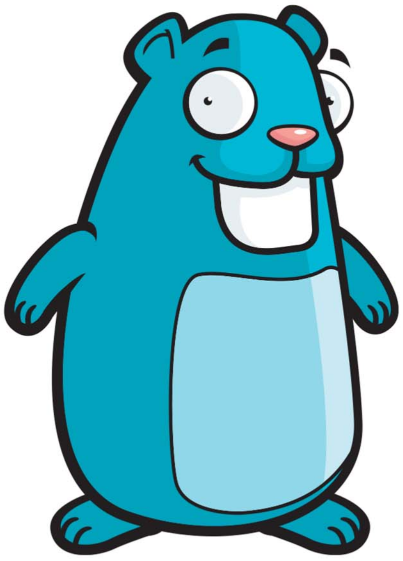
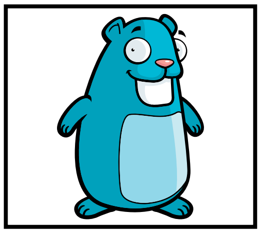


PRACTICAL GO

BUILDING SCALABLE NETWORK & NON-NETWORK APPLICATIONS

AMIT SAHA

WILEY



Practical Go

Building Scalable Network and Non-Network Applications

Amit Saha

WILEY

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

978-1-119-77381-8
978-1-119-77382-5 (ebk.)
978-1-119-77383-2 (ebk.)

Library of Congress Control Number: 2021948021

Cover image: © cthoman/Getty Images
Cover design: Wiley

I dedicate this book to all those who are working hard every day to find the right balance between the feeling of "Yes, I got this!" and "What am I even doing?" and continuing the battle that we call living.

About the Author

Amit Saha is a software engineer at Atlassian, located in Sydney, Australia. He has written *Doing Math with Python: Use Programming to Explore Algebra, Statistics, Calculus, and More!* (No Starch Press, 2015) and *Write Your First Program* (PHI Learning, 2013). His other writings have been published in technical magazines, conference proceedings, and research journals. He can be found online at `https://echorand.me`.

About the Technical Editor

John Arundel is a well-known Go writer, teacher, and mentor. He has been writing software for 40 years, and he thinks he's finally starting to figure out how to do it. You can find out more about John at `bitfieldconsulting.com`.

John lives in a fairytale cottage in Cornwall, England, surrounded by woods, wildlife, and a slowly deepening silence.

Acknowledgments

I want to thank the entire team at Wiley that made this book possible. First, Jim Minatel, who responded to my initial email expressing interest in publishing a book with Wiley. Jim then connected me with Devon Lewis, with whom I discussed the book proposal and who was instrumental in commissioning this book and overseeing the entire process. Next, I would like to thank Gary Schwartz, who in his capacity as project manager guided me throughout the entire project, ensuring that I was on track with my delivery of the chapters. Thank you, Judy Flynn, for your meticulousness in your role as the copyeditor. Finally, thank you, Barath Kumar Rajasekaran, for overseeing the proofreading process. Working with this team of fine folks was my version of a *Ulysses pact*.

John Arundel was kind enough to accept my request to be the technical reviewer of this book, and his insights and comments greatly helped to make the book better as well as making me a better Go programmer.

I want to thank all the community members on the Golang nuts and gRPC mailing lists who were always helpful in answering my questions and clarifying my doubts. My initial days of learning Go were mostly spent copying and pasting code from the "Go by example" project at `https://gobyexample.com`; hence I would like to acknowledge the efforts of the creator and maintainers of this project for their super-helpful resource.

Finally, I appreciate the efforts of the folks at Cooperpress who publish *Golang Weekly* (and the authors of the articles to which they linked) and *The Go Time* podcast. They helped me in learning Go and staying up to date with the latest happenings in the Go community.

