Inter-Service Communication with Go

Mastering protocols, queues, and event-driven architectures in Go

Dušan Stojanović

www.bpbonline.com

First Edition 2024

ISBN: 978-93-55517-289

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The information contained in this book is true to correct and the best of author's and publisher's knowledge. The author has made every effort to ensure the accuracy of these publications, but publisher cannot be held responsible for any loss or damage arising from any information in this book.

All trademarks referred to in the book are acknowledged as properties of their respective owners but BPB Publications cannot guarantee the accuracy of this information.

Dedicated to

My beloved friends:
***Marina**, **Jasmina**, **Tamara**,*
Borko**, **Miloš**, **Aleksandar** and **Simo

About the Author

Dušan Stojanović was born on May 27, 1989 in Smederevo, Serbia and raised in Baničina. He attended the University of Belgrade and received a Master's degree in computer science in 2013. Since then, he has been working in software development, playing central roles in numerous projects, such as user administration, online school platforms, e-commerce solutions, video-streaming platforms, advertising solutions, and AI chat applications as a software engineer.

His first book, **Building Server-side and Microservices with Go**, was published in 2021. Furthermore, he has published and written several technical articles on microservice development with Go and related topics.

He founded a small start-up company that tries to solve the parking problem in large cities. He lives in Belgrade (Serbia) and is a senior software developer.

About the Reviewers

- **Mahima Singla** is a dedicated Principal Software Design Engineer passionate about designing and implementing robust, scalable software solutions. With a focus on Cloud Governance, Cost Optimization, and Application fitment for the cloud, she excels in leveraging cutting-edge technologies, particularly in Cloud computing, AWS, and Kubernetes using Go and C#.

  Mahima is employed at Precisely Software and is crucial in the Studio Administrator Cloud project, contributing significantly to its architecture and development. She is proficient in AWS services like EC2, S3, and Lambda and creates resilient and scalable applications that align with business goals. Her expertise in Kubernetes demonstrates a commitment to staying ahead in container orchestration.

  Mahima has emphasized technical excellence, innovation, and collaboration throughout her career. She leads teams to deliver high-quality solutions and is dedicated to exceeding client expectations. Continuously staying updated with industry trends and embracing new technologies, she eagerly tackles challenges in cloud-native development, pushing the boundaries of what is possible in software engineering.

- **Andrii Klymenko** is an experienced backend engineer with a cloud-based solution certified by Amazon. He is focused on building highly available and scalable cloud-native solutions and tools, including AWS, PostgreSQL, Kafka, and Terraform. He is also focused on taking care of Infrastructure as Code and monitoring the correct thing with tools, such as AWS CloudWatch AWS X-Ray, with a passion for automating everything and avoiding doing anything that resembles manual work. During short breaks between consuming IT-related content to keep growing as a professional, he enjoys video games and playing guitar.

Acknowledgement

I want to express my deepest gratitude to my family and friends for their unwavering support and encouragement throughout the writing of this book, especially my brother Nikola and my parents, Slađana and Velimir.

I am also grateful to BPB Publications for their guidance and expertise in bringing this book to fruition. Revising this book was a long journey, with valuable participation and collaboration from reviewers, technical experts, and editors. Thank you for your patience and cooperation. All the revisions and the materials subsequently added made this book even better than initially imagined.

I also want to acknowledge the valuable contributions of my colleagues and co-workers, who have taught me so much during many years working in the tech industry. Special thanks to two of my dearest friends, Marijana and Stefan, for supporting my work and providing valuable feedback. Some of the materials included in this book are added based on their suggestions.

Special acknowledgment goes to my friends Marina, Jasmina, Tamara, Borko, Miloš, Aleksandar, and Simo, to whom I dedicate this book. The various communication approaches that we use between us were some sort of inspiration for this book.

Finally, I would like to thank all the readers who have expressed interest in my book and for their support in making it a reality. Your encouragement has been invaluable.

Preface

Modern software architecture often consists of multiple services that communicate with each other. Designing and building efficient and performant inter-service communication is essential. Without it, a solution will not provide a proper user experience. The Go programming language has become the preferred language for backend services due to its simplicity, efficiency, and concurrency support. A combination of well-designed inter-service communication systems and the Go programming language can create powerful software solutions.

This book was created to provide a guide for multiple inter-service communication concepts, including their advantages, disadvantages, and typical uses. All presented concepts will be covered with practical examples and implementations in the Go programming language. Implementations combine the Go standard library and third-party packages (when the standard library does not support a specific concept or the third-party package provides a more efficient solution). Code examples and illustrations will help you understand the concepts covered in each chapter.

This book is intended for developers familiar with the Go programming language who want to expand their knowledge and improve their development skills. It is also helpful for those who want to direct their careers towards software architecture and design. Many good ideas and practices can be found here.

This book will give you the knowledge and skills to become a proficient developer and recognize which inter-service communication concept best fits the specific software solution. I hope you will find this book informative and helpful.

Chapter 1: Fundamentals of Inter-Service Communication – It introduces the reader to basic inter-service communication concepts, like client, server, and communication channel. Furthermore,

the chapter also presents some service communication patterns. Ultimately, the code base for all examples covered in the following chapter is explained.

Chapter 2: RESTful Communication – It presents the basics of the REST architectural style and some best practices that can be used for implementation. The second part of the chapter covers SOAP. This protocol was predominantly used for inter-service communication in the past, and tried to compare it with REST, which has become standard for modern software solutions.

