GRPC

GRPC

- Google open sourced in Feb 2015
- **Transport**: `HTTP/2`
- **Wire format**: `Protocol Buffers v3` (Binary)
- **Service definition**: `Protocol Buffers IDL`
- Libraries in ~10 languages (native C, Go, Java)
- Microservices framework

What is gRPC for? (from official FAQ)

- Low latency, highly scalable, distributed systems
- Developing mobile clients which are communicating to a cloud server
- Designing a new protocol that needs to be accurate, efficient and language independent
- Layered design to enable extension e.g.

The Alternative?

- HTTP-JSON-REST APIs/Microservices
- **Transport**: `HTTP/1.1`
- **Wire format**: `JSON` (Text)
- **Service definition**:
  - `REST`, `Swagger`, `API Blueprint`
  - `JSON Schema`

HTTP 1.x: Limited Parallelism

New TCP connection per HTTP connection

Number of parallel HTTP requests

=

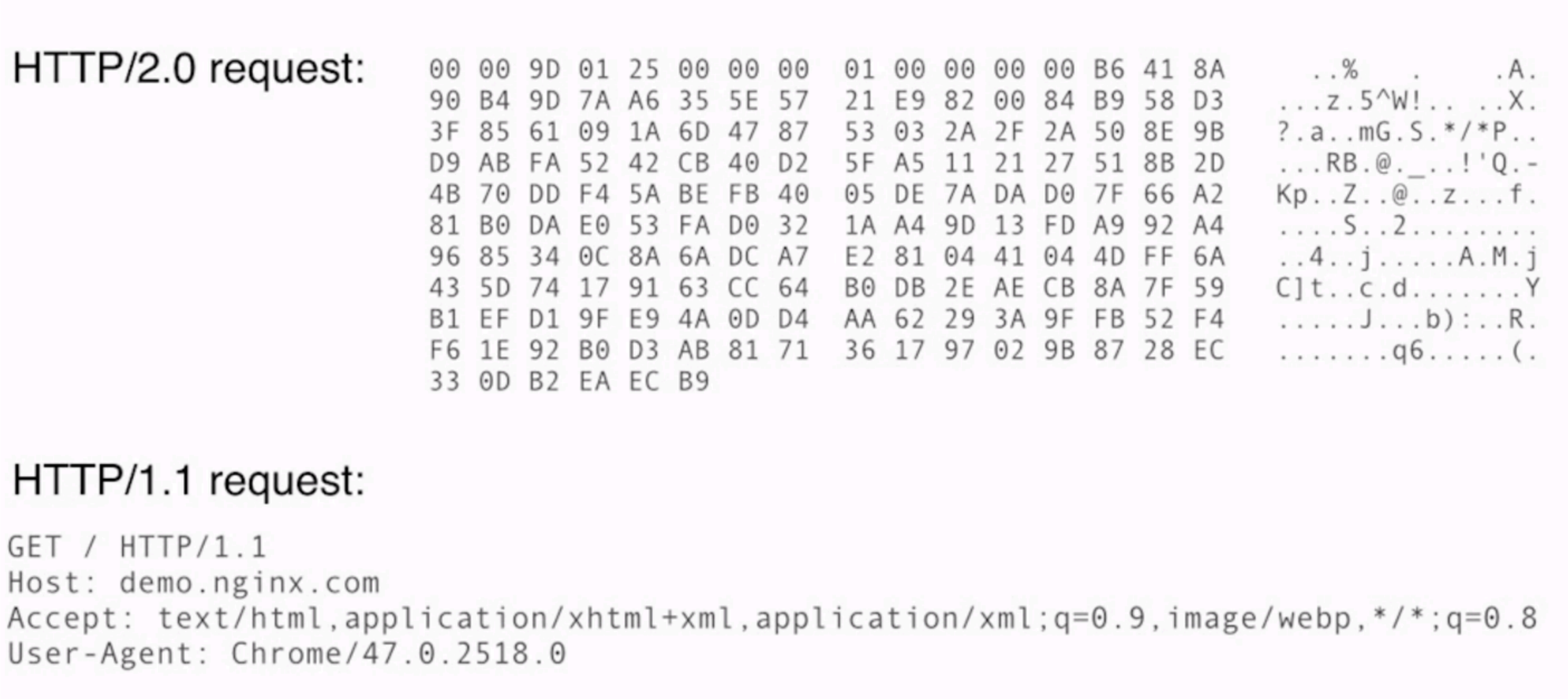
Number of TCP connections.

