Flutter

Belgium

gRPC: It's Not Rocket Science, but It's Close!

Who am I?

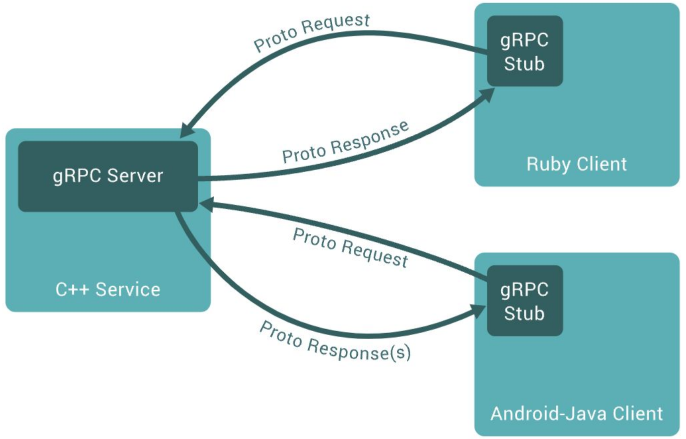
@KrisPypen krispypen @Kris Pypen

- Java developer: CMS, Webservices, Bolero,...
- PHP developer: Kunstmaan CMS, ...
- Android developer: Mobile applications
- **2018: Flutter Belgium Meetup organiser**
- **Flutter developer @ InvestSuite**
- ijsjesradar.app

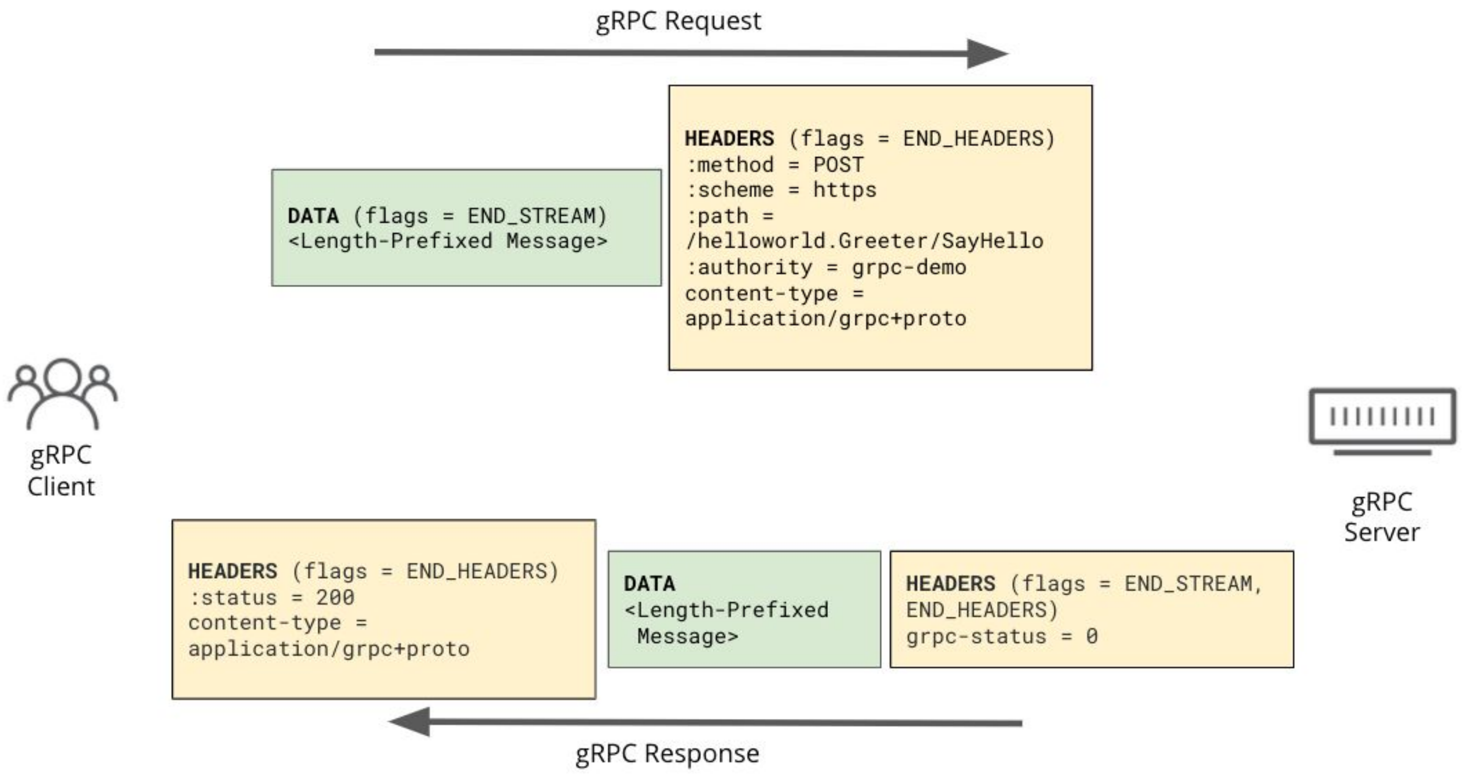
Kris Pypen

What is GRPC

gRPC is a cross-platform open source high performance remote procedure call (RPC) framework.



