Intro to gRPC-Web

Stanley Cheung @ Google

What is gRPC?

gRPC **R**emote **P**rocedure **C**alls.

It is a high performance, open source, general purpose, standards-based, feature-rich RPC framework.

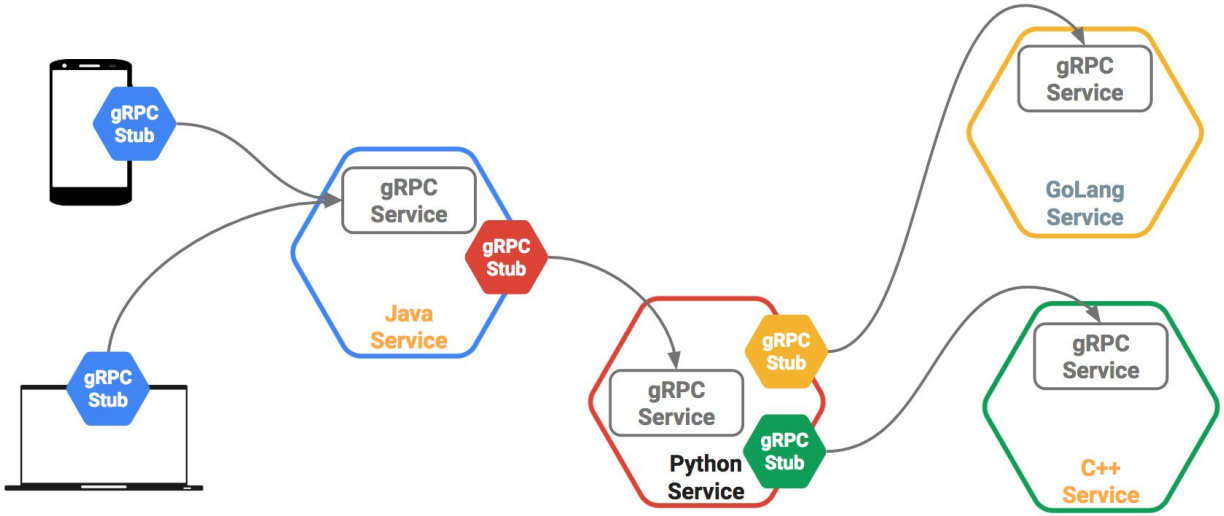
gRPC Speaks Your Language

Service definitions and client libraries

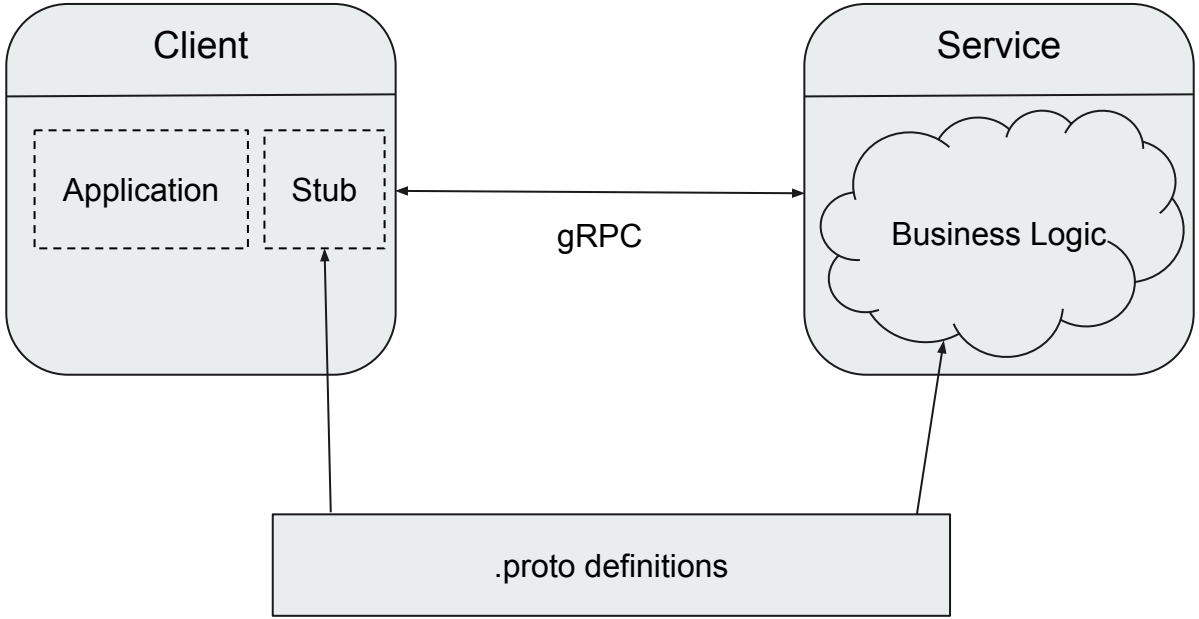
- Java
- Go
- C/C++
- C#
- Node.js
- PHP
- Ruby
- Python
- Objective-C

