

To what extend is it possible to build an efficient and secure Go gRPC API without previous experience?

Abstract

Learning how to write a gRPC API in Go with no experience in both subjects is difficult. The paper discusses the difficulties of learning how to write a gRPC API in Go with no experience and compares it to REST API also written in Go. The paper consists of research to difficulties learning Golang, in general and specific to the gRPC API model, and the challenges to gRPC APIs. It obviously contains research to the combination of these subjects too. Does Golang have the tools and necessary support to be able to develop a gRPC API? Is there sufficient documentation and learning material to know how to start the development of a gRPC API? How do the benefits and struggles compare to the development of a REST API in Golang? These are some of the topics that will be discussed in the paper.

Introduction

Let's start with an explanation about the topics at hand, the Go programming language also known as Golang and the gRPC model for APIs, with some of the points they are most known for.

Go is a programming language designed with simplicity and efficiency in mind. Some of the features that make Go stand out:

- Fast compilation times: Go compiles quickly, which makes it a good choice for projects that require fast build times.
- Concurrency support: Go's built-in concurrency features, such as goroutines and channels, make it easy to write concurrent code that can take advantage of modern multi-core processors.
- Easy to learn and use: Go has a clean and simple syntax, which makes it easy to learn and use. It also comes with a comprehensive standard library that provides a wide range of functionality out of the box.
- Garbage collection: Go has a built-in garbage collector that automatically reclaims memory that is no longer in use, which simplifies memory management.
- Cross-platform compatibility: Go can be compiled to run on multiple platforms, including Linux, macOS, and Windows.

gRPC is a high-performance, open-source RPC (Remote Procedure Call) framework. Some of the features that make gRPC APIs special include:

- High performance: gRPC is designed to be fast and efficient, with low overhead. It uses HTTP/2 as the underlying transport protocol, which allows it to take advantage of features such as multiplexing and streaming.
- Strong typing: gRPC uses Protocol Buffers for serialization, which provides strong typing and backwards compatibility.
- Easy to use: gRPC provides a simple API for defining and calling services, with support for streaming and bidirectional communication.
- Well-documented: gRPC has excellent documentation and a large, active community of users, making it easy to get started and get help when needed.

How was the research performed

This research was done by someone that has no experience in either subject. The person in question, me, will build a gRPC API from scratch in Golang. Then the difficulties, discoveries, improvements that could be made and recommendations will be explained before coming to a conclusion.

Learning material

There is plenty learning material available for this topic. This includes Go, gRPC and gRPC in Golang. Go is quite popular and has an active and helpful community. Furthermore, Go has excellent documentation and a lot of helpful tools and packages also include ample documentation on when and how to use it. While it seems that the community for gRPC is somewhat lacking at times there is still a lot of material and documentation you can find on internet.

The most astonishing amount of learning material available is specifically for the combination of the two topics. Go has official documentation on how to use gRPC and gRPC has official documentation on how to use it with Go. Not only that, there are also a plethora of tutorials available on gRPC APIs in Go. The most lacking information lies in some of the finer parts of gRPC and mainly how to use a gRPC API with some frontends like next.js for example, which is known to have very little documentation.

Learning the Go Language

The Go programming language is the base of the project, so let's begin with the analysis of Golang.

As a beginner going into writing Go blindly may give some trouble regarding go modules and package dependencies. However, Go has a set of commands to initialize a Go module and resolve package downloads and dependencies that are really easy to use. This makes it so that resolving this trouble is only a few google searches away and only makes it a minor issue.

Programming in Go is surprisingly intuitive in almost every way. An important part of Golang is handling pointers, but this is not a problem. Pointers are decently hard to master, but it is really easy to learn it to a usable and functional level in Go. Golang makes the learning process really easy, even with no experience in handling pointers in any programming language. This was my experience for most of the syntax in Go. Take the unique use of "!=" in Go for example. It allows for a smooth creation and of a variable and assigning a value to it with just 2 characters. Another great benefit of Golang is that functions can return multiple values, which can simplify development a lot. In case there is no use for a returned value it can just be discarded it by assigning it to "_". For loops also make use of this technique. The basic syntax for a for loop is :

```
for index, element := range someSlice {}
```

If there is a use for an index you can just assign it, if not just assign it as "_". Any for loop will not stray far from the basic format, because it is so easily to adapt, which makes it very clear and understandable. All of these examples are a testament to how simple Golang is.

