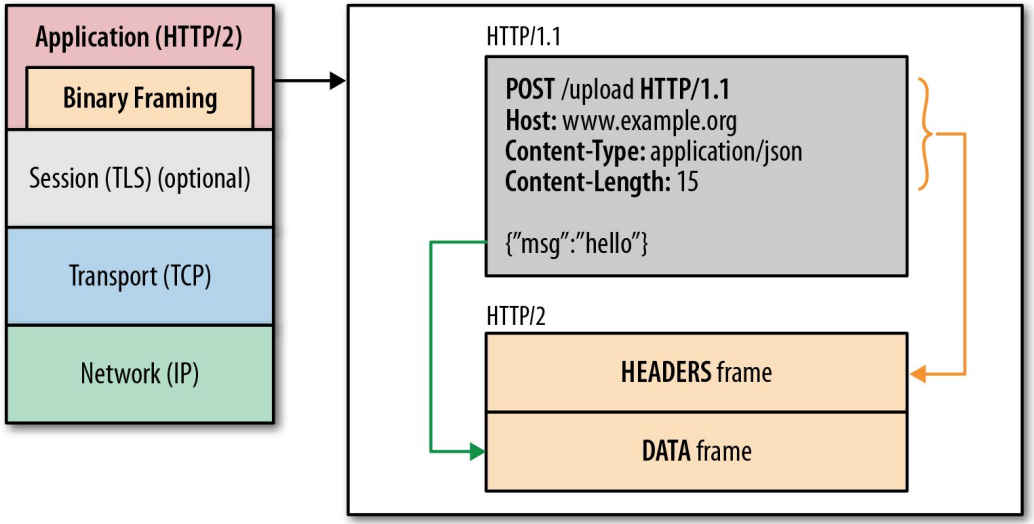
Request → Stream

- Streams are multiplexed
- Streams are prioritized

Binary framing layer

- Prioritization
- Flow control
- Server push

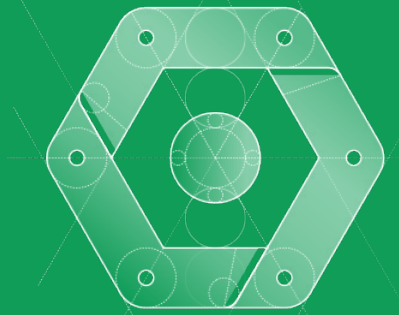
Header compression



*"... **we're not replacing all of HTTP** — the methods, status codes, and most of the headers you use today will be the same.*

*Instead, **we're redefining how it gets used "on the wire" so it's more efficient**, and so that it is more gentle to the Internet itself ...."*

- Mark Nottingham (HTTPbis chair)



HTTP/2 binary framing 101

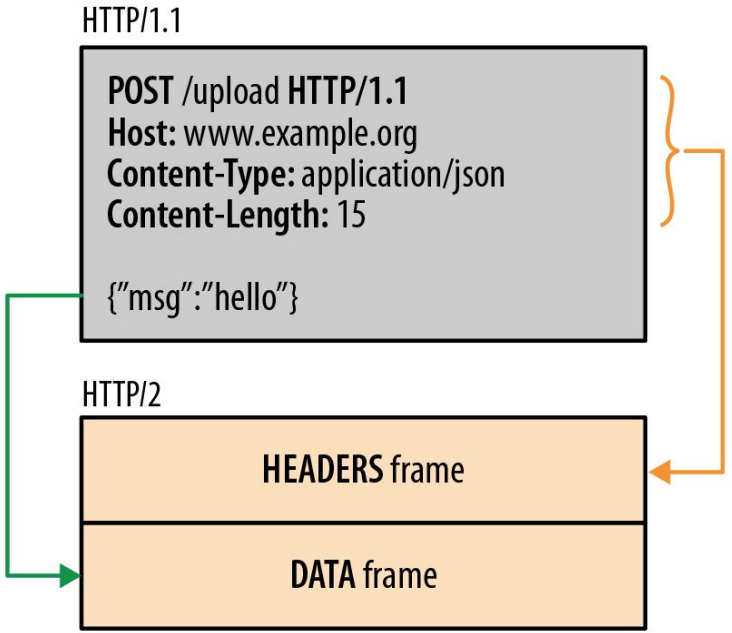
Sits between Socket interface and the API