More Languages...

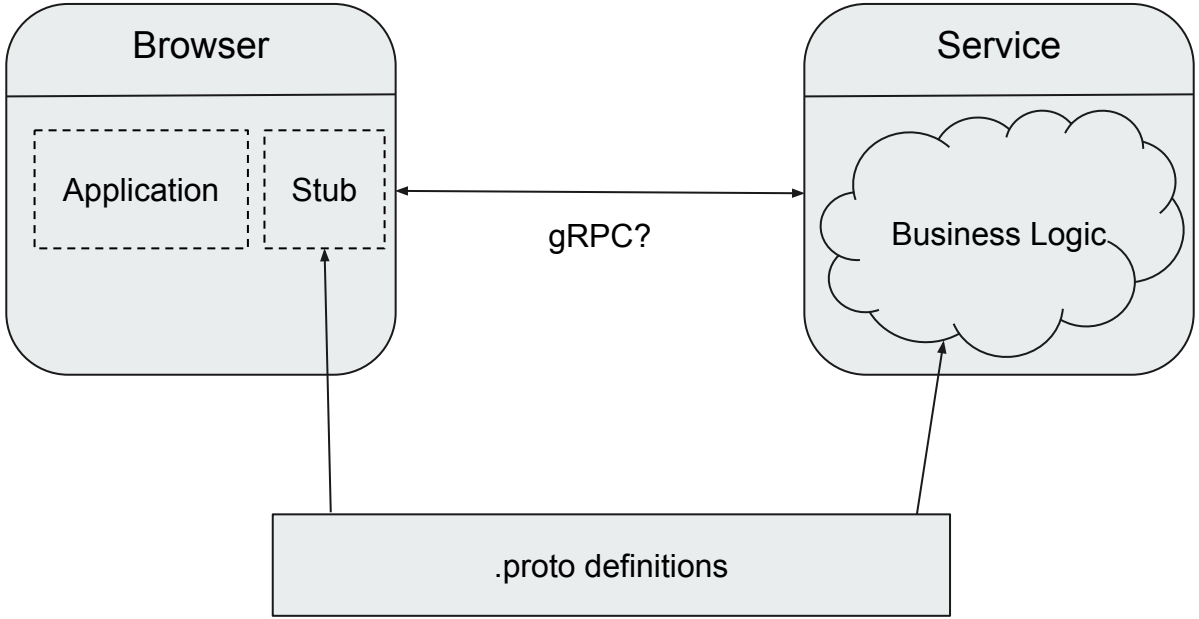
- Swift
- Haskell
- Rust
- ....



gRPC Basics


Client
Application
Stub
gRPC
Service
Business Logic
.proto definitions

gRPC-Web


Browser
Application
Stub
gRPC?
Service
Business Logic
.proto definitions

Not so Fast!