What is GRPC



```
DATA (flags = END_STREAM)
<Length-Prefixed Message>
```

```
HEADERS (flags = END_HEADERS)
:method = POST
:scheme = https
:path =
/helloworld.Greeter/SayHello
:authority = grpc-demo
content-type =
application/grpc+proto
```

```
HEADERS (flags = END_HEADERS)
:status = 200
content-type =
application/grpc+proto
```

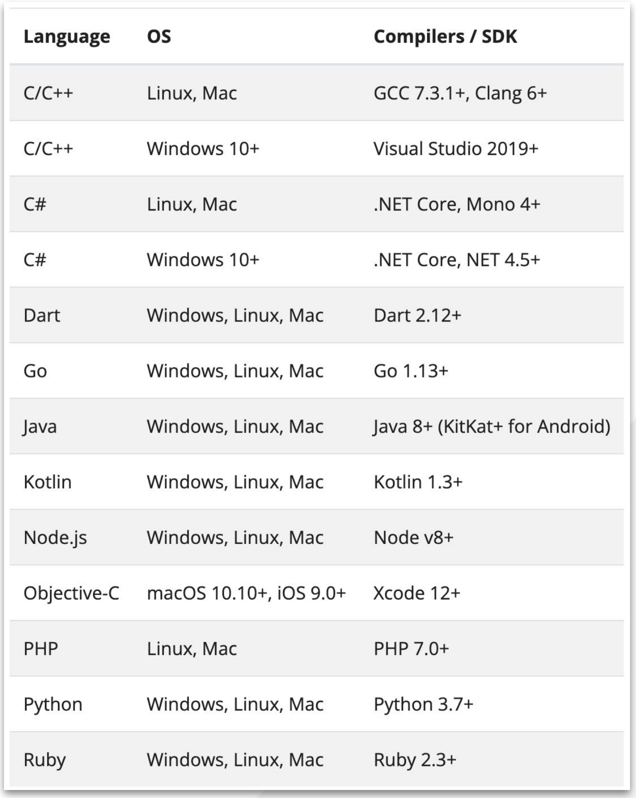
```
DATA
<Length-Prefixed
 Message>
```

```
HEADERS (flags = END_STREAM,
END_HEADERS)
grpc-status = 0
```

What is GRPC

Stubby	**Protobuf**	**gRPC**
2001	2008	2016
Internal @ Google	Public release	Public release

GRPC: Supported languages



Language	OS	Compilers / SDK
C/C++	Linux, Mac	GCC 7.3.1+, Clang 6+
C/C++	Windows 10+	Visual Studio 2019+
C#	Linux, Mac	.NET Core, Mono 4+
C#	Windows 10+	.NET Core, NET 4.5+
Dart	Windows, Linux, Mac	Dart 2.12+
Go	Windows, Linux, Mac	Go 1.13+
Java	Windows, Linux, Mac	Java 8+ (KitKat+ for Android)
Kotlin	Windows, Linux, Mac	Kotlin 1.3+
Node.js	Windows, Linux, Mac	Node v8+
Objective-C	macOS 10.10+, iOS 9.0+	Xcode 12+
PHP	Linux, Mac	PHP 7.0+
Python	Windows, Linux, Mac	Python 3.7+
Ruby	Windows, Linux, Mac	Ruby 2.3+

Some alternatives to GRPC

- REST
- GraphQL
- SOAP
- Apache Thrift

Protobuf

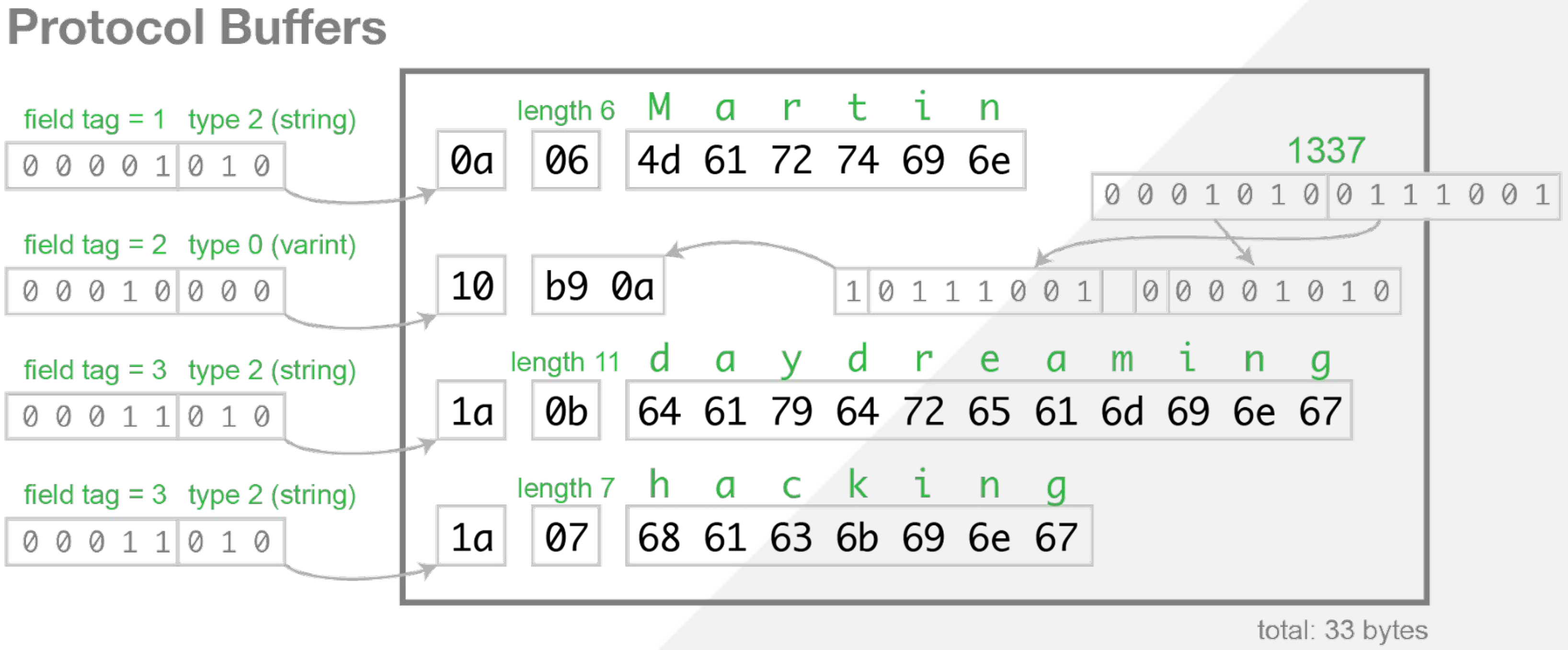
```
syntax = "proto3";
package weather.v1;

service WeatherInfoService {
 rpc GetCurrentWeatherInfo (WeatherInfoRequest)  returns (WeatherInfoResponse) {}
}

message GetCurrentWeatherInfoRequest {
 oneof main {
   string postal_code = 1;
   int location_id = 2;
 }
}
message WeatherInfoResponse {
 string message = 1;
 double temperature = 2;
 bool is_night = 3;
 bool is_cloudy = 4;
 bool is_rainy = 5;
}
```

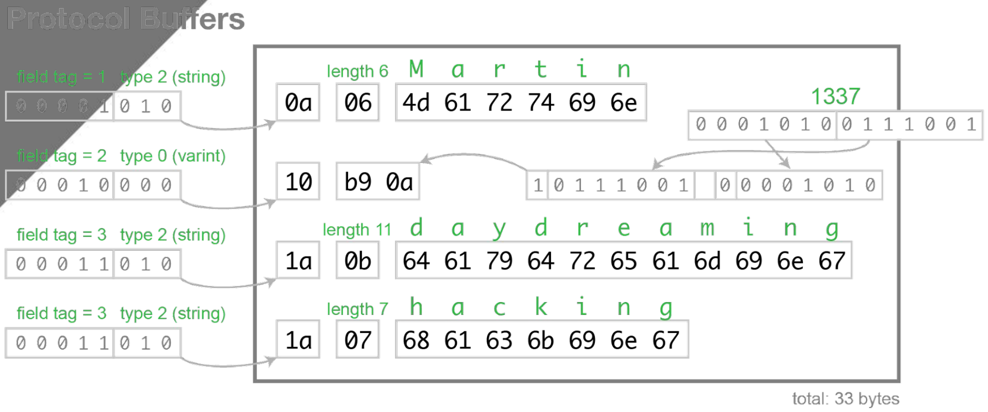
```
enum ErrorType {
 ERROR_TYPE_UNSPECIFIED =  0;
 ERROR_TYPE_NOTFOUND =  1;
 ERROR_TYPE_PERMISSION_DENIED =
2;
 ERROR_TYPE_BACKEND_GONE =  3;
 ERROR_TYPE_TOKEN_EXPIRED =  4;
 ERROR_TYPE_RATE_LIMITED =  5;
}
```