— Amit Saha

Contents at a Glance

Contents

NOTE **A glossary of relevant terms is available for free download from the book's web page: `https://www.wiley.com/go/practicalgo`.**

Introduction

Google announced the Go programming language to the public in 2009, with the version 1.0 release announced in 2012. Since its announcement to the community, and the compatibility promise of the 1.0 release, the Go language has been used to write scalable and high-impact software programs ranging from command-line applications and critical infrastructure tools to large-scale distributed systems. The Go language has made a huge contribution to the growth of a number of modern software success stories. For a number of years, my personal interest in Go has been due to its, for the lack of a better word, *boring* nature—that's what I like about it. It felt like it combined the power of the second programming language I learned, C, with the batteries-included approach of another favorite language of mine, Python. As I have written more programs using the Go language, I have learned to appreciate its focus on providing all the necessary tools and features to write production-quality software. I have often found myself thinking, "Will I be able to implement this failure-handling pattern in this application?" Then I look at the standard library package documentation, and the answer has always been a resounding "Yes!" Once you have grasped the fundamentals of Go, with almost zero effort on your part as the software developer, the result is a highly performant application out of the box.

My goal in this book is to showcase the various features of the Go language and the standard libraries (along with a few community-maintained packages) by developing various categories of applications. Once you have refreshed or learned the language fundamentals, this book will help you take the next step. I have adopted a writing style where the focus is on using various features of the language and its libraries to solve the particular problem at hand—one that you care about.

You will not find a detailed walk-through of a language feature or every feature of a certain package. You will learn *just enough* to build a command-line tool, a web application, or a gRPC application. I focus on a strictly chosen subset of the fundamental building blocks for such applications to provide a compact and actionable guide. Hence, you may find that the book doesn't cover the more higher-level use cases that you may want to learn about. That is intentional, as the implementation of those higher-level use cases is often dependent on domain-specific software packages, and hence no single book can do justice to recommending one without missing out on another. I also strive to use standard library packages as far as possible for writing the applications in the book. This is again done to ensure that the learning experience is not diluted. Nonetheless, I hope that the building blocks you learn about in the book will provide you with a solid foundation to leverage higher-level libraries to build your applications.

What Does This Book Cover?

This book teaches you concepts and demonstrates patterns to build various categories of applications using the Go programming language. We focus on command-line applications, HTTP applications, and gRPC applications.

The Getting Started chapter will help you set up your Go development environment, and it lays down some conventions for the rest of the book.

Chapter 1 and Chapter 2 discuss building command-line applications. You will learn to use the standard library packages to develop scalable and testable command-line programs.

Chapter 3 and Chapter 4 teach you how to build production-ready HTTP clients. You will learn to configure time-outs, understand connection pooling behavior, implement middleware components, and more.

Chapters 5 through 7 discuss building HTTP server applications. You will learn how to add support for streaming data, implement middleware components, share data across handler functions, and implement various techniques to improve the robustness of your applications.

Chapters 8 through 10 delve deep into building RPC applications using gRPC. You will learn about Protocol Buffers, implement various RPC communication patterns, and implement client-side and server-side interceptors to perform common application functionality.

In Chapter 11, you will learn to interact with object stores and relational database management systems from your applications.

Appendix A briefly discusses how you can add instrumentation into your applications.

Appendix B provides some guidelines around deploying your applications.

Each group of chapters is mostly independent from the other groups. So feel free to jump to the first chapter of a group; however, there may be references to a previous chapter.

Within each group, however, I recommend reading the chapters from beginning to end, as the chapters within a group build upon the previous chapter. For example, if you are keen to learn more about writing HTTP clients, I suggest reading Chapter 3 and Chapter 4 in that order.

I also encourage you to write and run the code yourself as you work through the book and to attempt the exercises as well. Writing the programs yourself in your code editor will build that Go muscle, as it certainly did for me while writing the programs in the book.

Reader Support for This Book

You can find links to the source code and resources related to the book at `https://practicalgobook.net`. The code from the book is also posted at `https://www.wiley.com/go/practicalgo`.

If you believe that you've found a mistake in this book, please bring it to our attention. At John Wiley & Sons, we understand how important it is to provide our customers with accurate content, but even with our best efforts an error may occur. To submit your possible errata, please email it to our Customer Service Team at `wileysupport@wiley.com` with the subject line "Possible Book Errata Submission."

Getting Started

To start off, we will install the necessary software needed for the rest of the book. We will also go over some of the conventions and assumptions made throughout. Finally, I will point out key language features you will use in the book and resources to refresh your knowledge about them.

Installing Go

The code listings in this book work with Go 1.16 and above. Follow the instructions at `https://go.dev/learn/` to install the latest version of the Go compiler for your operating system. It usually involves downloading and running a graphical installation process for Windows or macOS. For Linux, your distribution's package repository may contain the latest version already, which means that you can use your package manager to install the Go compiler as well.

