Building .NET Services Using gRPC

Alex Ryazhnov

gRPC and Protocol Buffers

An Introduction

Distributed Computing API Options

- API Architectures:
  - SOAP
  - REST – Representation State Transfer
  - GraphQL
  - gRPC

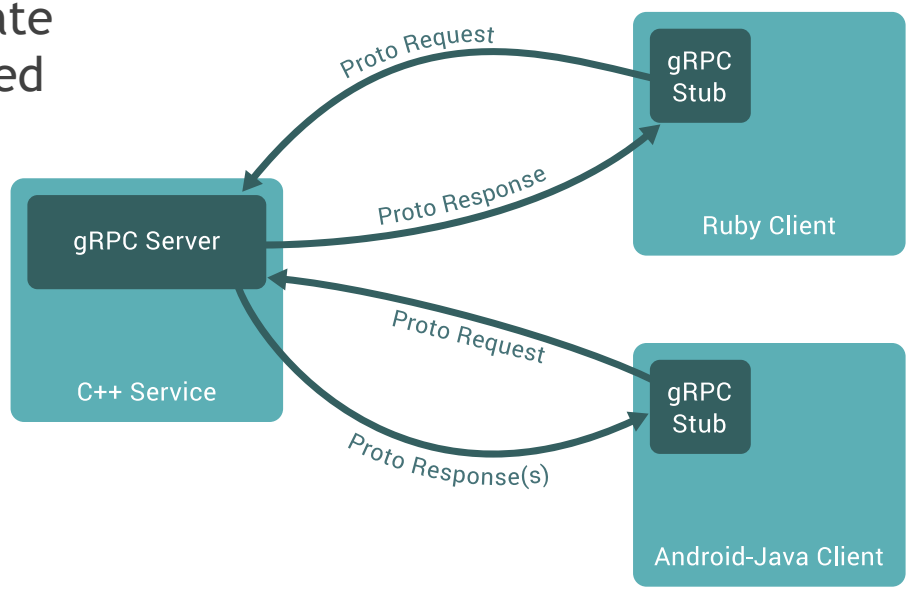
What is gRPC?

- gRPC – grpc Remote Procedure Call
- grpc.io: "*gRPC is a modern open source high performance Remote Procedure Call (RPC) framework that can run in any environment.*"
- Released from Google in 2016
- Used by Square, Netflix, Cisco, Juniper Networks
- Contract-based
- Uses HTTP/2 for transport. Protocol Buffers as the interface definition language
- Binary
- Streaming support

What is gRPC? (continued)

- Service is defined using Protocol Buffers (.proto file)
- gRPC uses protocol buffer compiler (protoc) to generate classes in supported language

