Quick start

This guide gets you started with gRPC in Go with a simple working example.

Prerequisites

- **Go**, any one of the **three latest major** releases of Go.

  For installation instructions, see Go's Getting Started guide.

- **Protocol buffer compiler**, `protoc`, version 3.

  For installation instructions, see Protocol Buffer Compiler Installation.

- **Go plugins** for the protocol compiler:

  1. Install the protocol compiler plugins for Go using the following commands:

```
$ go install google.golang.org/protobuf/cmd/protoc-gen-go@v1.28
$ go install google.golang.org/grpc/cmd/protoc-gen-go-grpc@v1.2
```

  2. Update your `PATH` so that the `protoc` compiler can find the plugins:

```
$ export PATH="$PATH:$(go env GOPATH)/bin"
```

Get the example code

The example code is part of the grpc-go repo.

1. Download the repo as a zip file and unzip it, or clone the repo:

```
$ git clone -b v1.52.0 --depth 1 https://github.com/grpc/grpc-go
```

2. Change to the quick start example directory:

```
$ cd grpc-go/examples/helloworld
```

Run the example

From the `examples/helloworld` directory:

1. Compile and execute the server code:

```
$ go run greeter_server/main.go
```

2. From a different terminal, compile and execute the client code to see the client output:

```
$ go run greeter_client/main.go
Greeting: Hello world
```

Congratulations! You've just run a client-server application with gRPC.

Update the gRPC service

In this section you'll update the application with an extra server method. The gRPC service is defined using protocol buffers. To learn more about how to define a service in a `.proto` file see Basics tutorial. For now, all you need to know is that both the server and the client stub have a `SayHello()` RPC method that takes a `HelloRequest` parameter from the client and returns a `HelloReply` from the server, and that the method is defined like this:

```
// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

Open `helloworld/helloworld.proto` and add a new `SayHelloAgain()` method, with the same request and response types:

```
// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
  // Sends another greeting
  rpc SayHelloAgain (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

Remember to save the file!

Regenerate gRPC code

Before you can use the new service method, you need to recompile the updated `.proto` file.

While still in the `examples/helloworld` directory, run the following command:

```
$ protoc --go_out=. --go_opt=paths=source_relative \
    --go-grpc_out=. --go-grpc_opt=paths=source_relative \
    helloworld/helloworld.proto
```

This will regenerate the `helloworld/helloworld.pb.go` and `helloworld/helloworld_grpc.pb.go` files, which contain:

- Code for populating, serializing, and retrieving `HelloRequest` and `HelloReply` message types.
- Generated client and server code.

Update and run the application

You have regenerated server and client code, but you still need to implement and call the new method in the human-written parts of the example application.

Update the server

Open `greeter_server/main.go` and add the following function to it:

```
func (s *server) SayHelloAgain(ctx context.Context, in *pb.HelloRequest) (*pb.HelloR
        return &pb.HelloReply{Message: "Hello again " + in.GetName()}, nil
}
```

Update the client

Open `greeter_client/main.go` to add the following code to the end of the `main()` function body:

```
r, err = c.SayHelloAgain(ctx, &pb.HelloRequest{Name: *name})
if err != nil {
        log.Fatalf("could not greet: %v", err)
}
log.Printf("Greeting: %s", r.GetMessage())
```

Remember to save your changes.

Run!

Run the client and server like you did before. Execute the following commands from the `examples/helloworld` directory:

1. Run the server:

   ```
   $ go run greeter_server/main.go
   ```

2. From another terminal, run the client. This time, add a name as a command-line argument:

   ```
   $ go run greeter_client/main.go --name=Alice
   ```

   You'll see the following output:

```
Greeting: Hello Alice
Greeting: Hello again Alice
```

What's next

- Learn how gRPC works in Introduction to gRPC and Core concepts.
- Work through the Basics tutorial.
- Explore the API reference.

Last modified April 8, 2022: Update Go quickstart to suggest using latest version (#964) (ba84d86)

View page source | Edit this page | Create child page | Create documentation issue | Create project issue