Once you have it installed, no further configuration is necessary to run the programs that you will write throughout the book. Verify that you have everything set up correctly by running the command `go version` from your terminal program. You should see an output telling you which Go version is installed and the operating system and architecture. For example, on my MacBook Air (M1), I see the following:

```
$ go version
go version go1.16.4 darwin/arm64
```

If you can see an output like the above, you are ready to continue with the next steps.

Choosing an Editor

If you don't yet have a favorite Go editor/integrated development environment (IDE), I recommend Visual Studio Code (`https://code.visualstudio.com/download`). If you are a Vim user, I recommend the vim-go extension (`https://github.com/fatih/vim-go`).

Installing Protocol Buffer Toolchain

For some chapters in the book, you will need the Protocol Buffers (protobuf) and gRPC tools for Go installed. You will install three separate programs: the protobuf compiler, `protoc`, and the Go protobuf and gRPC plug-ins, `protoc-gen-go` and `protoc-gen-go-grpc`, respectively.

Linux and macOS

To install the compiler, run the following steps for Linux or macOS:

1. Download the latest release (3.16.0 at the time of this book's writing) file from `https://github.com/protocolbuffers/protobuf/releases`, corresponding to your operating system and architecture. Look for the files in the *Assets* section. For example, for Linux on a x86_64 system, download the file named `protoc-3.16.0-linux-x86_64.zip`. For macOS, download the file named `protoc-3.16.3-osx-x86_64.zip`.
2. Next, extract the file contents and copy them to your `$HOME/.local` directory using the `unzip` command: `$ unzip protoc-3.16.3-linux-x86_64.zip -d $HOME/.local`.
3. Finally, add the `$HOME/.local/bin` directory to your `$PATH` environment variable: `$ export PATH="$PATH:$HOME/.local/bin"` in your shell's initialization script, such as `$HOME/.bashrc` for Bash shell and `.zshrc` for Z shell.

Once you have completed the preceding steps, open a new terminal window, and run the command `protoc --version`:

```
$ protoc --version
libprotoc 3.16.0
```

If you see output like the one above, you are ready to move on to the next step.

To install the protobuf plug-in for Go, `protoc-gen-go` (release v1.26), run the following command from a terminal window:

```
$ go install google.golang.org/protobuf/cmd/protoc-gen-go@v1.26
```

To install the gRPC plug-in for Go, `protoc-gen-go-grpc` (release v1.1) tool, run the following command:

```
$ go install google.golang.org/grpc/cmd/protoc-gen-go-grpc@v1.1
```

Then add the following to your shell's initialization file (`$HOME/.bashrc` or `$HOME/.zshrc`):

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

Open a new terminal window, and run the following commands:

```
$ protoc-gen-go --version
protoc-gen-go v1.26.0
$ protoc-gen-go-grpc --version
protoc-gen-go-grpc 1.1.0
```

If you see an output like above, the tools have been installed successfully.

Windows

NOTE You will need to open a Windows PowerShell window as an administrator to run the steps.

To install the protocol buffers compiler, run the following steps:

1. Download the latest release (3.16.0 at the time of this book's writing) file from `https://github.com/protocolbuffers/protobuf/releases`, corresponding to your architecture. Look for a file named `protoc-3.16.0-win64.zip` in the *Assets* section.
2. Then create a directory where you will store the compiler. For example, in `C:\Program Files` as follows: `PS C:\> mkdir 'C:\Program Files\protoc-3.16.0'`.
3. Next, extract the downloaded `.zip` file inside that directory. Run the following command while you are inside the directory where you downloaded the `.zip` file: `PS C:\> Expand-Archive.\protoc-3.16.0-win64\ -DestinationPath 'C:\Program Files\protoc-3.16.0'`.
4. Finally, update the Path environment variable to add the above path: `PS C:\> [Environment]::SetEnvironmentVariable("Path", $env:Path + ";C:\Program Files\protoc-3.16.0\bin", "Machine")`.

Open a new PowerShell window, and run the command `protoc --version`:

```
$ protoc --version
libprotoc 3.16.0
```

If you see an output like the one above, you are ready to move on to the next step.