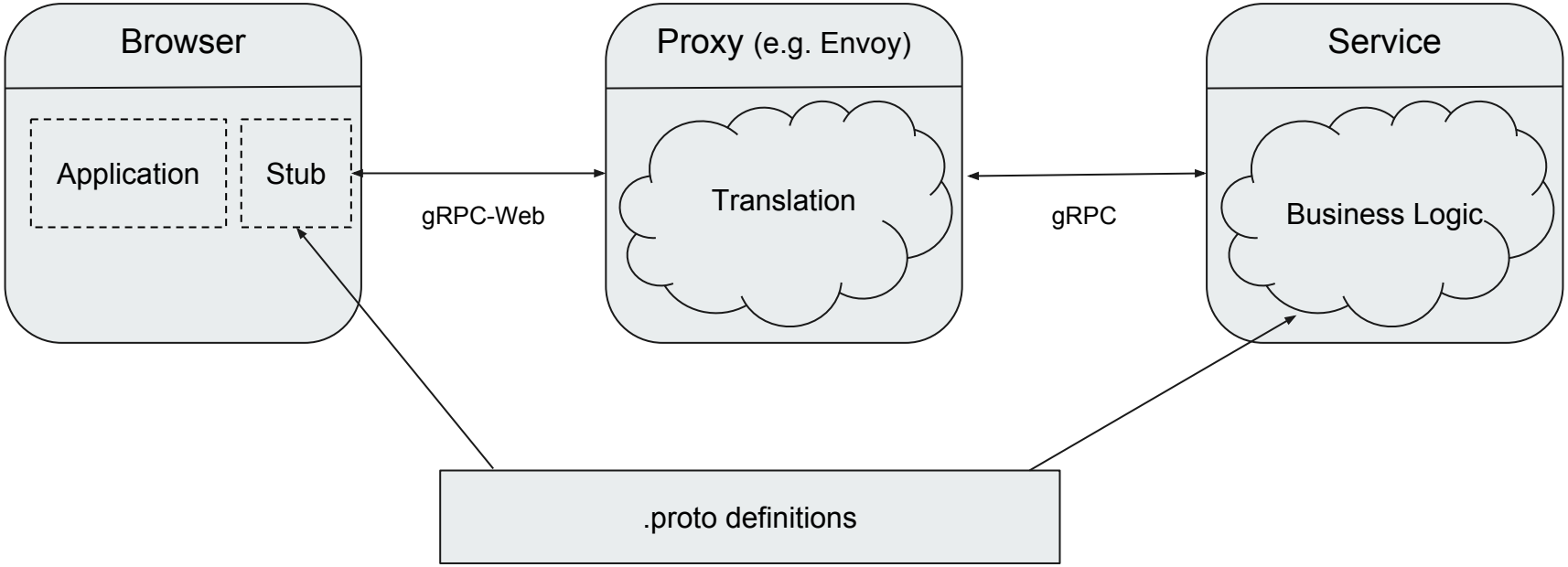
- Standard Web APIs (XHR, Fetch) don’t expose HTTP wire-transport details: protocol negotiation, “channels”, H2-only frames, stream cancellation
- Web clients prefers text data: security, JSON compatible encoding, streaming
- Response trailers are not supported
- Web-specific features: CORS, full-body compression only, security (XSRF/CSP)

gRPC-Web Spec

- gRPC-Web is an auxiliary protocol providing a translation layer between browser requests and gRPC
- Currently, the spec is implemented in Envoy. More to come
- Over HTTP/*, as negotiated by browsers (against proxies)
- Content-Type: application/grpc-web[-text][+proto]
- Trailers encoded into the response stream
- Current limitations: unary calls and server streaming only

https://github.com/grpc/grpc/blob/master/doc/PROTOCOL-WEB.md

gRPC-Web


Browser
Application
Stub
gRPC-Web
Proxy (e.g. Envoy)
Translation
gRPC
Service
Business Logic
.proto definitions

gRPC-Web

- GA Since Oct 2018
- Used internally in Google / Alphabet for over 2 years
- Cross-browser compatibility (supported by Google Closure library)
- Spec, Examples: https://github.com/grpc/grpc-web
- Client library: npm install grpc-web
- Experimental TypeScript support

Envoy

- gRPC-Web support is out-of-the-box.
- Enable the gRPC-Web HTTP filter in your envoy.yaml config file.