There are some downsides to simplicity too, however. Take a while loop for example. There is no while keyword in Golang, a while loop in Go is for with a condition. This obviously makes sense, but the words that are used used can make it confusing. Another case for this is deleting and item from an array. There is no built in method for this. The easiest way to remove an item at a specific index is to make a new array by putting a sub array from every element before the index and a sub array with every element after the index. This is a workaround in lack for a built in method that can still be done with 1 line of code. This proves that simplicity does not make every situation easier, although that is the case most of the time.

Of course not every part of Go is basic and simple. Take goroutines for example. A goroutine is a lightweight thread of execution. Goroutines are used to run concurrent tasks and are an essential part of the language's concurrency model. Goroutines are created using the go keyword, followed by a function call. For example:

```
go foo()
```

This will create a new goroutine that runs the foo function concurrently with the calling code. By using goroutines it is possible smooth out and improve programs a lot. However, this does not mean that goroutines are an absolute necessity to use or know how to use in Go. It is perfectly possible to write a good program using Golang without the use of goroutines, but it is a way to improve and optimise programs if you have a better mastery over Golang.

Learning the gRPC model

A gRPC API makes use of a proto file so it is necessary to learn how to use it, but what is a proto file?

A proto file is a file that uses the Protocol Buffer language to define data structures and the operations that can be performed on them. Proto files are used in the gRPC framework to define the structure of the data that is exchanged between client and server and the operations that can be performed on that data. A proto file typically consists of a series of message definitions, which define the data structures that will be used in the API, and a series of service definitions, which define the methods that can be called on the server.

To use this proto file, you would first need to compile it into code for your desired language using the Protocol Buffer compiler (protoc). This would generate the necessary code for serializing and deserializing the data structures defined in the proto file and for implementing the service defined in the file. However, it does more than just generate the defined structures and methods, it also generates types and methods necessary for defining client and server variables, which are vital to make a working gRPC API, or use one.

Once a server is defined and can be run with the use of the proto generated file, all that is left is to write the methods mentioned in the proto file. This is mostly just normal programming exercise.

The benefits of gRPC

So what is the reason for learning gRPC, what is the use case of it? There are several reasons gRPC may be beneficial to use. Since there are a lot of different APIs there are some comparisons to the standard REST API.

First of all is the performance of the model. gRPC is designed to be fast and efficient, with a smaller footprint and lower overhead compared to REST. It uses HTTP/2 as the underlying transport protocol, which allows it to take advantage of features such as multiplexing and streaming.

The second benefit is the strong typing. It means gRPC uses Protocol Buffers for serialization, which provides strong typing and backwards compatibility. This can make it easier to evolve and maintain a gRPC API over time.

Next is bi-directional streaming. gRPC supports bi-directional streaming, which allows both the client and server to send a stream of messages to each other. This can be useful for scenarios where a large amount of data needs to be transferred or where the communication is interactive.

Fourth is the fact that it has compact payloads. gRPC uses a binary encoding for its payloads, which can result in smaller message sizes compared to REST, which uses JSON or XML. This can be beneficial in environments with limited bandwidth or for mobile applications.

Last is the efficient resource utilization that gRPC has. It has support for streaming and connection reuse can lead to more efficient resource utilization on both the client and server side.

Note that every type of has their own set of benefits. Such as REST being widely adopted and familiar to many developers, having a large number of available tools and libraries, and being well-suited for certain types of APIs. It is important to evaluate the specific needs and requirements of your API and choose the architecture that is the best fit for your use case. gRPC is a very powerful type of API but it certainly does not fit in every kind of environment. It is best used in big project that need a quick and high performance API, that can handle a lot of data, and can make use of the asynchronous streaming capabilities of gRPC.

Is Go a good choice for gRPC

Go is an excellent language to write a gRPC API with for several reasons.

Let's begin with the most important reason, Go has excellent support for gRPC. Golang has native support for gRPC and provides a number of libraries and tools to make it easy to build gRPC APIs. This is the main reason on why Go is a good choice, but it is not the only one.