HTTP messages are decomposed into one or more frames

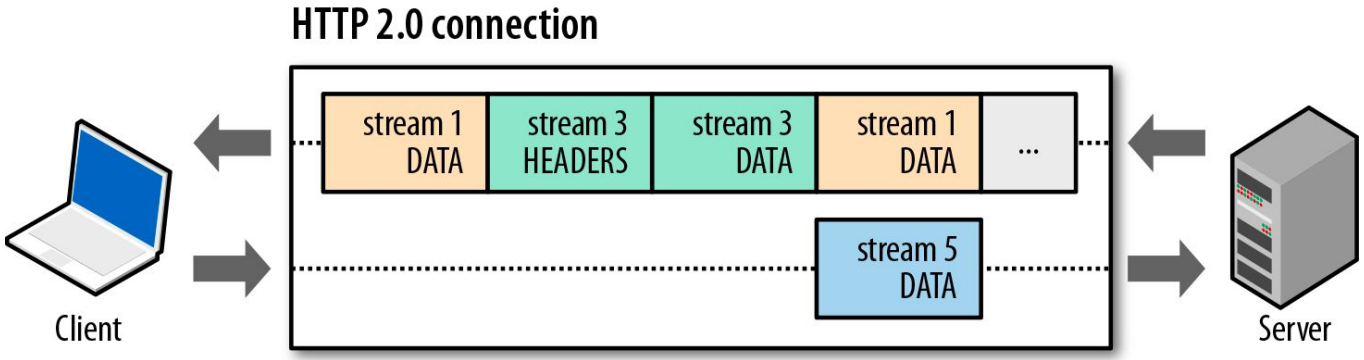
- HEADERS for meta-data
- DATA for payload
- RST_STREAM to cancel
- …

Each frame has a common header

- 9-byte, length prefixed
- Easy and efficient to parse



Basic data flow in HTTP 2.0...



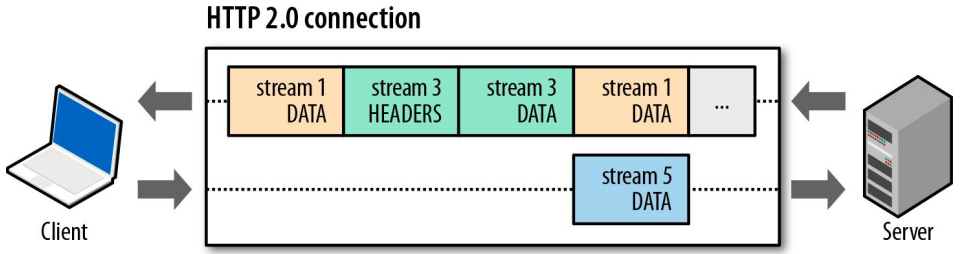
Basic data flow in HTTP 2.0...