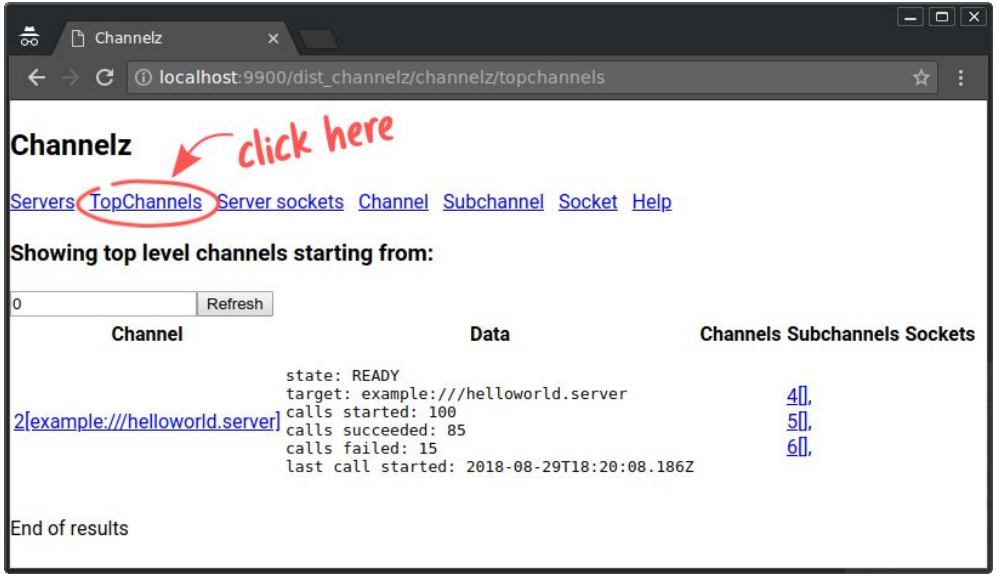
```
- filters:
  - name: envoy.http_connection_manager
    config:
      codec_type: auto
      stat_prefix: ingress_http
      route_config:
        virtual_hosts:
          domains: ["*"]
          routes:
          - match: { prefix: "/" }
            route: { cluster: greeter_service }
```

```
http_filters:
  - name: envoy.grpc_web
  - name: envoy.cors
  - name: envoy.router
```

```
clusters:
  - name: greeter_service
    type: logical_dns
    http2_protocol_options: {}
    lb_policy: round_robin
    hosts: [{ socket_address: {
      address: backend-server }}]
```

Example: Channelz

- Implemented with gRPC-Web
- Shows debug info and stats for gRPC service



https://grpc.io/blog/a_short_introduction_to_channelz

Let’s dive into an example!

Start with a Protocol Buffer

- Start with defining messages you want to send and receive

```
syntax = "proto3";

package helloworld;

message HelloRequest {
  string name = 1;
}

message HelloReply {
  string message = 1;
}
```

Generate Code for your Application

- The code generator tool "`protoc`" converts your .proto into JavaScript classes
- CommonJS and Closure style imports are supported now

```
const {HelloRequest, HelloReply} =
  require('./helloworld_pb.js');

var request = new HelloRequest();
request.setName(‘John’);
```

Add Service Definition

- Let's add a simple RPC method
- We provide a plugin `"protoc-gen-grpc-web"` to generate the gRPC-Web client stub class

```
syntax = "proto3";
package helloworld;

service Greeter {
  rpc SayHello (HelloRequest)
    returns (HelloReply);
}

message HelloRequest {
  string name = 1;
}
message HelloReply {
  string message = 1;
}
```

Write your Client Code

- Import the generated code
- You can start making RPCs from your application!
- gRPC-Web offers a familiar and consistent API as gRPC Node

```
const {GreeterClient} =
  require('./helloworld_grpc_web_pb.js');

const client = new
  GreeterClient('https://api.myhost.com');

client.sayHello(request, metadata,
  (err, response) => {
    console.log(response.getMessage());
  });
```

Server Streaming Support

- Server streaming RPCs are supported in gRPC-Web. Add a `"stream"` qualifier to the response type

```
syntax = "proto3";
package helloworld;

service Greeter {
  rpc SayHello (HelloRequest)
    returns (HelloReply);

  rpc RepeatHello (HelloRequest)
    returns (stream HelloReply);
}
```

Server Streaming Support

- Streaming RPCs follow the Node Stream API.

```
var stream = client.repeatHello(
  request, metadata);

stream.on('data', (response) => {
  console.log(response.getMessage());
});

stream.on('metadata', (metadata) => {
  // …
});
```

Compile your JS

- Use your favorite tool to compile all your JavaScript source code into browser-consumable form
- We are looking into better integration into popular front-end frameworks

```
$ npm install
$ npx webpack
```

TypeScript Support

- We have added experimental TypeScript support.
- Contributions welcome!