HTTP Headers

Uncompressed plain text headers for each and every HTTP request

HTTP/2 & Protobuf 101

HTTP/2 - Binary



HTTP/2.0 request:

```
00 00 9D 01 25 00 00 00  01 00 00 00 00 B6 41 8A     ..%    .    .A.
90 B4 9D 7A A6 35 5E 57  21 E9 82 00 84 B9 58 D3   ...z.5^W!.. ..X.
3F 85 61 09 1A 6D 47 87  53 03 2A 2F 2A 50 8E 9B   ?.a..mG.S.*/*P..
D9 AB FA 52 42 CB 40 D2  5F A5 11 21 27 51 8B 2D   ...RB.@._..!'Q.-
4B 70 DD F4 5A BE FB 40  05 DE 7A DA D0 7F 66 A2   Kp..Z..@..z...f.
81 B0 DA E0 53 FA D0 32  1A A4 9D 13 FD A9 92 A4   ....S..2........
96 85 34 0C 8A 6A DC A7  E2 81 04 41 04 4D FF 6A   ..4..j.....A.M.j
43 5D 74 17 91 63 CC 64  B0 DB 2E AE CB 8A 7F 59   C]t..c.d.......Y
B1 EF D1 9F E9 4A 0D D4  AA 62 29 3A 9F FB 52 F4   .....J...b):..R.
F6 1E 92 B0 D3 AB 81 71  36 17 97 02 9B 87 28 EC   .......q6.....(.
33 0D B2 EA EC B9
```

HTTP/1.1 request:

```
GET / HTTP/1.1
Host: demo.nginx.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Chrome/47.0.2518.0
```

HTTP/2 Request/Response Multiplexing

Interleave multiple requests and responses in parallel without blocking on any one

Use a **single TCP connection** to deliver multiple requests and responses in parallel.

Enable flow-control, server push, etc.

HTTP/2 - Streams

- 'independent, bidirectional sequence of frames exchanged between the client and server within an HTTP/2 connection'
- beyond request/response
- effectively supercedes 'websockets'