Streams are multiplexed, because frames can be interleaved

All frames (e.g. HEADERS, DATA, etc) are sent over single TCP connection

Frame delivery is prioritized based on stream dependencies and weights

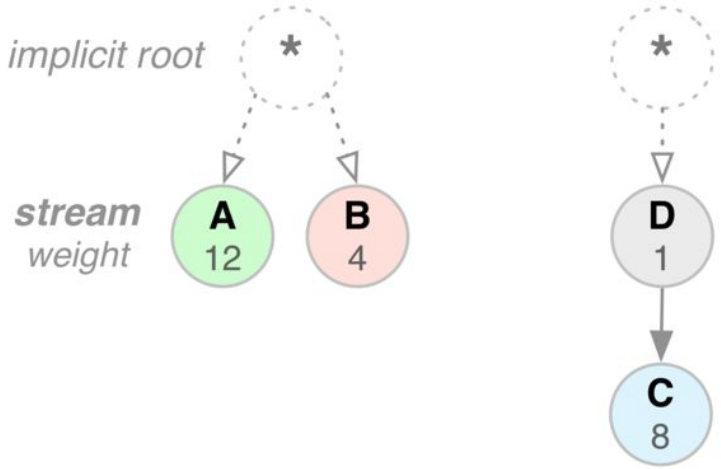
DATA frames are subject to per-stream and connection flow control



Stream prioritization in HTTP/2...

Each stream can have a weight
[1-256] integer value

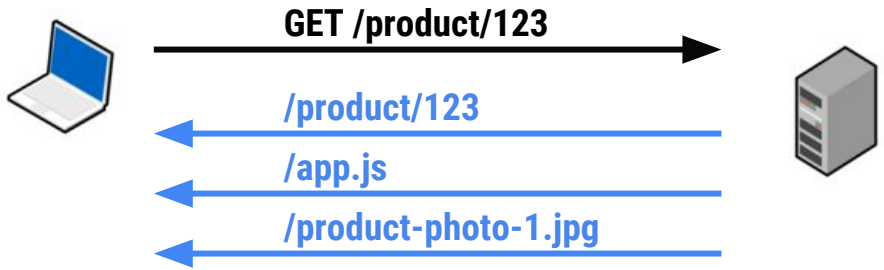
Each stream can have a dependency
parent is another stream ID



1. *E.g.. style.css (“A”) should get 2/3rd’s of available resources as compared to logo.jpg (“B”)*
2. *E.g.. product-photo-1.jpg (“D”) should be delivered before product-photo-2.jpg (“C”)*

***NOTE**: Prioritization is an advisory optimization hint to the server*

Server push, “because you’ll also need…”



HTTP/2 server: *“You asked for /product/123, I know you’ll also need app.js and product-photo-1.jpg, so… I promise to deliver these to you, no need to ask for them. That is, unless you decline or cancel.”*

- Server push is optional and can be disabled by the client
- Server push is subject to flow control - e.g. “you can only push 5Kb”
- Pushed resources can be cached and prioritized individually

Per-stream flow control

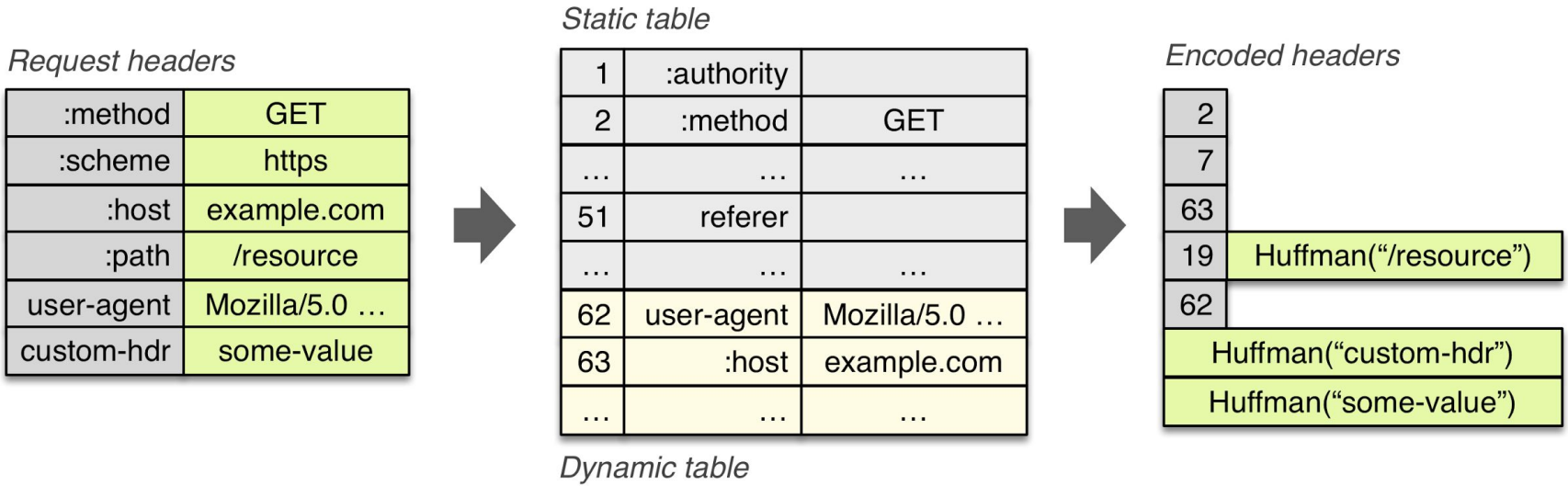
HTTP/2 client: *"I want photo.jpg, please send the first 20KB."*

