Week 3 Lesson: gRPC and Protobuf

Objectives

Learn about gRPC, the protocol that Go microservices use to communicate.
By the end of this lesson, you can create a server and client that communicate via gRPC using protocol buffers, which lets you define the structure of a message.

Prerequisites

- Basic understanding of Go syntax
- Ability to create, define, and use Go packages and structs
- Basic understanding of server-client communications

Session 1

Introduction to gRPC

Note: While the other sessions are more or less Go specific, gRPC can be applied to many other popular languages like Java, Python, Node, Ruby, etc.

What is RPC?

A Remote Procedure Call (RPC) is a function call with execution details that are transparent to the callers. The actual code is being executed on remote machines via TCP/UDP connections, a process in the same machine, or a function in the same process.

Where do we need RPC?

RPC allows two parties to communicate. This can be between an end user and a server, or between servers.
With the proliferation of microservice architecture, we need to choose the way how servers talk to each other.

What are the alternatives to RPC?

Communication is not a new problem we want to solve. Let's look at the other ways.

Protocol Type/Format	**Syntax**	**Advantages**	**Disadvantages**
Raw TCP	Varies depending on the defined protocol	Allows you to define your own protocol to be business optimized	Much more effort
HTTP: HTML Forms	a=b&c=d	Browser support	Lack of rich structure and type
HTTP: XML	<Root><Item><Key>a</Key><Value>b</Value></Item></Root	Well-structured and contains types	Verbose
HTTP: JSON	{"a": "b", "c": "d"}	Simpler than XML, easier to read	N/A

-

What is gRPC?

gRPC is defined as follows:

- An implementation of RPC initially developed by Google.
- Uses Protocol Buffers to define the message structure.
- Uses HTTP/2.
- Contains best practices features like timeout, authentication, etc.

What is Protocol Buffers?

Protocol Buffers (also known as protobuf) provides a method for serializing structured data.
Here is an example of a proto file:

```
service UserService {
 rpc GetUserList(GetUserListRequest) returns (GetUserListResponse) {
   option (google.api.http) = {
     get: "/users"
   };
 }
}

message GetUserListRequest {
 int32 limit = 1;
}

message GetUserListResponse {
 repeated User users = 1;
}

message User {
 int64 ID = 1;
 string name = 2;
}
```

This file can be used to generate both gRPC and HTTP clients.

gPRC and JSON-based RESTful Comparison

Advantages of gRPC

- Developer Friendly

    - Code generation
    - API description using Protocol Buffers
- Performance
    - Efficiency in data transfer
    - Leverages HTTP/2

Note: Using OpenAPI toolkit (swagger), RESTful API also has some of the benefits

Disadvantages of gRPC

- Limited language support. New or unpopular languages may not be supported officially, like Rust, Lua, Kotlin.
- Debugging is not as straightforward because it's not in plaintext.

Demo

Create a hello world server and client in gRPC from scratch. The contents are from https://grpc.io/docs/quickstart/go/.
Screen recording: https://asciinema.org/a/FWltPaz1zpWvzsjIijBB5TQ77

Exercise

Follow the guide at https://grpc.io/docs/quickstart/go/ to create a gRPC server and client, and run the client to call the server to get responses.

For those who are interested in this topic, we can use reflection to have a similar CURL experience for gRPC. This is optional, but may be worth mentioning as it improves development experience.

Reference

https://developers.google.com/protocol-buffers/docs/overview
https://medium.com/@bimeshde/grpc-vs-rest-performance-simplified-fd35d01bbd4
https://swagger.io/tools/swagger-codegen/