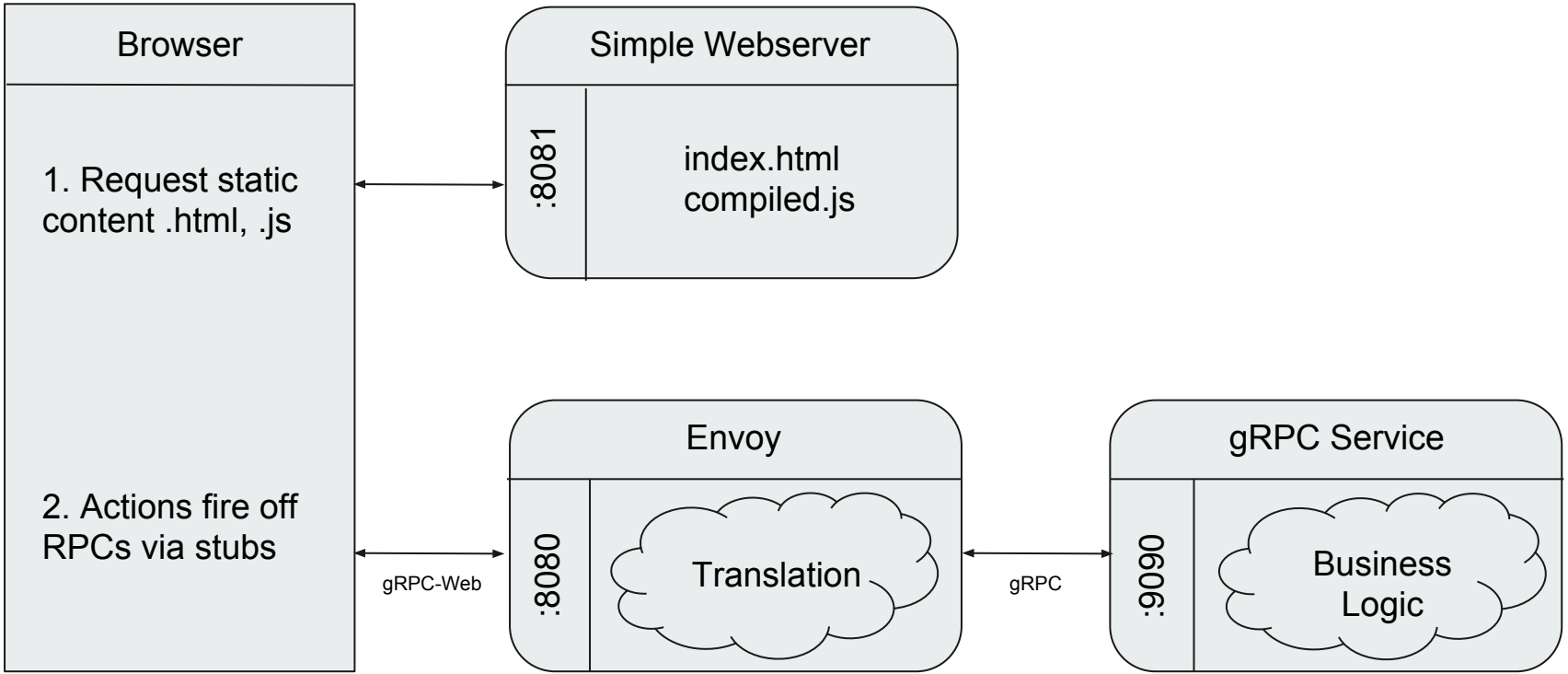
```
import * as grpcWeb from ‘grpc-web’;

const call = client.sayHello(
  request, metadata,
  (err: grpcWeb.Error,
   response: HelloReply) => {
    console.log(response.getMessage());
  });

call.on(‘status’,
  (status: grpcWeb.Status) => {
    // …
  });
```

Demo!

Demo


Browser
1. Request static content .html, .js
2. Actions fire off RPCs via stubs
Simple Webserver
:8081
index.html
compiled.js
gRPC-Web
Envoy
:8080
Translation
gRPC
gRPC Service
:9090
Business Logic

Future

- In-process connect support (e.g. Node, Java, Go)
- Interceptors
- Integrations into frameworks like Angular, React, etc

Questions