Flutter Communication gRPC

Peter Kassenaar –
info@kassenaar.com

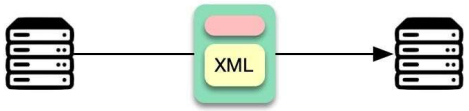
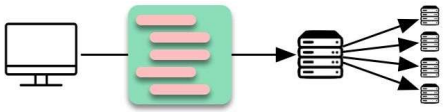
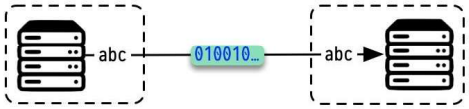
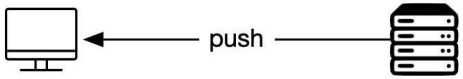
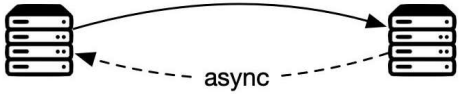
Communicating via gRPC

A possible way of communication inside your app with a backend

Communication – lots of options!

API Architecture Styles

ByteByteGo.com

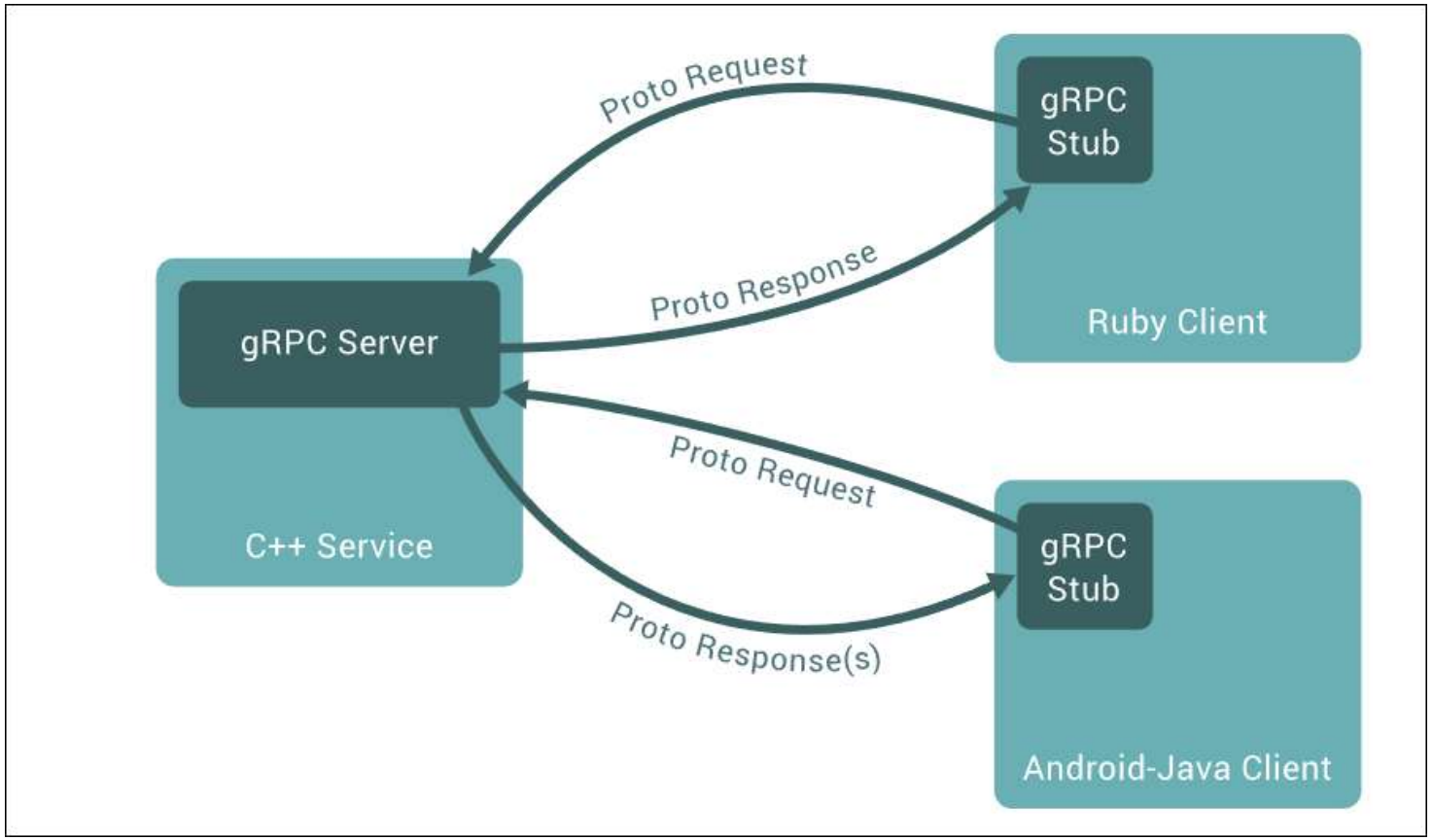
Style	Illustration	Use Cases
SOAP		XML-based for enterprise applications
RESTful		Resource-based for web servers
GraphQL		Query language reduce network load
gRPC		High performance for microservices
WebSocket		Bi-directional for low-latency data exchange
Webhook		Asynchronous for event-driven application

gRPC

What is gRPC?