HTTP/2 server: *"Ok, 20KB... pausing stream until you tell me to send more."*

HTTP/2 client: *"Send me the rest now."*

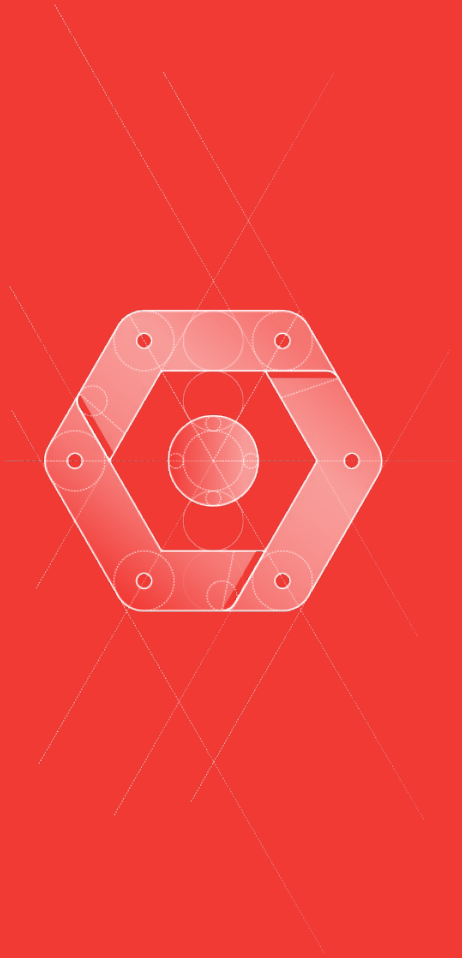
- Flow control allows the client to pause stream delivery, and resume it later
- Flow control is a "credit-based" scheme
  - *Sending DATA frames decrements the window*
  - *WINDOW_UPDATE frames update the window*

HPACK header compression



- *Literal values are (optionally) encoded with a static Huffman code*
- *Previously sent values are (optionally) indexed*
  - *e.g. “2” in above example expands to “method: GET”*

Protocol Buffers



What are Protocol Buffers?

Structured representation of data

Google's lingua franca for data
- 48k+ Message Types
- 12k+ Proto files

Evolutionary Development

Incrementally solved problems, Now used for:
- RPC Systems
- Persistent Data Storage

Why Protocol Buffers?

Protocol buffers:

- Efficient and compact binary data representation
- Clear compatibility rules; can easily be extended over time
- Generates idiomatic, easy to use classes for many languages
- Strongly typed; less error prone

Why not?