To install the protobuf compiler for Go, `protoc-gen-go tool` (release v1.26), run the following command from a terminal window:

```
C:\> go install google.golang.org/protobuf/cmd/protoc-gen-go@v1.26
```

To install the gRPC plug-in for Go, `protoc-gen-go-grpc` (release v1.1) tool, run the following command:

```
C:\> go install google.golang.org/grpc/cmd/protoc-gen-go-grpc@v1.1
```

Open a new Windows PowerShell Window, and run the following commands:

```
$ protoc-gen-go --version
protoc-gen-go v1.26.0
$ protoc-gen-go-grpc --version
protoc-gen-go-grpc 1.1.0
```

If you see an output like the one above, the tools have been installed successfully.

Installing Docker Desktop

For the last chapter in the book, you will need the ability to run applications in software containers. Docker Desktop (`https://www.docker.com/get-started`) is an application that allows us to do that. For macOS and Windows, download the installer from the above website corresponding to your operating system and architecture, and follow the instructions to complete the installation.

For Linux, the installation steps will vary depending on your distribution. See `https://docs.docker.com/engine/install/#server` for detailed steps for your specific distribution. I also recommend that for ease of use (not recommended for production environments), you configure your docker installation to allow non-root users to run containers without using `sudo`.

Once you have followed the installation steps for your specific operating system, run the following command to download a docker image from Docker Hub and run it to ensure that the installation has been successfully completed:

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
109db8fad215: Pull complete
Digest: sha256:0fe98d7debd9049c50b597ef1f85b7c1e8cc81f59c8d
623fcb2250e8bec85b38
Status: Downloaded newer image for hello-world:latest
```