Protobuf


Protocol Buffers
field tag = 1 type 2 (string)
0 0 0 0 1 0 1 0
field tag = 2 type 0 (varint)
0 0 0 1 0 0 0 0
field tag = 3 type 2 (string)
0 0 0 1 1 0 1 0
field tag = 3 type 2 (string)
0 0 0 1 1 0 1 0
length 6
M a r t i n
0a 06 4d 61 72 74 69 6e
1337
0 0 0 1 0 1 0 0 1 1 1 0 0 1
10 b9 0a
1 0 1 1 1 0 0 1 0 0 0 0 1 0 1 0
length 11
d a y d r e a m i n g
1a 0b 64 61 79 64 72 65 61 6d 69 6e 67
length 7
h a c k i n g
1a 07 68 61 63 6b 69 6e 67
total: 33 bytes

Protobuf

JSON


Protocol Buffers
field tag = 1 type 2 (string)
0 0 0 0 1 0 1 0
field tag = 2 type 0 (varint)
0 0 0 1 0 0 0 0
field tag = 3 type 2 (string)
0 0 0 1 1 0 1 0
field tag = 3 type 2 (string)
0 0 0 1 1 0 1 0
0a 06 length 6 M a r t i n 4d 61 72 74 69 6e
1337
0 0 0 1 0 1 0 0 1 1 1 0 0 1
10 b9 0a
1 0 1 1 1 0 0 1 0 0 0 0 1 0 1 0
1a 0b length 11 d a y d r e a m i n g 64 61 79 64 72 65 61 6d 69 6e 67
1a 07 length 7 h a c k i n g 68 61 63 6b 69 6e 67
total: 33 bytes

```
{
  "firstName": "Martin",
  "age": 1337,
  "word1": "daydreaming",
  "word2": "hacking"
}
```

= 33 bytes

= 91 bytes

(Non-)breaking changes

message changes:

- add properties
- rename property
- remove fields (* unless your app relies on it)
- rename message (* if not using Any)
- add/remove enum types
- NEVER change type of a property!

Removing a field:

```
message WeatherInfo {
 string message = 1;
 double temperature = 2;
 reserved 3,4;
 bool is_rainy = 5;
}
```

```
message WeatherInfoResponse {
 string message = 1;
 double temperature = 2;
 bool is_night = 3;
 bool is_cloudy = 4;
 bool is_rainy = 5;
}
```

```
enum ErrorType {
 ERROR_TYPE_UNSPECIFIED =  0;
 ERROR_TYPE_NOTFOUND =  1;
 ERROR_TYPE_PERMISSION_DENIED =
2;
 ERROR_TYPE_BACKEND_GONE =  3;
 ERROR_TYPE_TOKEN_EXPIRED =  4;
 ERROR_TYPE_RATE_LIMITED =  5;
}
```

(Non-)breaking changes

How to do versioning:

```
com.acme.weather.v2.WeatherInfoService.GetLocations
```

```
com.acme.weather.WeatherInfoService.GetLocations V2
```

Performance

- Data format: gRPC employs Protocol Buffers, a **binary** serialization format => smaller payloads
- HTTP2 by default
- Streaming: gRPC supports bidirectional streaming, => continuous data exchange between client and server

Validating proto specs

```
> $ buf lint

weather/v1/weather.proto:49:19:message weather.v1.WeatherInfo: fields
is_cloudy and is_rainy both have the same tag 4

weather/v1/weather.proto:27:3:field weather.v1.WeatherInfo.is_rainy: unknown
type bbool
```

https://buf.build/

Protos to Dart

Converting proto specs to dart code:

```
$ protoc -Iprotos protospecs  --dart_out=grpc:grpc-api/lib/src/
```

Tip: generate every file on its own

```
$ find protospecs/ -iname "*.proto" -exec protoc
--dart_out=grpc:grpc-api/lib/src/ {} --proto_path protospecs \;
```

⇒ less git conflicts on numbers ☝️

Tip: generate code into a seperate pub, used by server and client

Error handling

gRPC Status codes:

Number	Code
0	OK
1	CANCELLED
2	UNKNOWN
3	INVALID_ARGUMENT
4	DEADLINE_EXCEEDED
5	NOT_FOUND
6	ALREADY_EXISTS
7	PERMISSION_DENIED
8	RESOURCE_EXHAUSTED
9	FAILED_PRECONDITION
10	ABORTED
11	OUT_OF_RANGE
12	UNIMPLEMENTED
13	INTERNAL
14	UNAVAILABLE
15	DATA_LOSS
16	UNAUTHENTICATED