Protocol Buffers also are:

No use as a markup language

Not human readable (native format)

Only meaningful if you have the message definition

Message Format (proto2)

Uniquely numbered fields
Typed
Hierarchical Structure
Optional and Repeated fields
Extensible without breaking backward compatibility

```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 4;
}
```

Message Format (proto3)

Protocol Buffers language version 3

Specified by syntax = “proto3”;

All fields are optional in proto3

No user specified default values

No groups (FYI for those that use them)

```
syntax = “proto3”;

message Person {
  string name = 1;
  int32 id = 2;
  string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    string number = 1;
    PhoneType type = 2;
  }
  repeated PhoneNumber phone = 4;
}
```

Extensible

Add new fields without breaking backwards-compatibility

old implementations ignore the new fields when parsing

In proto3 any field can be removed, but don’t renumber existing fields

```
syntax = “proto3”;

message Person {
  string name = 1;
  int32 id = 2;
  string email = 3;
  address addr = 5;

  message address {
    string firstLine = 1;
    string secondLine = 2;
    string postalCode = 3;
    string country = 4;
  }

  ...
}
```

Protocol Buffer Compiler

https://github.com/google/protobuf

gRPC

gRPC goals

Build an open source, standards-based, best-of-breed, feature-rich RPC system

efficient and idiomatic

Create easy-to-use, efficient and idiomatic libraries

performant and scalable

Provide a performant and scalable RPC framework

micro-services

Enable developers to build microservice-based applications

gRPC in a nutshell

IDL to describe the API

Automatically generated servers and clients in 10+ languages

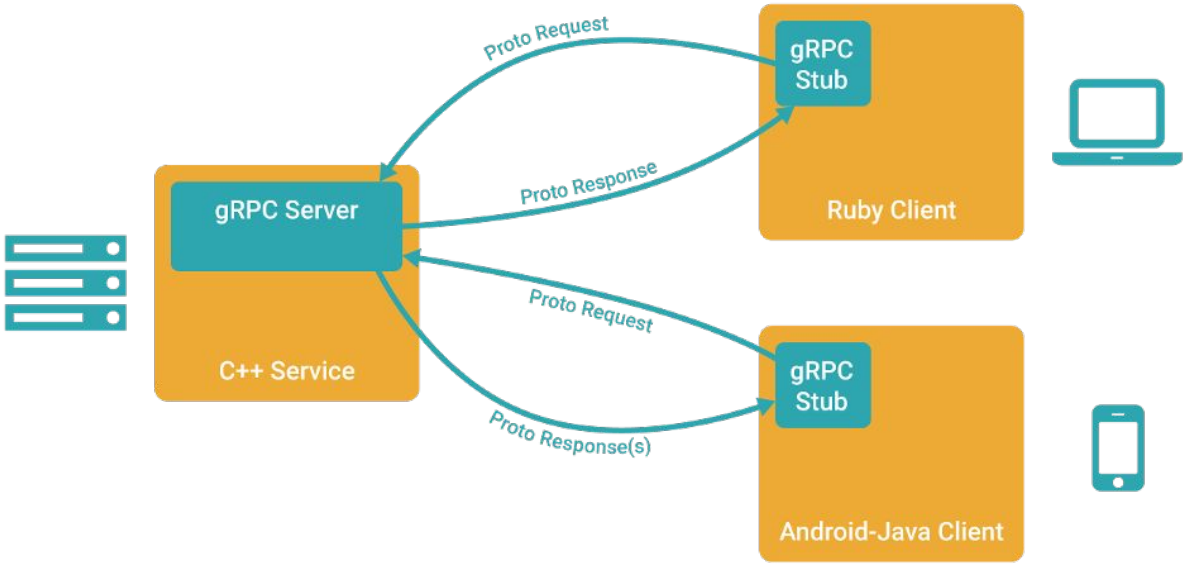
Takes advantage of feature set of HTTP/2

Lightweight open connections

Point to point

Streaming! Bidirectional streaming!