- gRPC is 'just' one of the possible forms of communication
  - It is an implementation detail.
  - gRPC CAN (in theory) be swapped out for other types of communication below the bloc archtecture
- *"A high performance, open source universal RPC framework"*
- https://grpc.io/
- Basically a wrapper around `http`, using `http/2` protocol
- Binary format (as opposed to `json` and `xml`)

Introduction to gRPC



https://grpc.io/docs/what-is-grpc/introduction/

“gRPC in 5 minutes”

https://www.youtube.com/watch?v=njC24ts24Pg

gRPC Prerequisites

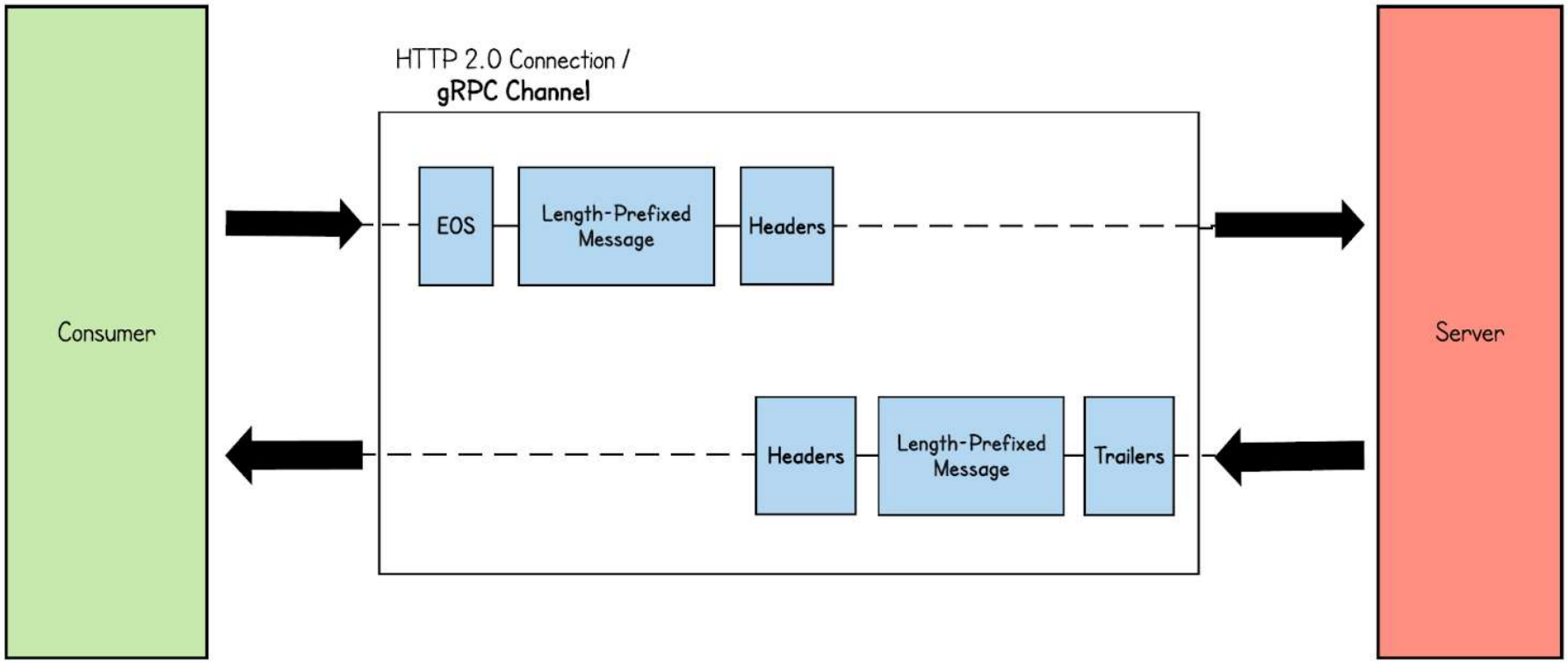
- gRPC has multiple elements:
- SERVER – server responds to gRPC calls and sends data
- CLIENT – your app. Uses gRPC services to communicate with the backend
- Dart – as the programming language
- Protocol buffer compiler `protoc` (‘protobuf’)
- Dart plug-in for the compiler `protoc-gen-dart`

Channels

- Client communicates with Server via a Channel

```
final channel = ClientChannel(
  'localhost', // Replace with server hostname or IP
  port: 50051,
  options: const ChannelOptions(
    credentials: ChannelCredentials.insecure(),
  ),
);
```