Error handling with “oneof”

```
message GetCurrentWeatherInfoResponse {
 oneof main {
   WeatherInfo weather_info = 1;
   Error error = 2;
 }
}
message Error {
 ErrorType type = 1;
 map<string, string> message = 2; // key: locale, value: message
}
enum ErrorType {
 ERROR_TYPE_UNSPECIFIED = 0;
 ERROR_TYPE_NOTFOUND = 1;
 ERROR_TYPE_PERMISSION_DENIED = 2;
 ERROR_TYPE_BACKEND_GONE = 3;
 ERROR_TYPE_TOKEN_EXPIRED = 4;
 ERROR_TYPE_RATE_LIMITED = 5;
}
```

GRPC Server

pubspec.yaml:

```yaml
dependencies:
 grpc: ^3.1.0
 grpc_api:
   path: ../grpc-api
```

GRPC Server

```
class WeatherInfoService extends WeatherInfoServiceBase {
 @override
 Future<GetCurrentWeatherInfoResponse> getCurrentWeatherInfo(
     ServiceCall call, GetCurrentWeatherInfoRequest request) async {
   if (request.locationId == 'abc') {
     return GetCurrentWeatherInfoResponse(
         weatherInfo: WeatherInfo(temperature: 20.1, isRainy: false, isNight: true, message: 'Hello'));
   }
   throw const GrpcError.notFound();
 }

 @override
 Future<GetLocationsResponse> getLocations(ServiceCall call, GetLocationsRequest request) {
   throw const GrpcError.unimplemented();
 }
}

Future<void> main(List<String> args) async {
 final server = Server.create(
   services: [WeatherInfoService()],
   codecRegistry: CodecRegistry(codecs: const [GzipCodec()]),
 );
 await server.serve(port: 50051);
 print('Server listening on port ${server.port}...');
}
```