gRPC client/server architecture

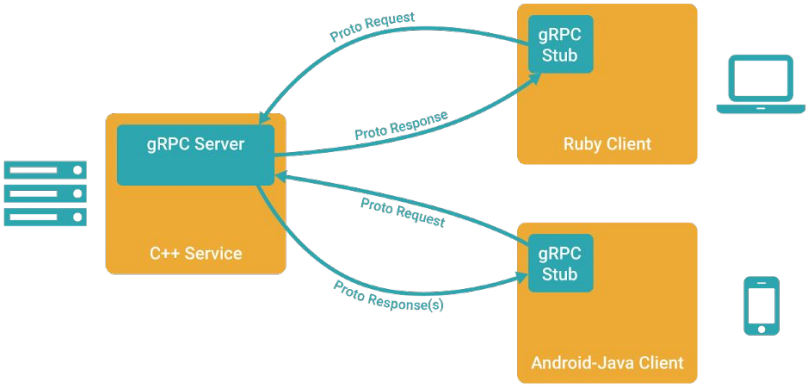


Getting Started

Define a service in a .proto file using Protocol Buffers IDL

Generate server and client code using the protocol buffer compiler with grpc plugin

Use the gRPC API to write a simple client and server for your service in the languages of your choice



RPC Messages

The most common use-case is a unary request-response RPC

Service Definition

```
service RouteGuide {
  rpc GetFeature(Point) returns (Feature);
}
message Point {
  int32 latitude = 1;
  int32 longitude = 2;
}
message Feature {
  string name = 1;
  Point location = 2;
}
```

Ordered Bidirectional Streaming RPC

The second most common use-case is sending multiple request or response messages in the context of a single RPC call

Server Streaming RPC Definition

```
service RouteGuide {
  rpc ListFeatures (Rectangle) returns (stream Features);
  ...
}

message Rectangle {
  Point lo = 1;
  Point hi = 2;
}
```

Client Streaming RPC Definition

```
service RouteGuide {
  rpc RecordRoute (stream Point) returns (RouteSummary);
  ...
}

message RouteSummary {
  int32 point_count = 1;
  int32 feature_count = 2;
  int32 distance = 3;
  int32 elapsed_time = 4;
}
```

Bidirectional Streaming RPC Definition

```
service RouteGuide {
  rpc RouteChat (stream RouteNote) returns (stream RouteNote);
  ...
}

message RouteNote {
  string location = 1;
  string message = 2;
}
```

gRPC Language Support

Implementations

- C core
  - Native bindings in C++, Node.js, Python, Ruby, ObjC, PHP, C#
- Java using Netty or OkHttp (+ inProcess for testing)
- Go

Authentication

SSL/TLS

gRPC has SSL/TLS integration and promotes the use of SSL/TLS to authenticate the server, and encrypt all the data exchanged between the client and the server. Optional mechanisms are available for clients to provide certificates to accomplish mutual authentication.

OAuth 2.0

gRPC provides a generic mechanism to attach metadata to requests and responses. Can be used to attach OAuth 2.0 Access Tokens to RPCs being made at a client.