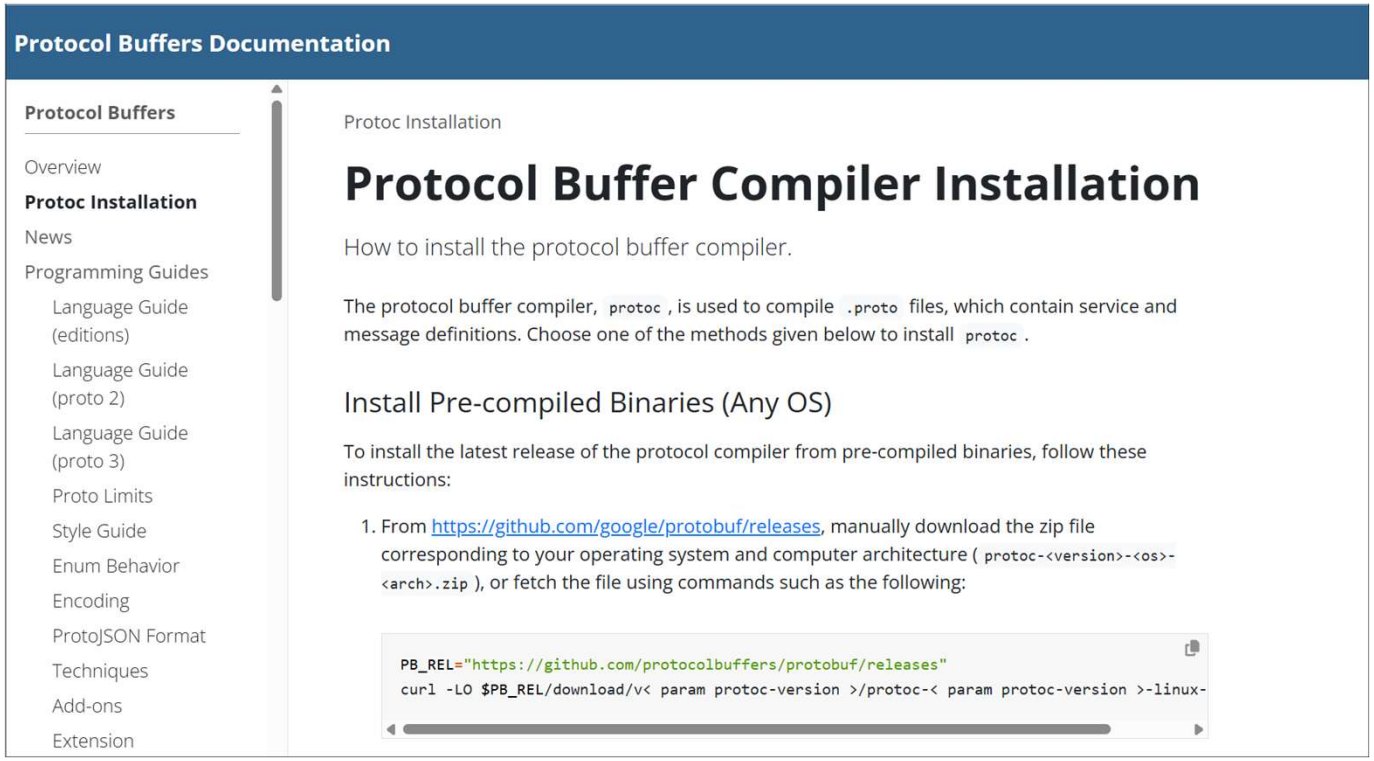
More on Channels



https://thenewstack.io/grpc-a-deep-dive-into-the-communication-pattern/

Protoc - compiler

- Needed to compile `.proto` files in your app
- Steps: protobuf.dev/installation/