```
Hello from Docker!
This message shows that your installation appears to be
working correctly.
..
```

That completes our software installation for the book. Next, we will quickly cover some conventions used throughout the book.

Guide to the Book

In the following sections, you will learn various bits and pieces of information that will help you get the most out of the book. First, I discuss the choice of the module path for the code listings.

Go Modules

In this book, all applications will start by initializing a module as the first step. This will translate to running the `go` command, `go mod init <module path>`. Throughout the book, I have used a "placeholder" module path, which is `github.com/username/<application-name>`. Thus, in applications where we have written our module to consist of more than one package, the import path looks like this: `github.com/username/<application-name>/<package>`.

You can use these module paths if you are not planning to share these applications. If you plan to share your applications, or develop them further, you are encouraged to use your own module path, which is pointing to your own repository, likely a Git repository hosted on `https://bitbucket.org`, `https://github.com` or `https://gitlab.com`. Simply substitute `username` by your own username in the repository hosting service. It's also worth noting that the code repository for the book, `https://github.com/practicalgo/code`, contains the module path as `github.com/practicalgo/code/<chap1>/<application-name>`, in other words, an actual path that exists rather than a placeholder path.

Command Line and Terminals

You will be required to execute command-line programs throughout the book. For Linux and macOS, the default terminal program running your default shell is sufficient. For Windows, I assume that you will be using the Windows PowerShell terminal instead of the default command-line program. Most of the command-line executions are shown as executed on a Linux/macOS terminal, indicated by the `$` symbol. However, you also should be able to run the same command on Windows. Wherever I have asked you to execute a command to create a directory or copy a file, I have indicated the commands for both Linux/macOS and Windows, where they are different.

Terms

I have used some terms throughout the book that may be best clarified here to avoid ambiguity and set the right expectations.

Robustness and Resiliency

Both terms, *robustness* and *resiliency*, express the ability of an application to handle unexpected scenarios. However, these terms differ in their expected behavior under these circumstances as compared to their normal behavior. A system is *robust* if it can withstand unexpected situations and continue to function to some degree. This will likely be suboptimal behavior, as compared to normal behavior. On the other hand, a system is *resilient* if it continues exhibiting its normal behavior, potentially taking a finite amount of time before being able to do so. I put forward the following examples from the book to illustrate the difference.

In Chapter 2, you will learn to enforce time-outs for command-line application functionality that is executing a user-specified program. By enforcing time-outs, we avoid the scenario where the application continues to hang indefinitely because of bad user output. Since we configure an upper bound on how long we want to allow the user-specified command to be executed, we will exit with an error when this duration expires before the command could be completed. This is not the normal behavior of the application—that we should wait for the command to complete—but this suboptimal behavior is necessary to allow the application to recover from an unexpected situation, such as the user-specified command taking longer than expected. You will find similar examples throughout, notably when sending or receiving network requests in Chapters 4, 7, 10, and 11. We will refer to these techniques as introducing robustness in our applications.

In Chapter 10, you will learn to handle transient failures in your gRPC client applications. You will write your applications in a manner in which they can tolerate temporary failures that are likely to be resolved soon. We refer to this as introducing resilient behavior in our applications. However, we also introduce an upper time limit, which we allow to resolve the potentially temporary failure. If this time limit is exceeded, we consider that the operation cannot be completed. Thus, we introduce robustness as well.

To summarize, resiliency and robustness both aim to handle unexpected situations in our applications, and this book uses these terms to refer to such techniques.

Production Readiness

I use the term *production readiness* in the book as all steps that you should think about as you develop your application but before you deploy it to any kind of a production environment. When the production environment is your own personal server where you are the only user of your application, the techniques that you will learn will likely be sufficient. If the production environment means that your application will perform critical functionality for your users, then the techniques in this book should be the absolute baseline and a starting point. Production readiness consists of a vast body of often domain-specific techniques across various dimensions—robustness and resiliency, observability, and security. This book shows you how to implement a small subset of these topics.

Reference Documentation

The code listings in the book use various standard library packages and a few third-party packages. The descriptions of the various functions and types are limited to the contextual usage. Knowing where to look when you want to find out more about a package or function is important to get the most out of the book. The key reference documentation for all standard library packages is `https://pkg.go.dev/std`. When I import a package as `net/http`, the documentation for that package will be found at the path `https://pkg.go.dev/net/http`. When I refer to a function such as `io.ReadAll()`, the function reference is the package `io`'s documentation at `https://pkg.go.dev/io`.

For third-party packages, the documentation is available by going to the address `https://pkg.go.dev/<import path>`. For example, the Go gRPC package is imported as `google.golang.grpc`. Its reference documentation is available at `https://pkg.go.dev/google.golang.org/grpc`.

Go Refresher

I recommend going through the topics in "A Tour of Go," at `https://tour.golang.org/list`, to serve as a refresher of the various features that we will be using to implement programs in the book. These include for loops, functions, methods, struct and interface types, and error values. Additionally, I want to highlight the key topics that we will use extensively, along with references to learn more about them.

Struct Type

We will be using struct types defined by the standard library and third-party packages, and we will also be defining our own. Beyond defining objects of struct types, we will be working with types that embed other types—other struct types and interfaces. The section "Embedding" in the "Effective Go" guide (`https://golang.org/doc/effective_go#embedding`) describes this concept. We will also be making use of anonymous struct types when writing tests. This is described in this talk by Andrew Gerrand, "10 things you (probably) don't know about Go": `https://talks.golang.org/2012/10things.slide#1`.

Interface Type

To use the various library functions and to write testable applications, we will be making extensive use of interface types. For example, we will be making extensive use of alternative types that satisfies the `io.Reader` and `io.Writer` interfaces to write tests for applications that interface with the standard input and output.

Learning to define a custom type that satisfies another interface is a key step to writing Go applications, where we plug in our functionality to work with the rest of the language. For example, to enable sharing data across HTTP handler functions, we will define our own custom type implementing the `http.Handler` interface.

The section on interfaces in "A Tour of Go," `https://tour.golang.org/methods/9`, is useful to get a refresher on the topic.

Goroutines and Channels

We will be using goroutines and channels to implement concurrent execution in our applications. I recommend going through the section on Concurrency in "A Tour of Go": `https://tour.golang.org/concurrency/1`. Pay special attention to the example use of select statements to wait on multiple channel communication operations.

Testing

We will be using the standard library's `testing` package exclusively for writing all of the tests, and we will use Go test to drive all of the test executions. We have also used the excellent support provided by libraries such as `net/http/httptest` to test HTTP clients and servers. Similar support is provided by gRPC libraries. In the last chapter, we will use a third-party package, `https://github.com/testcontainers/testcontainers-go`, to create local testing environments using Docker Desktop.

In some of the tests, especially when writing command-line applications, we have adopted the style of "Table Driven Tests," as described at `https://github.com/golang/go/wiki/TableDrivenTests`, when writing the tests.