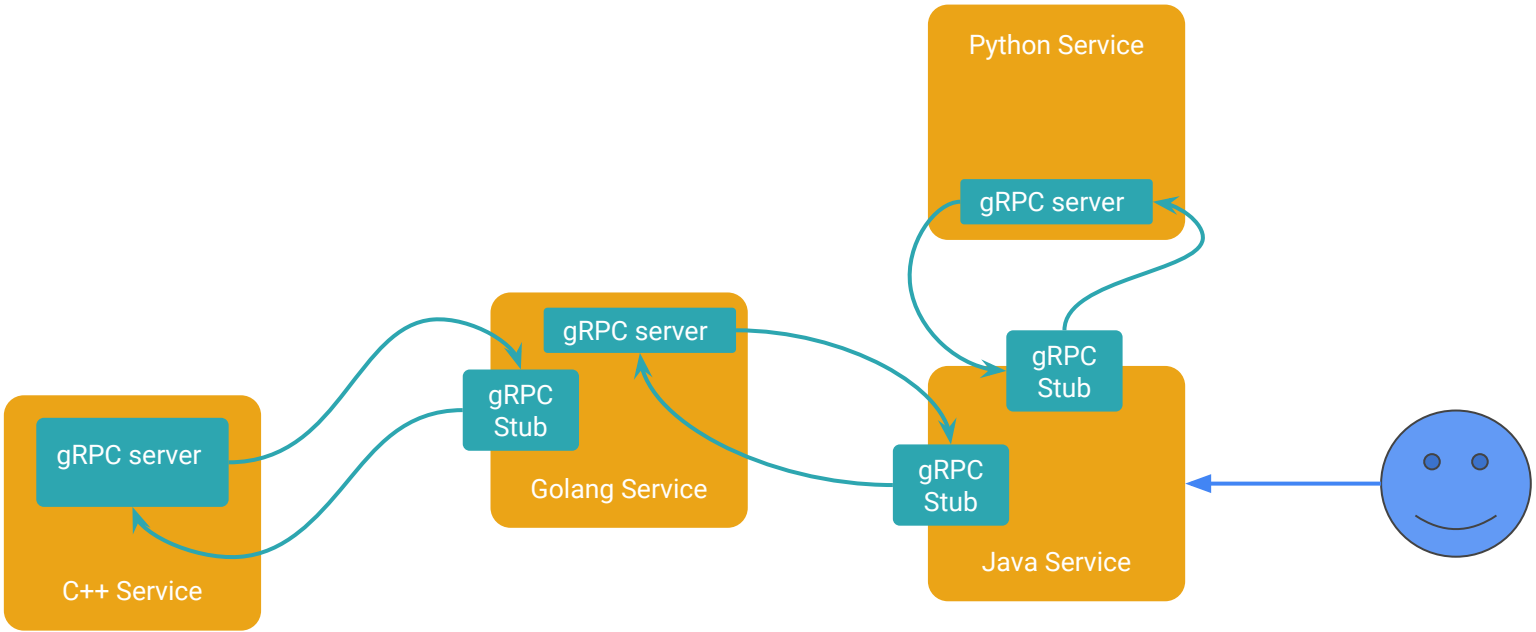
MicroServices using gRPC

Multi-language
Freedom to pick the language independently for each micro-service, based on performance, library availability, team expertise etc

Loosely coupled development
Blocks of functionality can be broken off into separate MicroService. Allows organic growth

High Performance
Make use of the strengths of HTTP/2 and Protocol Buffers

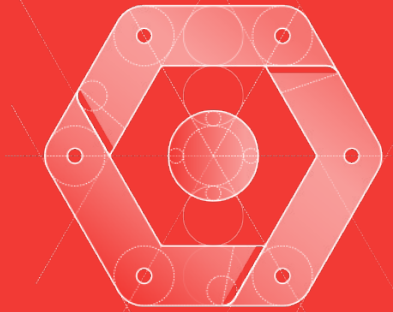
Polyglot Microservices Architecture



Demo gRPC

Fingers crossed :)