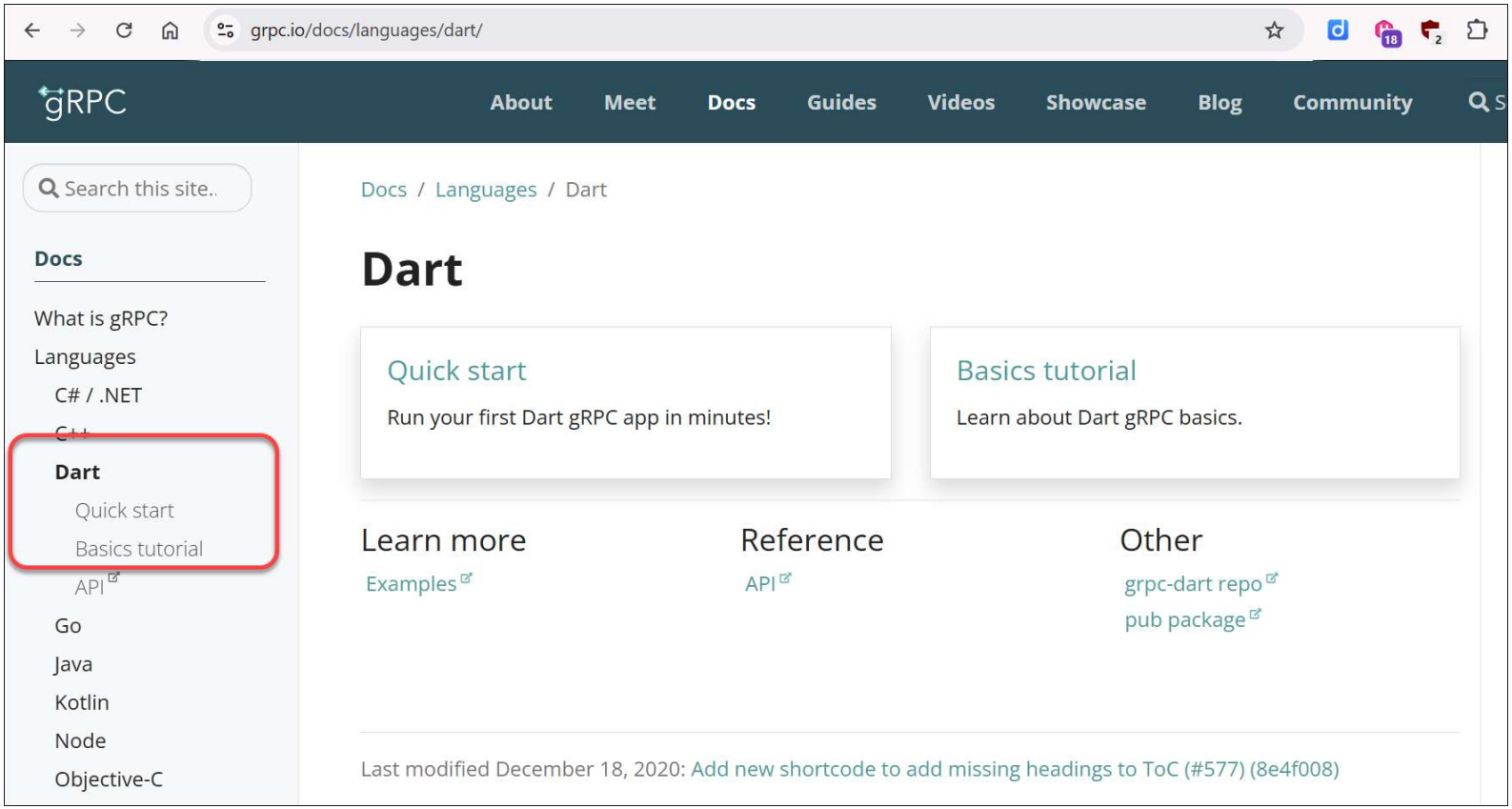


Example – the Hello World demo

Getting started with gRPC Dart server & Flutter client

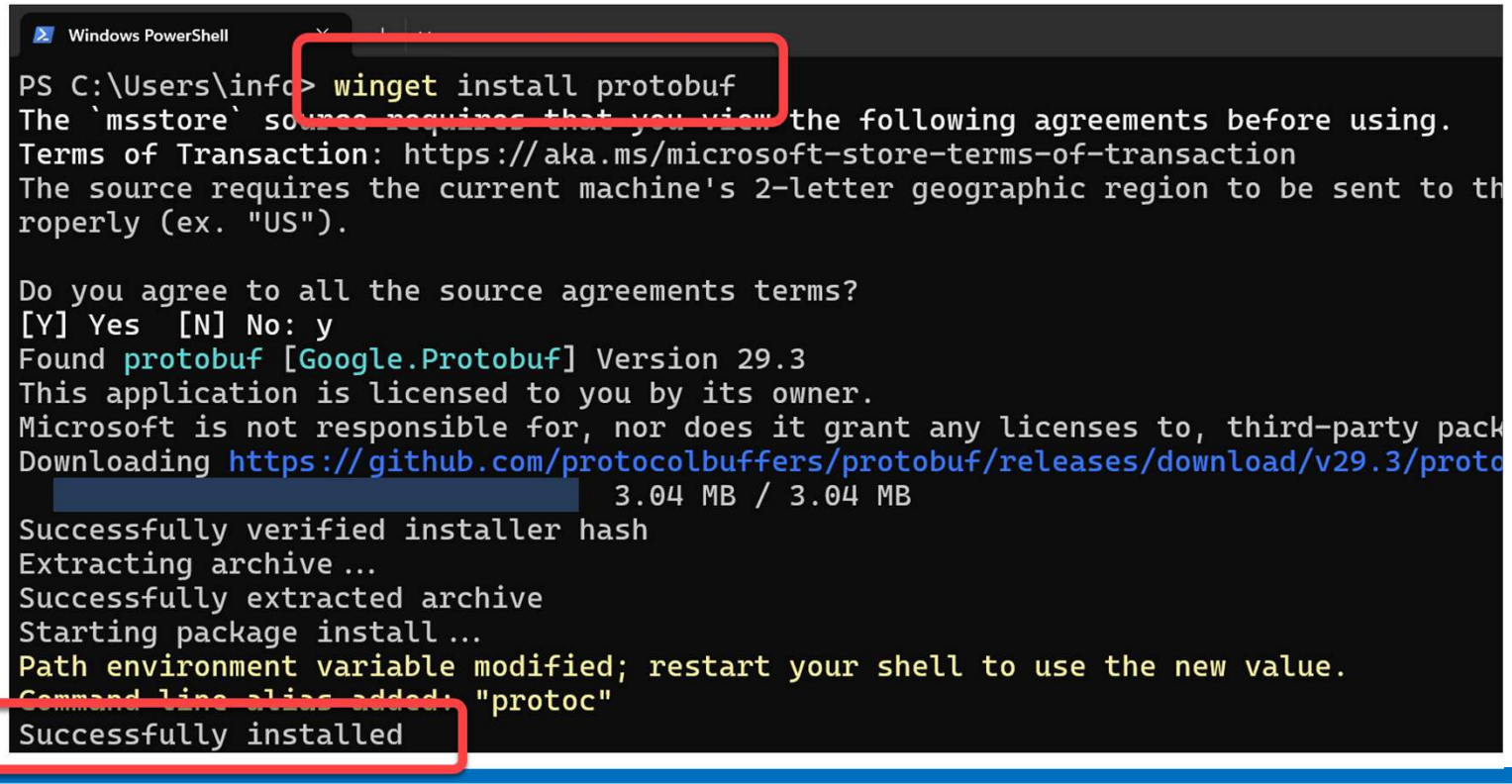
Quick Start online (warning – going fast!)

- https://grpc.io/docs/languages/dart/



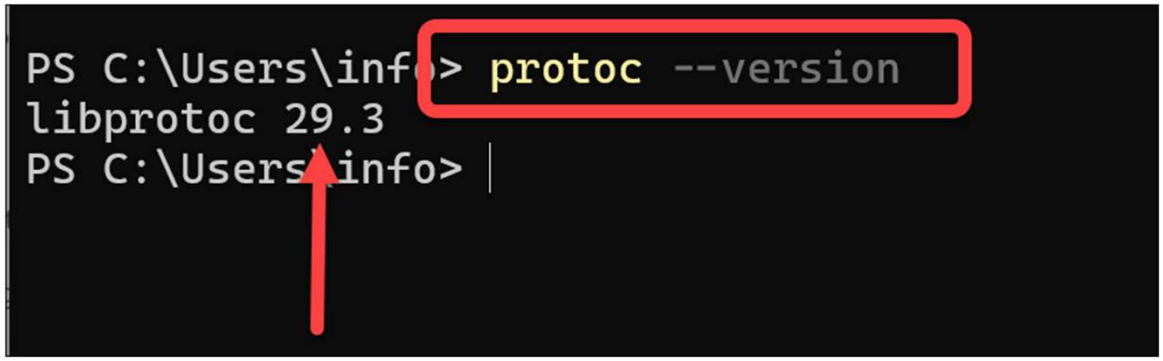
Installing Protobuf on windows

- `winget install protobuf`
- Other OS's: see protobuf.dev/installation/



Check installed version

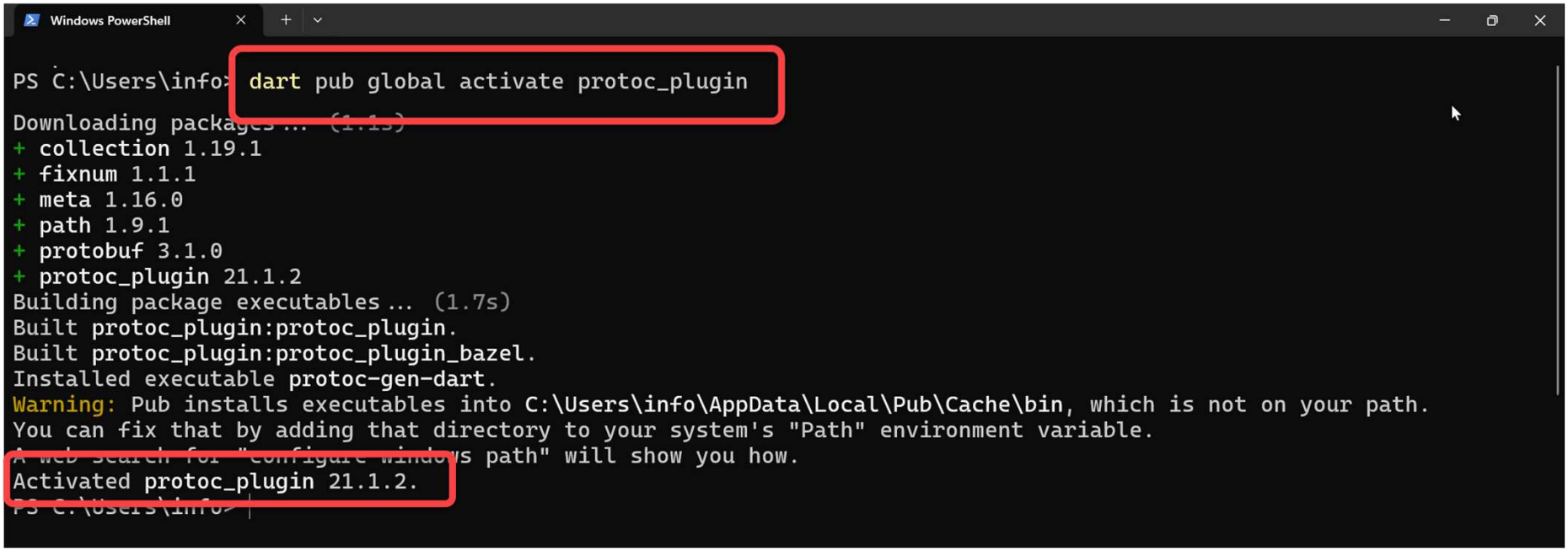
- In a new terminal, check installation
- `protoc --version`



> Protocol buffer compiler (**protoc**) — *is a compiler for protocol buffers definitions files(.proto files). It can generate C++, Java,golang,dart and Python source code for the classes defined in .proto file.*

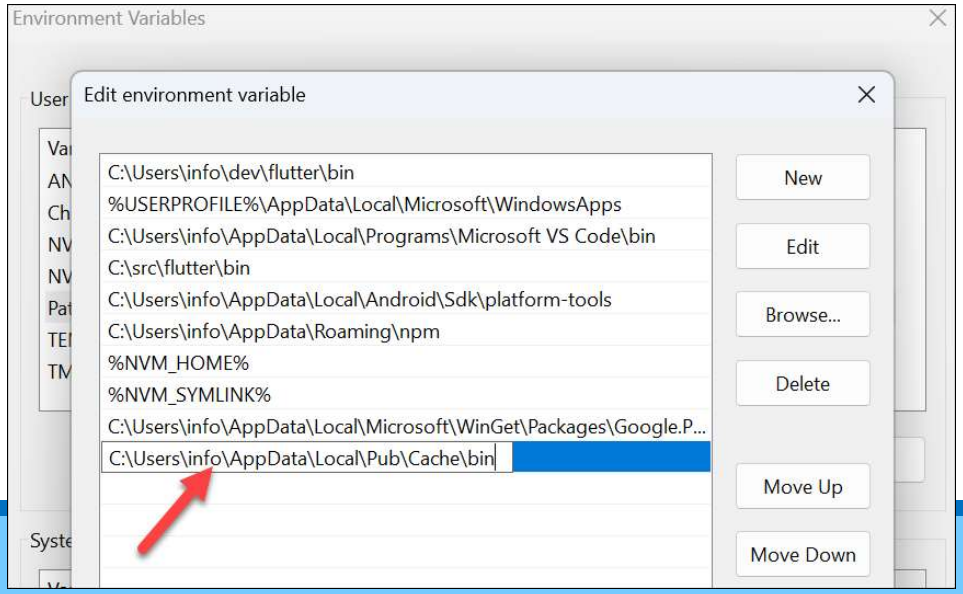
Install dart compiler plugin

- Install the protocol compiler plugin for Dart (`protoc-gen-dart`)
- `dart pub global activate protoc_plugin`



Update your path

- The plugin must be in your path!
- Add `C:\Users\<name>\AppData\Local\Pub\Cache\bin` to the user `PATH` variable
  - Or, update for YOUR path. See terminal for details.
- It generates Dart files for working with data in protocol buffers format

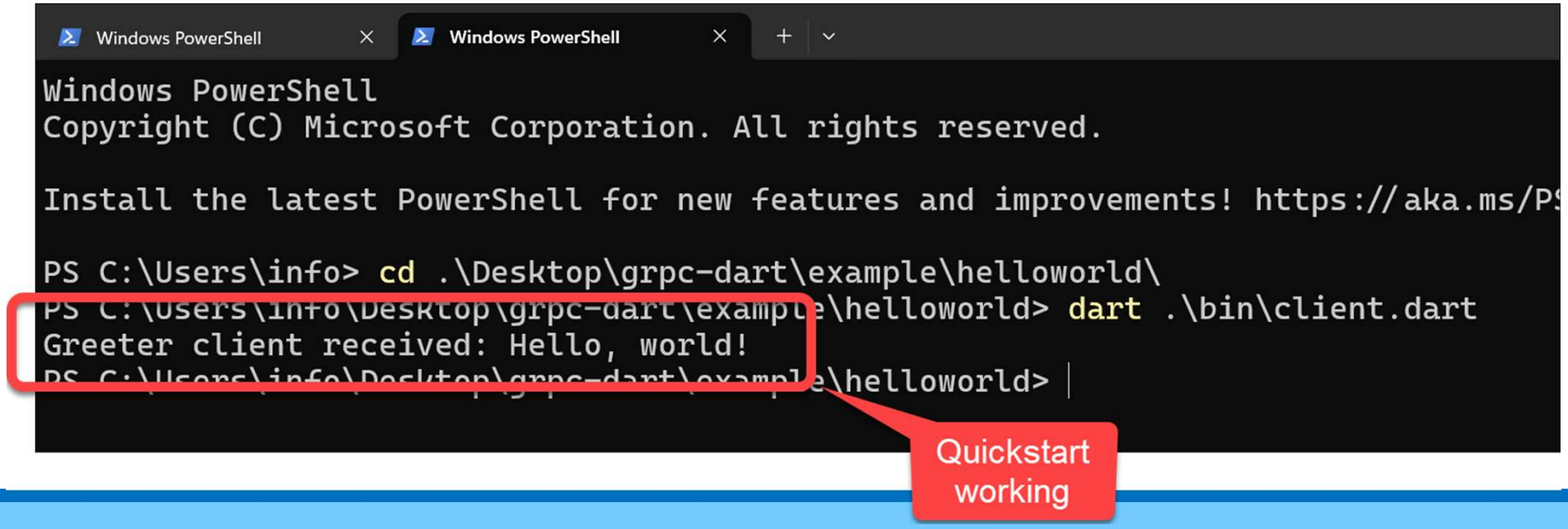


More info on the dart plugin

https://pub.dev/packages/protoc_plugin

Using the QuickStart App

- Follow the guidelines on grpc.io/docs/languages/dart/quickstart/ to start a text based Dart-version of server and client.
- Result:



However, we want a Flutter implementation

- Create a new, basic Flutter app, or use existing app
  - Add grpc package: `flutter pub add grpc`
  - Add protobuf package: `flutter pub add protobuf`

```
flutter pub add grpc
Resolving dependencies...
Downloading packages... (2.1s)
+ args 2.7.0
  async 2.12.0 (2.13.0 available)
+ crypto 3.0.6
  fake_async 1.3.2 (1.3.3 available)
+ fixnum 1.1.1
+ google_identity_services_web 0.3.3
+ googleapis_auth 1.6.0
+ grpc 4.0.1
+ http2 2.3.1
  leak_tracker 10.0.
  material_color_util
+ protobuf 3.1.0
  vm_service 14.3.1 (
Changed 8 dependencie
5 packages have newer
Try `flutter pub outd
```

```
flutter pub add protobuf
Resolving dependencies...
Downloading packages...
  async 2.12.0 (2.13.0 available)
  fake_async 1.3.2 (1.3.3 available)
  leak_tracker 10.0.8 (10.0.9 available)
  material_color_utilities 0.11.1 (0.12.0 available)
  protobuf 3.1.0 (from transitive dependency to direct dependency)
  vm_service 14.3.1 (15.0.0 available)
```

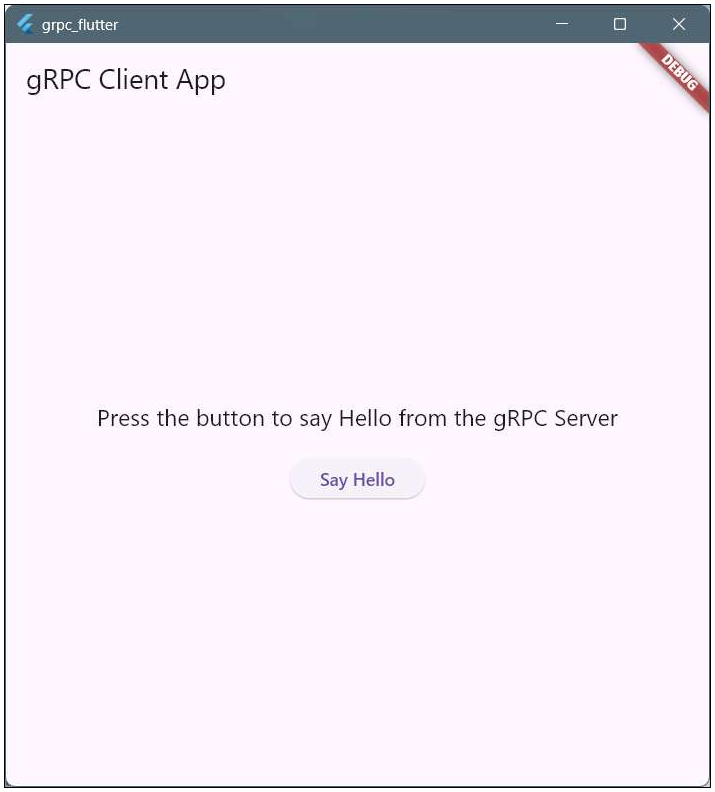
Our example: `../examples/_550-gRPC-demo`

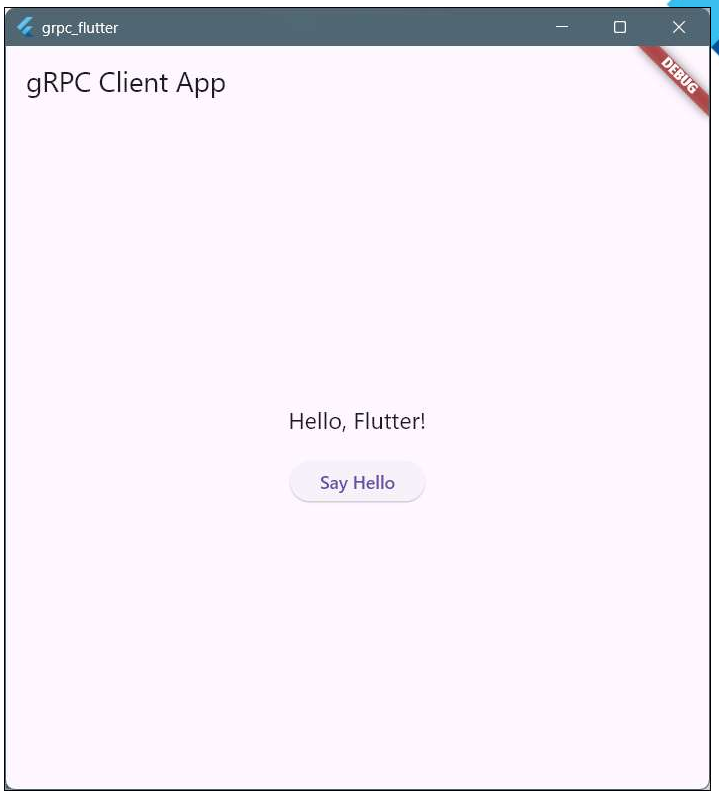
- Open the application
- In a terminal, run the server
  - `dart run ./server/server.dart`
- In another terminal, run the client
  - `flutter run`
- See `main.dart` for the homepage code

```dart
String _responseMessage = "Press the button to say Hello from the gRPC S

Future<void> _callGrpcService() async {   PeterKassenaar, Today • added
  final channel = ClientChannel(
    'localhost', // Replace with your server's hostname or IP
    port: 50051,
    options: const ChannelOptions(
      credentials: ChannelCredentials.insecure(),
    ),  // ChannelOptions
  );  // ClientChannel
```

Result


grpc_flutter
gRPC Client App
DEBUG
Press the button to say Hello from the gRPC Server
Say Hello


grpc_flutter
gRPC Client App
DEBUG
Hello, Flutter!
Say Hello

Optional: updating the app

- If you want to do something different, you have to
  - 1. update the server
  - 2. update the app
- For instance, in `helloworld.proto`, add a method

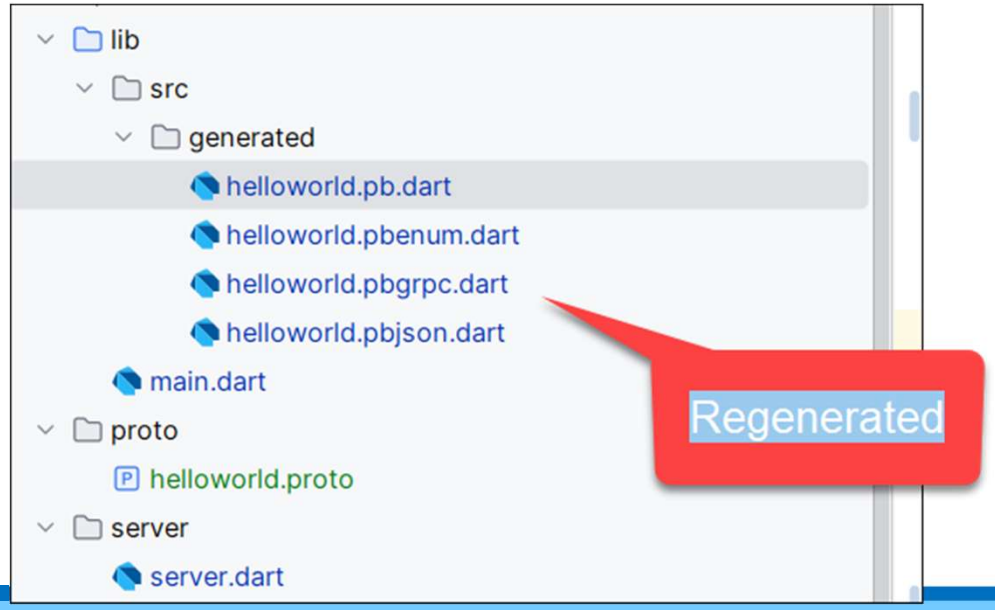
```
service Greeter {
  // …
  // We now define a second method, that returns another greeting
  rpc SayHelloAagain(HelloRequest) returns (HelloReply){  }
}
```

See `helloworld.proto` for other updates!

Regenerate gRPC Code

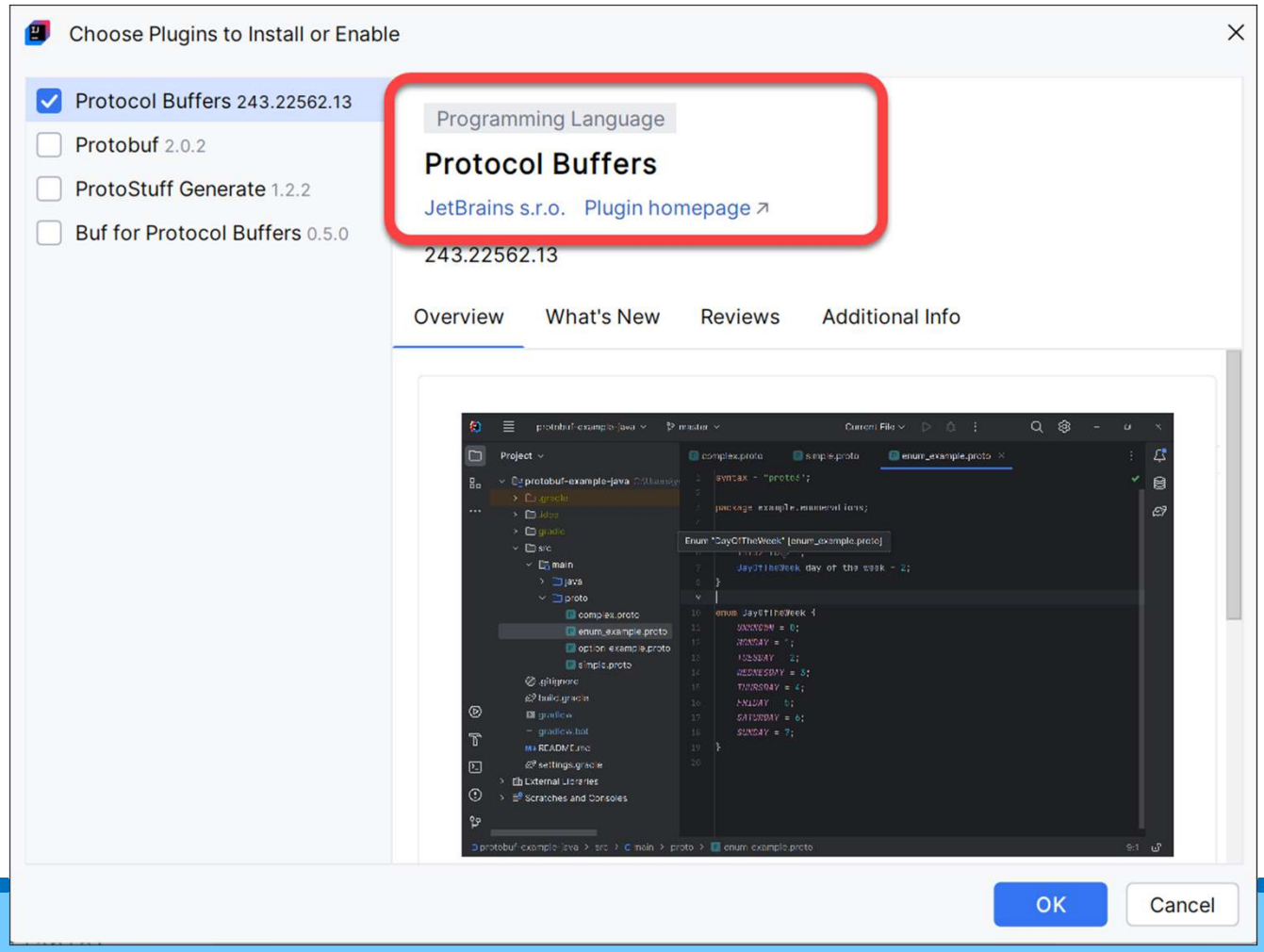
- Before we can use the new service method, we have to recompile the updated .proto file
- From the `/proto` directory, run

```
protoc helloworld.proto --dart_out=grpc:../lib/src/generated
```



Optional: `.proto` plugin

- IntelliJ can work with `.proto` files via a plugin



Update `server.dart`

- Update `server.dart` so the new method `sayHelloAgain()` can be called:

```
/// Dart implementation of the gRPC helloworld.Greeter server.
class GreeterService extends GreeterServiceBase {
  …
  @override
  Future<HelloReply> sayHelloAagain(ServiceCall call, HelloRequest request,)
  async {
    return HelloReply()..message = 'Hello again, ${request.name}!';
  }
}
```

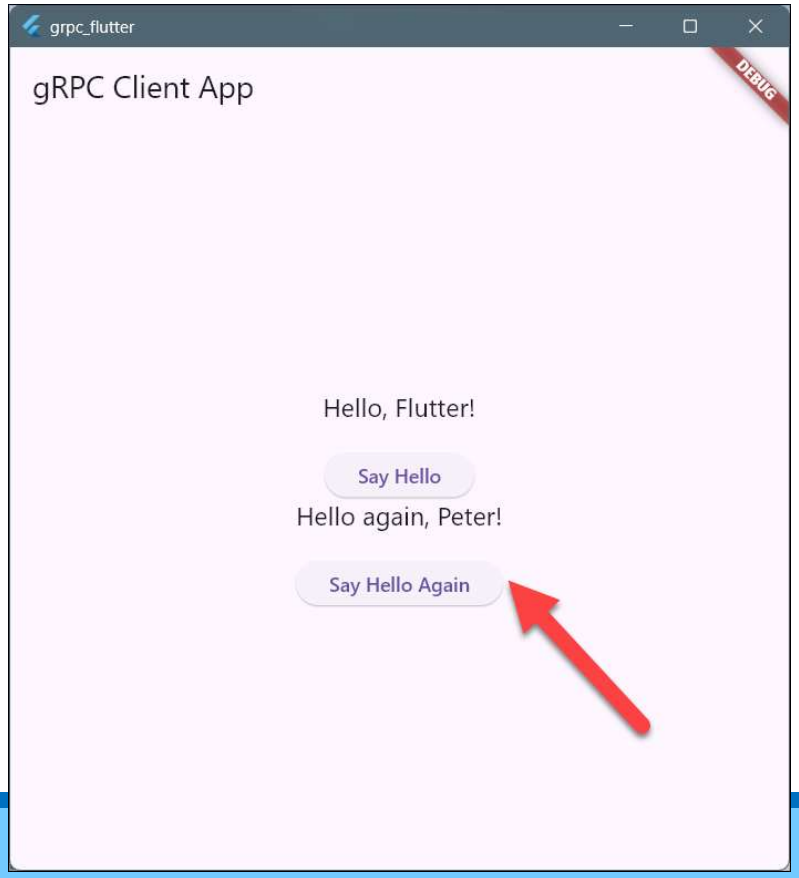
Update `main.dart`

- Update `main.dart` so the new method is called
  - This one is simply duplicating the string. Of course you can adapt it to meet your needs.

```
…
final response2 = await stub.sayHelloAagain(
  HelloRequest()..name = 'Peter’,
)
…
// Create UI to call and show _response2…
```

Restart the server

- After updating and recompiling the server, ALWAYS stop a possible running instance (`Ctrl+C`) and restart the server
  - `dart run server.dart`
- Possible, simple UI:



Workshop

- Study the example on how to use gRPC server and client files
  - `../examples/_550-gRPC-demo`
- Optional: follow the Optional steps in this presentation to activate an extra method on the server
  - Extract `ClientChannel()` creation to its own function
- Optional: Create a new method on the server, recompile and test.
- Optional: Create a completely new Flutter app, implementing gRPC Server and Client and get it working.
- Documentation
  - https://grpc.io/docs/languages/dart/quickstart/
  - https://medium.com/@itachisasuke/using-grpc-with-flutter-a-step-