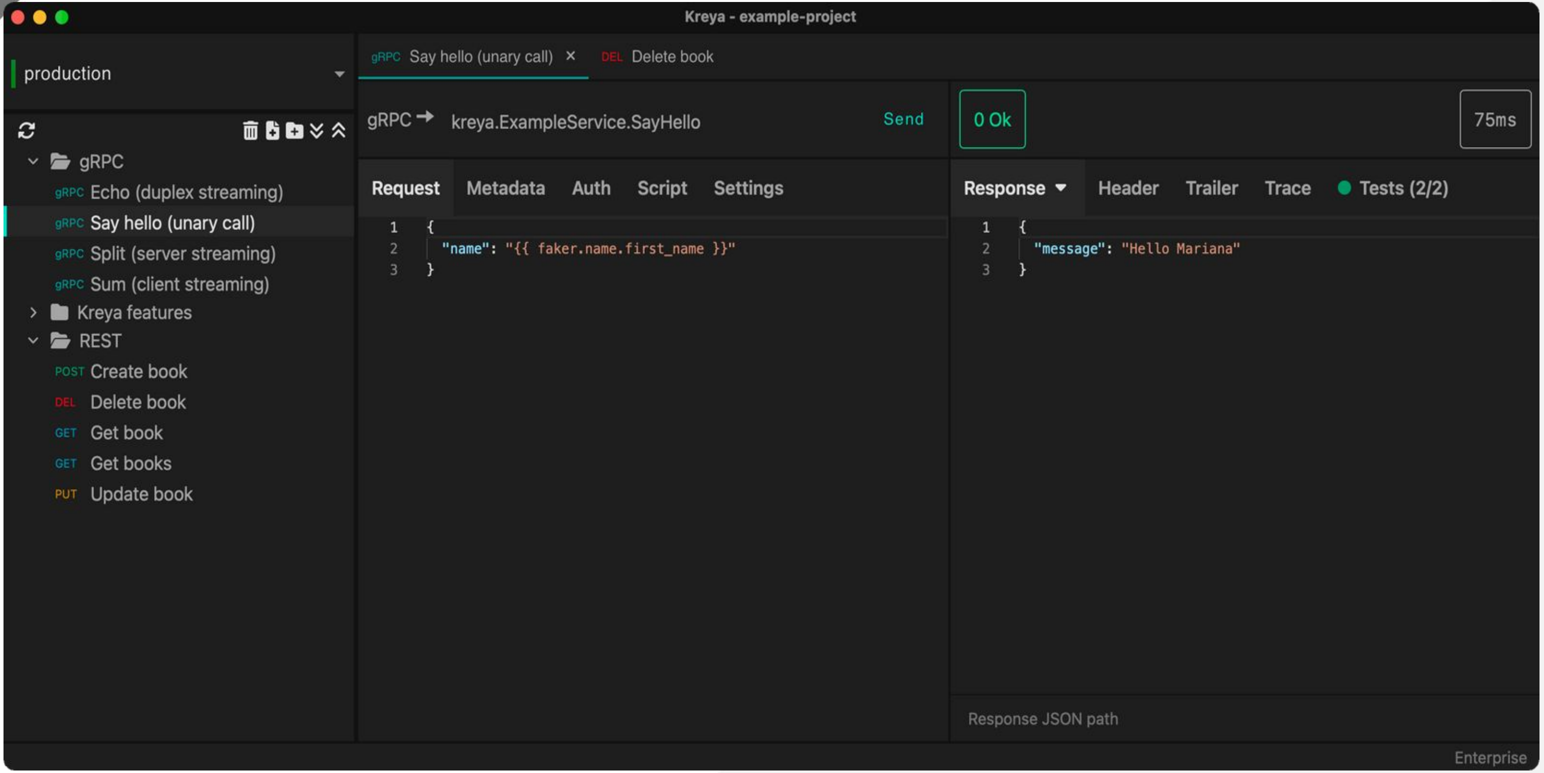
GRPC Server

<DEMO time!>

Debugging tools

- Kreya
- BloomRPC
- PostMan

Debugging tools: Kreya



Kreya - example-project

production

gRPC
- gRPC Echo (duplex streaming)
- gRPC Say hello (unary call)
- gRPC Split (server streaming)
- gRPC Sum (client streaming)

Kreya features

REST
- POST Create book
- DEL Delete book
- GET Get book
- GET Get books
- PUT Update book

gRPC Say hello (unary call) × DEL Delete book

gRPC → kreya.ExampleService.SayHello Send

0 Ok 75ms

Request Metadata Auth Script Settings

```
{
  "name": "{{ faker.name.first_name }}"
}
```

Response ▾ Header Trailer Trace ● Tests (2/2)

```
{
  "message": "Hello Mariana"
}
```