Protocol Buffers

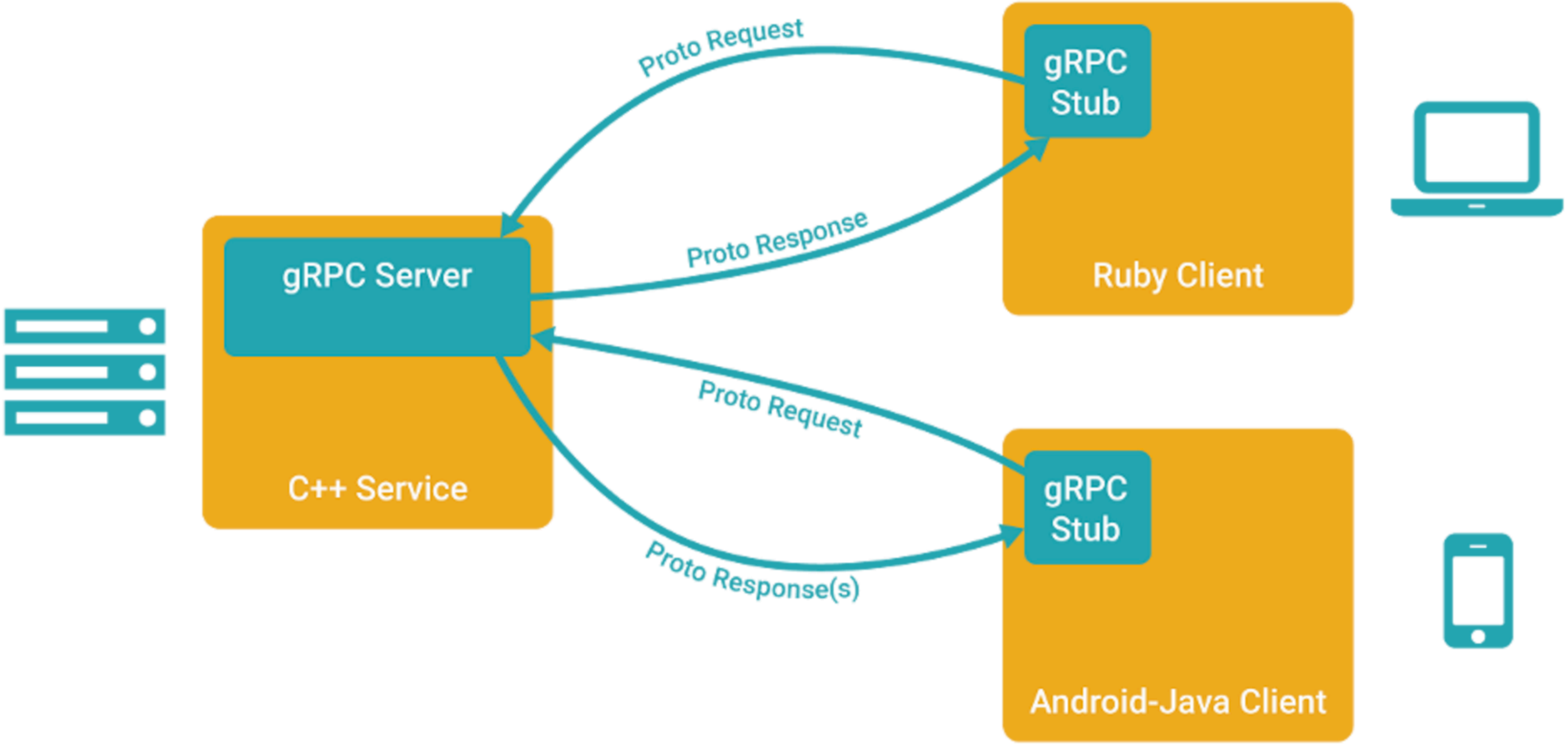
- mechanism for serializing structured data
- Interface Definition Language (IDL)
- binary, compact, fast
- versioned

```
syntax = "proto3";
```

Create contract

```
service GreetingService {

  rpc greeting (HelloRequest) returns (HelloResponse) {} }

message HelloRequest {

   string name = 1;
 }

 message HelloResponse {

  string message = 1;


}
```

Generate Server Interfaces and Client Stubs



Generate Server Interfaces and Client Stubs

Generate client and server code to extend from using proto3 compiler

For Java, there is **protobuf-maven-plugin** for Maven and **protobuf-gradle-plugin** for Gradl to help

For .NET, **Grpc.Tools.1.0.1** NuGet package has **protoc.exe**

Implement Server

Create a **service implementation** extending from generated base class

Create a **server** with port and using the service implementation

Start the server

Implement Client

Create a **channel** for the connection

Create a **request**

Send the request using the stub

Handle the responses in sync or async mode

Unary

Unary RPCs where the client sends a single request to the server and gets a single response back, just like a normal function call.

Server streaming

The client sends a request to the server and gets a stream to read a sequence of messages back. The client reads from the returned stream until there are no more messages

Client streaming

The client send a sequence of messages to the server using a provided stream. Once the client has finished writing the messages, it waits for the server to read them and return its response.

Bi-di streaming

Both sides send a sequence of messages using a read-write stream. The two streams operate independently. The order of messages in each stream is preserved.

Key Benefits

- Focus on the service/API design
- Freedom to pick language which suits the problem
- Server-to-server friendly
- Server-to-mobile friendly
- Growing community. Square, CoreOS, Docker.

References

- http://www.grpc.io/
- https://developers.google.com/protocol-buffers/—
- gRPC with REST and Open APIs