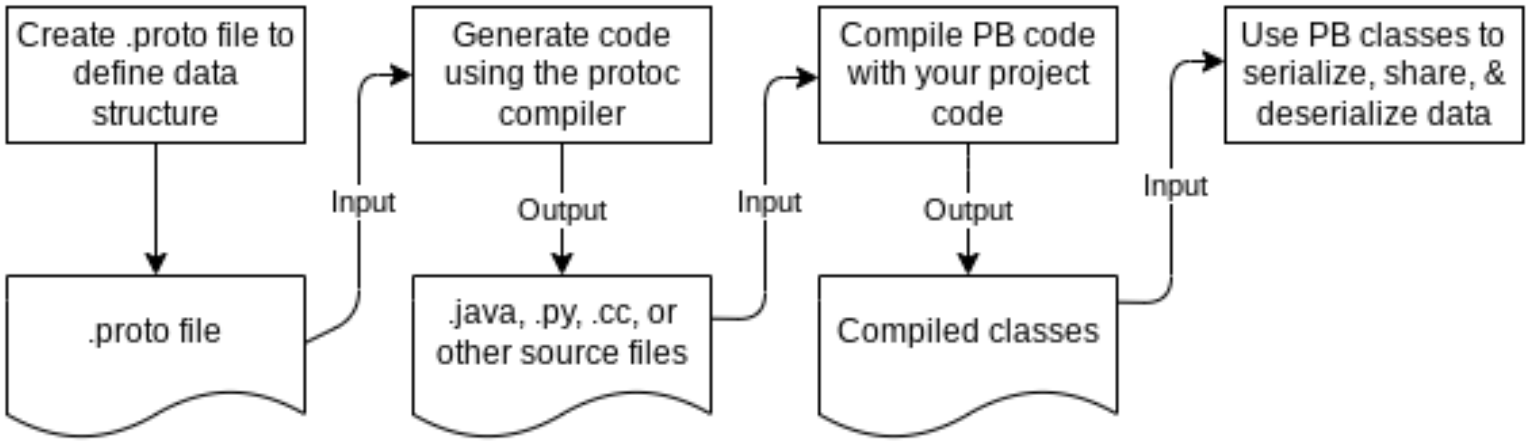
```
// The greeter service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

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

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



Protocol Buffers

- IDL – Interface Definition Language
- Released by Google in 2008
- Language independent
- Fast and efficient
- Key features:
  - Uses binary format
  - Separates context and data

Protocol Buffers Workflow



- Protocol Buffers documentation:
  - https://developers.google.com/protocol-buffers/docs/overview

Demo Plan

- Employee REST API .NET Core ---> gRPC Server in .NET CORE
- gRPC Client to consume Employee API
- gRPC Streaming endpoints
- gRPC Middleware and Interceptors
- gRPC Security

Creating gRPC Server

- Define Service interface and payload messages in .proto file
- Generate gRPC Server infrastructure code and models using protoc compiler
  - Grpc.Tools package
- Implement gRPC Service in .NET Core
  - Override generated gRPC Service Base methods
  - Register gRPC services using AddGrpc() extension method
  - Use MapGrpcService on endpoints to register implemented gRPC Service
  - Configure Kestrel with HTTP/2

Creating gRPC Client

- Use protoc compiler to generate Client code
  - Can use Add Service Reference File or URL option to find .proto file
- Use GrpcChannel.ForAddress to create channel
  - Channel represents a long-lived connection to a gRPC Service
  - Can be expensive. Should reuse channel for gRPC calls
- Create client that uses the channel to make call(s)

gRPC Streaming

- Server
  - Client sends a request message
  - Server returns a stream of messages
- Client
  - Client sends a stream of messages
  - Server responds with a single message
  - Client should call CompleteAsync() to notify service of stream completion
- Duplex (Bi-directional)
  - Client and Server streaming messages to each other.
  - Streams are independent. Messages can be read/written in any order

Middleware & gRPC Interceptors

- Middleware runs for all HTTP requests including gRPC
- Middleware does not have access to deserialized gRPC message
  - Can only access bytes from the request and response streams
- Interceptor is gRPC concept that operates on gRPC layer of abstraction
  - Has access to deserialized message
- Interceptors run after middleware
- Interceptors can be configured for both client and server
- Multiple interceptors can be chained to construct pipeline

Securing gRPC Service

- Use ASP.NET Authentication/Authorization
- Can decorate gRPC Service class or methods with Authorize attribute
- On client side can use Metadata on individual call or interceptors for attaching authentication information

gRPC vs REST

Summary and wrap-up

gRPC vs REST differences

gRPC

- Contract required (.proto)
- Action based
- Tight Coupling
- HTTP 2
- Protocol Buffers
- Unary Request-Response or streaming
- Native code generation

REST

- Optional contract (OpenAPI)
- Resource based
- Loose Coupling
- Typically HTTP 1.1
- JSON
- Request-Response model
- Third party tools

gRPC Strengths and Usage Scenarios

- gRPC strengths:
  - Performance: HTTP/2 over HTTP 1.x; small message payloads; efficient protobuf serialization
  - Native code generation in multiple languages
  - First-class streaming support
- gRPC usage scenarios:
  - Microservices: service-to-service communication
  - Limited bandwidth scenarios: Mobile apps will benefit from small message payload
  - Multi-language environments: ability to generate server/client code in multiple languages

gRPC Limitations

- gRPC limitations:
  - Messages are not human readable
  - Limited browser support: requires gRPC-Web and proxy layer
- REST API is a better choice for browser apps support

- Contact information:
  - Email: alex.ryazhnov@gmail.com
  - LinkedIn: https://www.linkedin.com/in/alex-ryazhnov-98445239/
- Code will be available at:
  - https://github.com/ryazan05/MDC2022-GrpcDotNetDemo

Thank you!