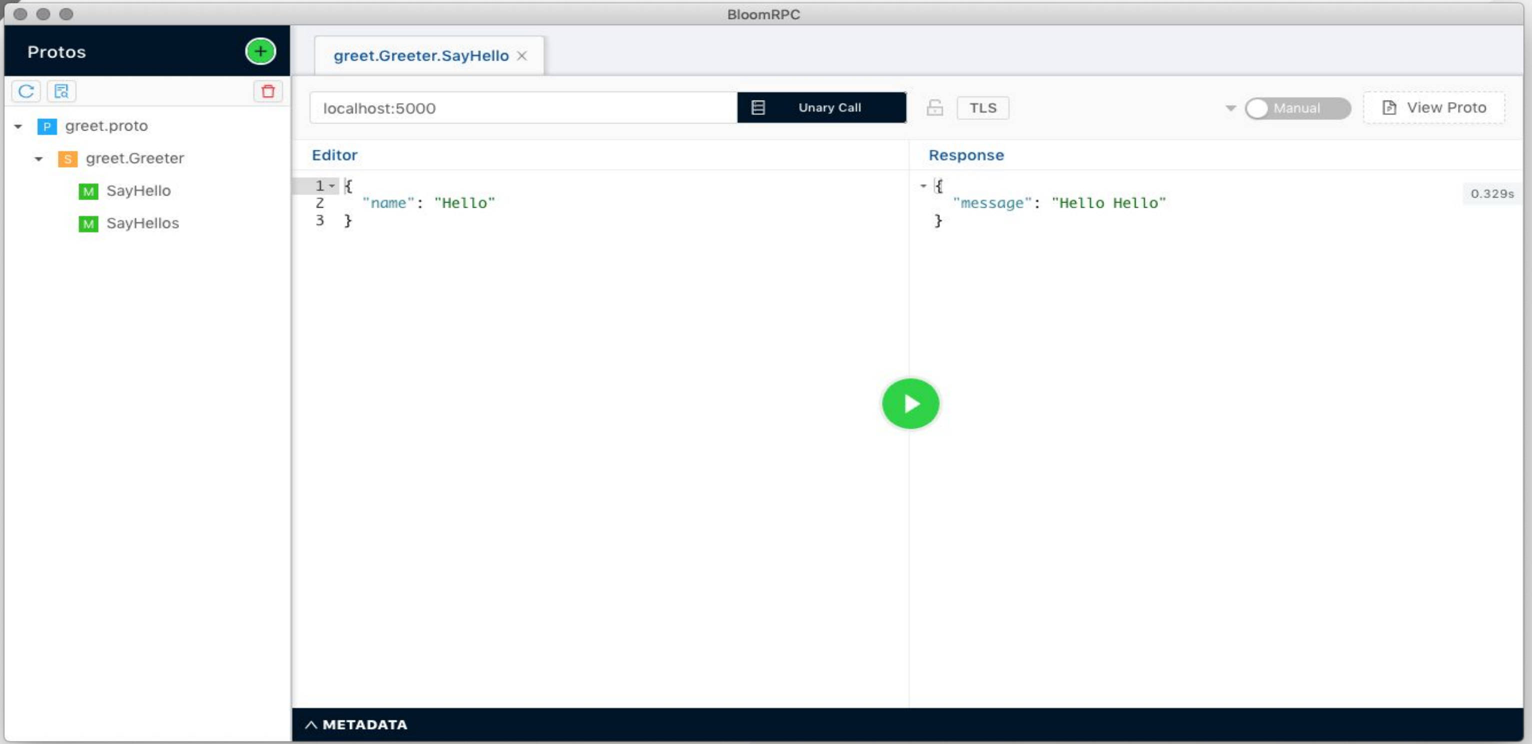
Response JSON path

Enterprise

Debugging tools: **Kreya**

<DEMO>

Debugging tools: BloomRPC



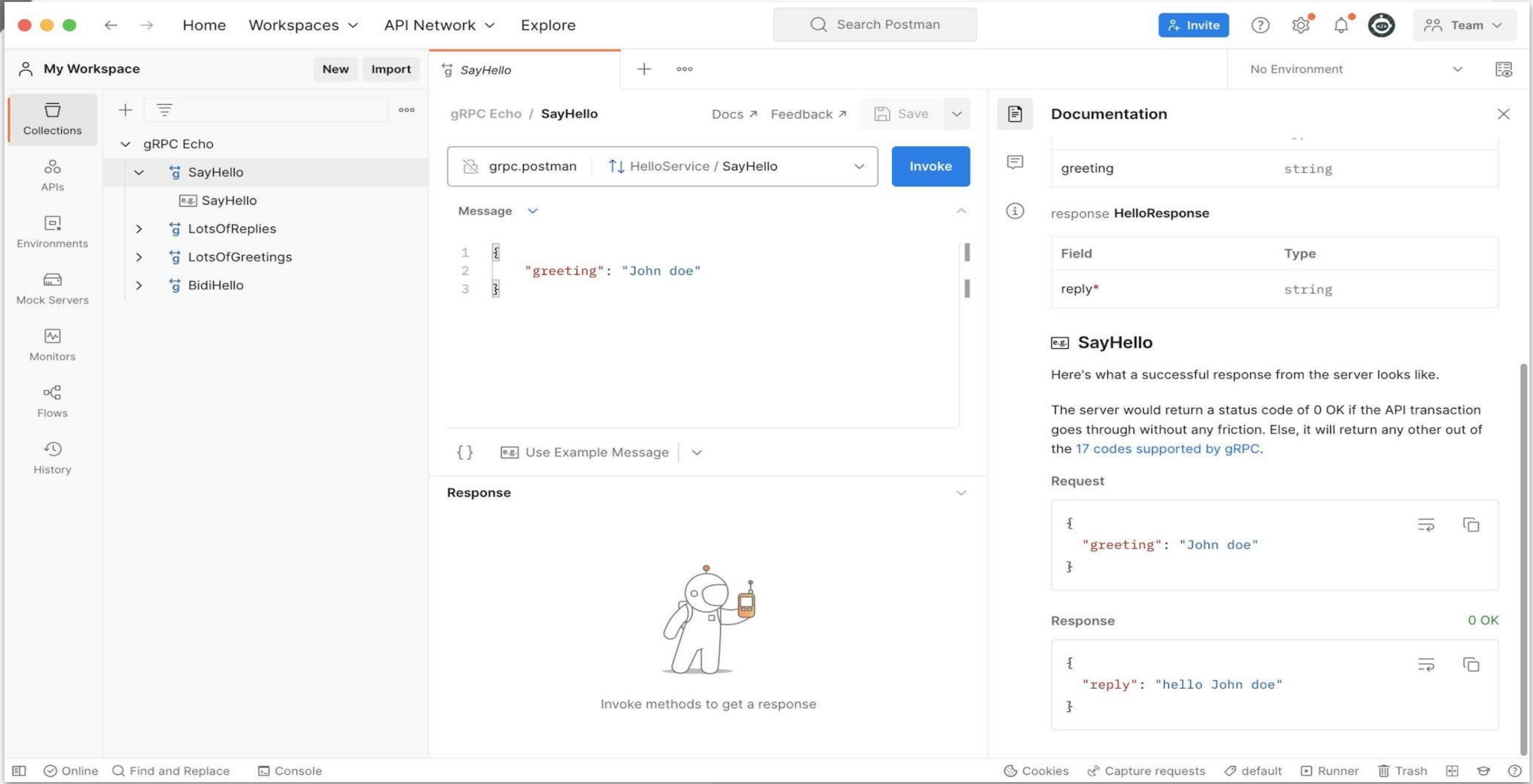
localhost:5000

```
{
  "name": "Hello"
}
```

```
{
  "message": "Hello Hello"
}
```

0.329s

Debugging tools: PostMan



```
{
    "greeting": "John doe"
}
```

Field	Type
greeting	string

response **HelloResponse**

Field	Type
reply*	string

SayHello

Here's what a successful response from the server looks like.