Mobile



Google Cloud Platform

Abelana Mobile Sample App

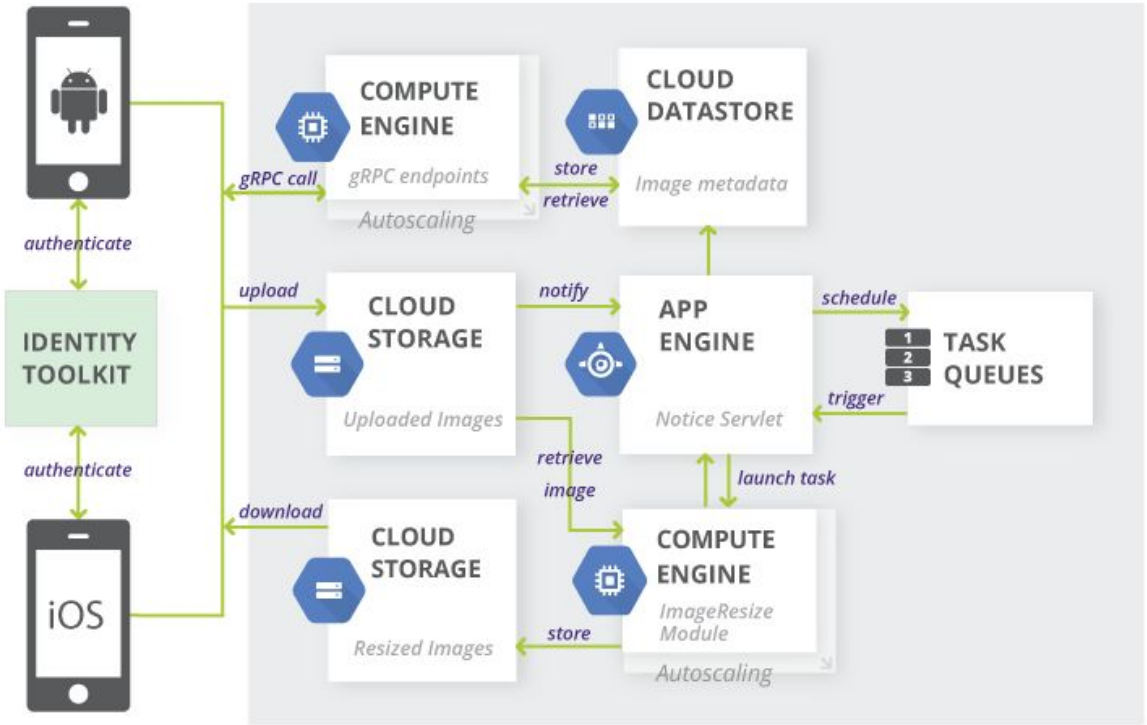
Photo-sharing App

Mobile Clients communicate with gRPC endpoint

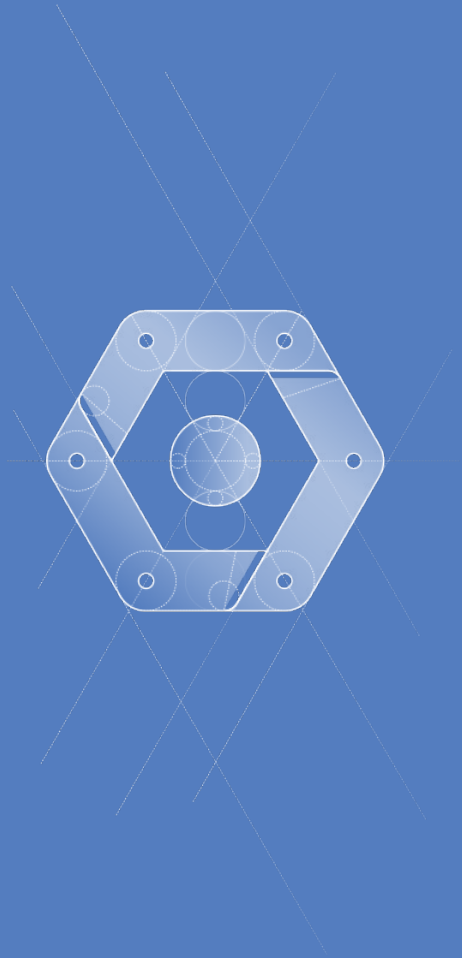
Clients for iOS and Android

https://cloud.google.com/solutions/mobile/image-management-mobile-apps-grpc

Abelana Architecture



Wrap-up



grpc is **Open Source**

We want your help!

http://grpc.io/contribute

https://github.com/grpc

irc.freenode.net *#grpc*

@grpcio

grpc-io@googlegroups.com

Thank you! Danke und auf Wiedersehen :)

@hostirosti @grpcio @googlecloud

Google Cloud Platform