Go's built-in concurrency features, such as goroutines and channels, make it easy to write concurrent code. This can be particularly useful when building a gRPC API and can multiply the functionality and ease the use of the streaming capabilities of gRPC.

Golang's comprehensive standard library provides a wide range of functionality out of the box, including support for networking, serialization, and cryptography. This can make it easier to build a gRPC API without needing to rely on external libraries with little to no documentation.

Go compiles quickly, which can be beneficial when working on a large project with many dependencies, which is often the case since gRPC is made to be an efficient API for larger projects.

As mentioned before, Go also has a large and active community of users, with many resources and libraries available online. This can make it easier to get help and support when working with Go and gRPC.

Summary

Both Golang and the gRPC model have a lot of potential and can go into extreme detail which makes them very hard subjects to master. I do think there is a difference in the learning difficulty between the two subject. Golang is very simplistic and easy to understand whereas gRPC has a steeper learning curve. gRPC is by no means rocket science, but it is a more complicated model for APIs especially when compared to REST. The difference in difficulty is not exclusive to building the API but also using it. Where a REST API is very easy to use in any circumstance, there should be some thoughts about whether the gRPC capabilities are necessary and how much effort it takes for the application/frontend to use the gRPC API.

Conclusion

It is absolutely possible to develop a good gRPC API in Go despite having no experience with gRPC or Go. There will always be room for improvement, especially if you begin from nothing. This does not take away that anyone can make a gRPC API in Go if there is demand for one, if some effort is put into learning make one.


References

H, J. (2022, 11 november). gRPC vs. REST: How Does gRPC Compare with Traditional REST APIs? DreamFactory Software- Blog. <https://blog.dreamfactory.com/grpc-vs-rest-how-does-grpc-compare-with-traditional-rest-apis/>

CodeOpinion. (2022, 23 juni). Where should you use gRPC? And where NOT to use it! YouTube. <https://www.youtube.com/watch?v=4SuFtQV8RCK>

sname. (2020, 2 november). Maruti Techlabs. <https://marutitech.com/rest-vs-grpc/>

Ogbonna, V. (2022, 6 september). Building a secure API with gRPC. Snky. <https://snyk.io/blog/building-a-secure-api-with-grpc/>

School, T. (2021, 26 juni). Is gRPC better than REST? Where to use it? DEV Community . <https://dev.to/techschoolguru/is-grpc-better-than-rest-where-to-use-it-3blg>

Laith Academy. (2022, 24 februari). Build a Rest API with GoLang. YouTube. https://www.youtube.com/watch?v=d_L64KT3SFM

Forbes, E. (2020, 28 april). Go gRPC Beginners Tutorial. TutorialEdge. <https://tutorialedge.net/golang/go-grpc-beginners-tutorial/>

One Minute Notes. (2022, 23 april). #10 Go Tutorial | Build A CRUD API with gRPC. YouTube. <https://www.youtube.com/watch?v=4gsDeKVfUUU>

TutorialEdge. (2020, 2 mei). Beginners Guide to gRPC in Go! YouTube. https://www.youtube.com/watch?v=BdzYdN_Zd9Q

Mulders, M. (2020, 30 november). Creating a web server with Golang. LogRocket Blog.
<https://blog.logrocket.com/creating-a-web-server-with-golang/>

Hitesh Choudhary. (2021, 3 oktober). Creating server for golang frontend. YouTube.
<https://www.youtube.com/watch?v=xh79JXJy0yY>

grpc package - google.golang.org/grpc - Go Packages. (z.d.). <https://pkg.go.dev/google.golang.org/grpc>

The Go Programming Language. (z.d.). <https://go.dev>

Quick start. (2022, 7 april). gRPC. <https://grpc.io/docs/languages/go/quickstart/>

Overview | Protocol Buffers |. (z.d.). Google Developers. <https://developers.google.com/protocol-buffers/docs/overview>

Bamimore, O. (2021, 2 december). Concurrency patterns in Golang: WaitGroups and Goroutines. LogRocket Blog. <https://blog.logrocket.com/concurrency-patterns-golang-waitgroups-goroutines/>

Gillis, A. S. (2020, 11 mei). Go (programming language). IT Operations.
<https://www.techtarget.com/searchitoperations/definition/Go-programming-language>