The server would return a status code of 0 OK if the API transaction goes through without any friction. Else, it will return any other out of the 17 codes supported by gRPC.

Request

```
{
  "greeting": "John doe"
}
```

Response 0 OK

```
{
  "reply": "hello John doe"
}
```

GRPC Client

pubspec.yaml:

```yaml
dependencies:
 grpc: ^3.1.0
 grpc_api:
   path: ../grpc-api
```

GRPC Client

```dart
import 'package:grpc/grpc.dart' ;
import 'package:helloworld/src/generated/helloworld.pbgrpc.dart' ;
Future<void> main(List<String> args)  async {
 final channel = ClientChannel(
   'localhost',
   port: 50051,
   options: ChannelOptions(
     credentials: ChannelCredentials.insecure(),
     codecRegistry:
         CodecRegistry(codecs:  const [GzipCodec(), IdentityCodec()]),
   ),
 );
 final stub = GreeterClient(channel);
 final name = args.isNotEmpty ? args[ 0] : 'world';
 try {
   final response = await stub.sayHello(
     HelloRequest()..name = name,
     options: CallOptions(metadata: {
       'custom-header-1' : 'value1',
       'custom-header-2' : 'value2',
     }),
   );
   print('Greeter client received:  ${response.message}');
 } catch (e) {
   print('Caught error:  $e');
 }
 await channel.shutdown();
}
```

GRPC SSL pinning

```
class ChannelCredentialsWithCertificatePinning  extends ChannelCredentials {
 ByteData certBytes;
 ChannelCredentialsWithCertificatePinning.secure( this.certBytes) : super.secure();

 @override
 SecurityContext get securityContext {
   final context = SecurityContext(withTrustedRoots:  false);
   context.setAlpnProtocols(supportedAlpnProtocols,  false);
   context.setTrustedCertificatesBytes(certBytes.buffer.asUint8List());
   return context;
 }
}

Future<void> main(List<String> args)  async {
 final channel = ClientChannel(
   'localhost',
   port: 50051,
   options: ChannelOptions(
    credentials:
      ChannelCredentialsWithCertificatePinning.secure( await
rootBundle.load('assets/grpc_certificate.pem' )),
    //credentials: const ChannelCredentials.secure(),
    //credentials: const ChannelCredentials.insecure(),
    codecRegistry: CodecRegistry(codecs:  const [GzipCodec(), IdentityCodec()]),
   ),
 );
…
```

GRPC Client

<DEMO>

GRPC on Web

- gRPC relies on trailing headers but browsers don't expose those.
- Fallback to http1.1 if needed
- Envoy proxy server with grpc_web filter enabled

Not supported:

Client-side and Bi-directional streaming

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

GRPC mocks

- Implement a grpc server with mocked data
- Mock your service clients with mocktail

```
class WeatherInfoServiceClientMock extends Mock implements WeatherInfoServiceClient {
 WeatherInfoServiceClientMock() {
   registerFallbackValue(GetCurrentWeatherInfoRequest());
 }
}
class MockResponseFuture<T> extends Mock implements ResponseFuture<T> {
 final T value;
 MockResponseFuture(this.value);
 Future<S> then<S>(FutureOr<S> onValue(T value), {Function? onError}) =>
     Future.value(value).then(onValue, onError: onError);
}
void main() {
   test('getCurrentWeatherInfo', () async {
     final client = WeatherInfoServiceClientMock();
     when(() => client.getCurrentWeatherInfo(any())).thenAnswer((_) => MockResponseFuture(
         GetCurrentWeatherInfoResponse(
             weatherInfo: WeatherInfo(message: 'Hello', temperature: 20.1, isRainy: false, isNight:
true))));
     final response = await client.getCurrentWeatherInfo(GetCurrentWeatherInfoRequest());
     expect(response.weatherInfo.isRainy, false);
   });
}
```

GRPC links

https://github.com/krispypen/meetup_demo_grpc

https://pub.dev/packages/grpc

https://github.com/grpc/grpc-web/blob/master/net/grpc/gateway/examples/echo/envoy.yaml

https://github.com/grpc-ecosystem/awesome-grpc

https://blog.postman.com/grpc-vs-rest/

```
await question?.ask();
```