Chapter 3: HTTP – It covers the basics of HTTP and all related concepts, like Methods, Requests, Responses, etc. This chapter illustrates and explains how to install all the tools necessary for implementing communication between two services. Finally, it shows how to run and test the implemented solution.

Chapter 4: Protocol Buffers – This covers the basic concept of protocol buffers and the most popular solutions. Furthermore, the chapter introduces gRPC, the most popular protocol buffer solution. It also explains how to implement inter-service communication and run and test the solution.

Chapter 5: Message Queuing Protocols – It explains the concepts of message queuing protocols and presents AMPQ as the most popular. The second part of the chapter presents RabbitMQ as a well-known message queuing solution and explains how to install it and use it for implementing inter-service communication.

Chapter 6: Publisher/Subscriber – It introduces the Publisher/Subscriber design pattern and explains how it can be implemented with Redis (with installation guides). It also presents the advantages and disadvantages of this concept, with real-life examples.

Chapter 7: Event-Driven Architecture – It covers the basics of event-driven architecture and introduces Kafka as one of the most popular and widely used solutions. The second part of the chapter will explain and illustrate an implementation of inter-service communication based

on concepts of event-driven architecture with the Go programming language and Kafka.

Chapter 8: Final Observations – It summarizes previous chapters and offers some good practices related to inter-service communication. Particular emphasis will be on versioning and error handling as essential design concepts. Ultimately, some ideas for future development are presented.

Code Bundle and Coloured Images

Please follow the link to download the
Code Bundle and the ***Coloured Images*** of the book:

https://rebrand.ly/9a14ed

The code bundle for the book is also hosted on GitHub at **https://github.com/bpbpublications/Inter-Service-Communication-with-Go**. In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at **https://github.com/bpbpublications**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at **business@bpbonline.com** with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit **www.bpbonline.com**. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit **www.bpbonline.com**.

Table of Contents

Chapter 1
Fundamentals of Inter-Service Communication

Introduction

This first chapter will present the basics of inter-service communication, such as clients, servers, types of communication, and so on. We will discuss common communication patterns and frequent problems in inter-service communication. Ultimately, we will introduce a service template that will be used to implement different communication concepts in the following chapters.

Structure

The chapter covers the following topics:

- Basic communication concepts
- Inter-service communication
- Communication patterns
- Client-server implementation with Go

Objectives

After reading this chapter, you will be able to recognize common inter-service communication concepts and challenges. The following chapters will explain how to deal with these challenges.

Basic communication concepts

In communication, we have two sides: one that initiates communication to request something, called the client, and one that handles the request and provides a response, called the server. An example of client-server communication is presented in *Figure 1.1*:

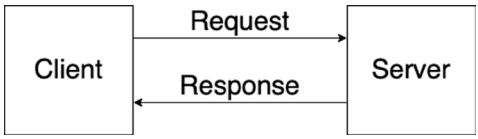

Figure 1.1: *Client-server communication*

In modern software systems, front-end applications (web portals, mobile applications) are often called clients, while back-end services are called servers. Regarding inter-service communication, the terms sender (caller) and receiver are frequently used instead of client and server. In the code examples in the following chapters, we will use this more generic naming convention. The client and server terms will be used in this chapter to keep everything by the book.

Client and server usually communicate through some communication channel. A channel can be some physical medium, like a telephone cable, or a logical connection over a multiplexed medium, such as a radio channel.

Two types of communication are most common:

- **Synchronous**: The client will wait for the server to respond. Real-life examples of synchronous communications are phone calls, video calls, in-person meetings, and so on.
- **Asynchronous**: The client will send a request to the server but will not wait for a response. It will continue to do something else until a response is received. Real-life examples of asynchronous communication are emails, chat applications like Slack, Messenger, or Viber, mail sent by post, and so on.

In the following section, we will see how synchronous and asynchronous communication can be used in inter-service communication. In contrast, we will discuss these topics in detail in the following chapters.

Inter-service communication

Modern software solutions most often consist of multiple services. Usually, one service requires information from another, so they must communicate. Let us assume we have a video streaming solution (like Netflix). In that system, we can have two services, one that handles users and the other that handles video content (movies and TV shows). All user actions (login, logout, and so on) will be handled by user service, while all video-related actions (adding a new movie, adding a new TV show, and so on) will be handled by video service.

However, there are situations when resources controlled by users and video services must be combined. For example, the system must know which video content the user watched. Then, services will communicate to exchange necessary data.

Each service in the system will have a database where data will be stored. There are no technical limitations for the user service to access the video service database, but this is not a good practice. One service should never access another service's database.

If the communication between the two services is synchronous, services will directly call each other, as shown in *Figure 1.2(a)*. REST architecture that uses HTTP protocol and gRPC are examples of how synchronous inter-service communication can be implemented.

If the inter-service communication is asynchronous, services will communicate through some message broker (Kafka, RabbitMQ). One service will send a message to the broker, and when another service is ready, it will pick up the message from the broker, as shown in *Figure 1.2(b)* (MB stands for message broker).

It is possible to combine synchronous and asynchronous approaches in the same system. For example, two services will communicate synchronously, while two others will communicate asynchronously, as shown in *Figure 1.2(c)*. Refer to the following figure: