Extracted from:

Practical Microservices

Build Event-Driven Architectures with Event Sourcing and CQRS

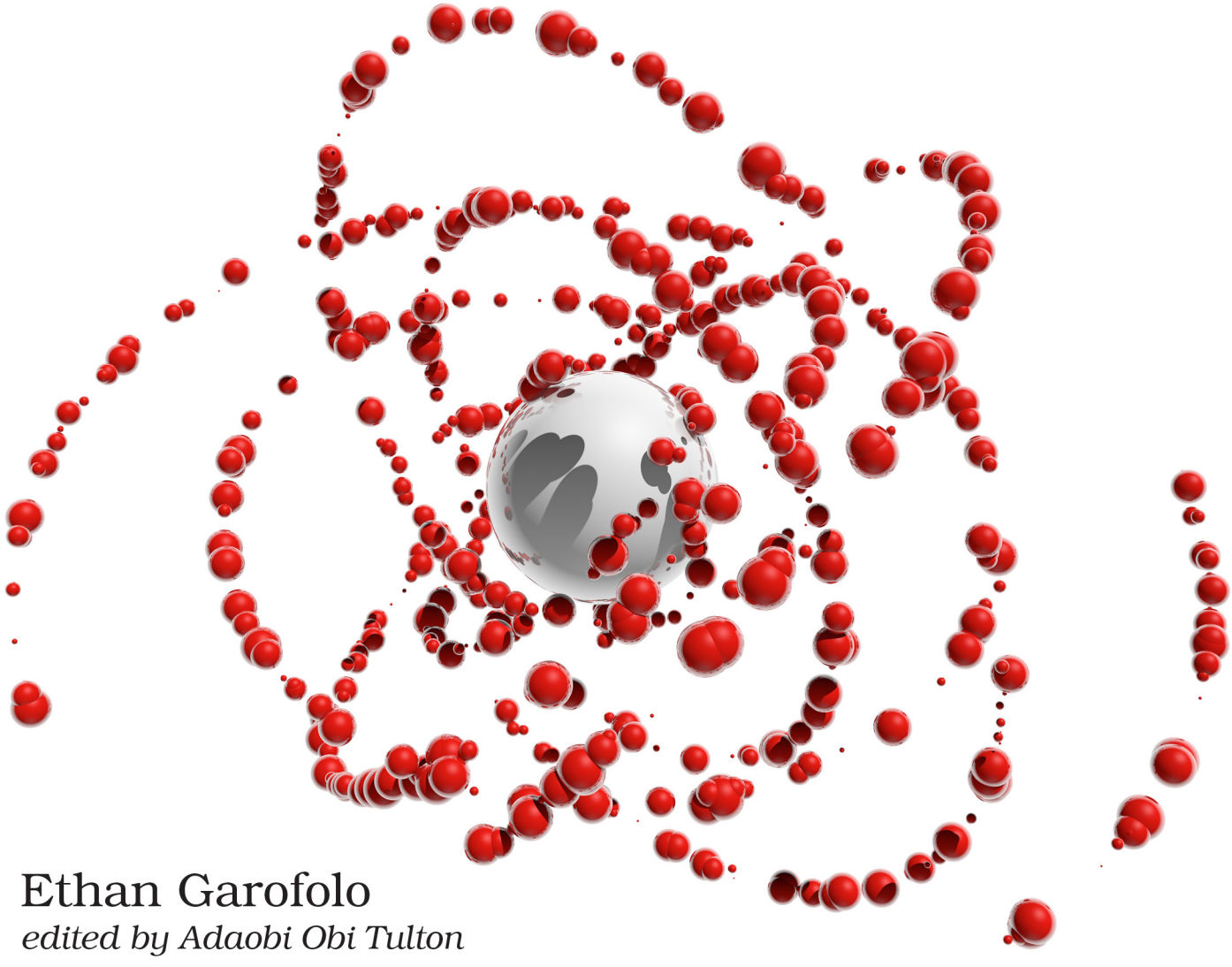
The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Practical Microservices

Build Event-Driven Architectures with Event Sourcing and CQRS



Ethan Garofolo

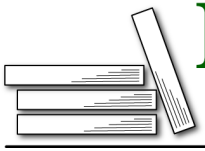
edited by Adaobi Obi Tulton

Practical Microservices

Build Event-Driven Architectures with Event Sourcing and CQRS

Ethan Garofolo

The Pragmatic Bookshelf
Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Executive Editor: Dave Rankin
Development Editor: Adaobi Obi Tulton
Copy Editor: Jasmine Kwityn
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-645-7
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—April 2020

For Julie, Sofia, Eva, and Angelo—

you are life and joy

Introduction

Here's the deal, y'all—software is easy to write, but it's hard to change.

If you're reading this book, then chances are you can write code. Maybe you're a software developer, a technical lead, an architect, or a CTO. You've shipped web-based applications that have provided a lot of business value to the people who paid you to do that. *You're good at what you do,* but something keeps happening with your projects, where productivity slows down despite spending more money and hiring more people. You want to learn how to fix that, and maybe you've heard about microservices and are wondering how to build them.

Telling a Tale of a Different Kind of Keyboard

So let's digress into a story about something that at once is what many adults wish to have spent more time practicing and what children often dread: the piano. I never really learned the piano, but I did learn one song well enough that I was asked to play it in a church group. The song is called "A Poor Wayfaring Man of Grief," and really liking it, I figured I needed to know how to play it. I can look at a staff and figure out what the notes are, but I can't do that fast enough to read the music and press the corresponding keys in real time.

So I set about memorizing the song. One. Beat. At. A. Time.

I put my fingers for both hands in place for the first beat and played it, removed my fingers from the keys, and then without looking at the music put them back and played that first set of notes. Then I read the second set and basically played a game of Simon[1] until I had memorized the song.

And I did. I could reproduce the notes. But woe was me when I made mistakes halfway through the song! It was a bit like typing a password. When you goof on entering a password, you generally have to start over.

1. https://en.wikipedia.org/wiki/Simon_(game)

One day a friend of mine offered to teach me piano properly. We sat down, and she observed my playing. She was gracious in how she let me know, "If you're ever going to play this instrument, you're going to have to unlearn a lot of things."

There's a technique to piano. For example, if you're walking your left hand up the keys, you'll play notes from your pinky to your thumb. When it's time to go to higher notes, your middle finger slides over to the right of your thumb, and it makes for an elegant trip up to the higher pitches. As an understatement, I did not discover this technique when I was memorizing that song. Folks who do learn good technique can work an amazing art.[2] What these other folks do is closed to me until I pay the price of unlearning.

Learning microservices-based architecture was and continues to be a similar process of unlearning. You're likely skilled in Model-View-Controller (MVC) frameworks that model systems in terms of Create-Read-Update-Delete (CRUD) operations. Maybe you've used frameworks like Ruby on Rails or Django.

But you keep running into a productivity wall. It becomes extremely difficult to add features and fix bugs. You've changed frameworks and you've changed languages. Is every project fated to hit this point where every enhancement is excruciating?

The answer to that question is a resounding "NO!"

What This Book Is

This is a hands-on tutorial. In this book, you're going to build a system according to a different set of principles—ones that will allow you to build systems where your productivity will not come to an encumbered halt. You're going to build *microservices*, focused units of autonomous functionality.

Chapter 1, You Have a New Project, on page ? introduces the project you're going to build, and we'll take a crack at it using the traditional CRUD-based approach, seeing how it paints us into a corner. Following that, in Chapter 2, Writing Messages, on page ? we'll unmask the monolith, and you'll see why most writing on microservices misses the point. We'll conclude this part of the book by writing code to interact with a piece of technology called a message store—a database optimized for storing asynchronous messages. These messages will form the basis of your architecture.

2. https://www.youtube.com/watch?v=9fAZIQ-vpdw

With the fundamentals in place, you'll leverage this message store to add features to your system by building microservices and other autonomous components. You'll send emails (Chapter 9, Adding an Email Component, on page ?) and even transcode videos (Chapter 10, Performing Background Jobs with Microservices, on page ?). Along the way, the architecture you implement will allow you to slice and dice the same data in varied and interesting ways.

Once the bulk of the system is in place, in Chapter 12, Deploying Components, on page ?, you'll deploy your microservices-based system into the wild. In Chapter 13, Debugging Components, on page ? you'll learn how this message-based architecture helps you keep a health system running. You'll even get exposure to how this architecture alters how you test your system in Chapter 14, Testing in a Microservices Architecture, on page ?.

What This Book Isn't

This is a hands-on tutorial, not a reference book or *The Definitive Guide*™ to microservices. There are topics we won't cover in detail. In Chapter 15, Continuing the Journey, on page ?, the final chapter, we call out some of these topics, and you'll have pointers on how to continue your learning.

This isn't a book about flashy technology. You're going to use PostgreSQL,[3] a world-class relational database, and that's about as exciting as the tools get. The excitement, instead, is in the principles.

Separating Principles from Implementations

This is the book that I wish had been written years ago when I first started learning service-based architecture. Finding information on high-level principles is pretty easy, but there weren't any hands-on tutorials. But by getting so low-level in this book, my biggest fear is that you interpret this particular implementation as microservices gospel. The details here serve only to show what the principles might look like in practice.

Using This Book

The chapters in the book build on top of one another. In Chapter 1, you start writing a project that we add to as we go throughout the book. I can't imagine a reading order other than "each chapter in order" that makes sense. Maybe I'm lacking in imagination, but that's my recommendation to you.

3. https://www.postgresql.org

As you're reading through the code, I can't stress highly enough the value of creating your own project folder, creating each file, and typing its contents yourself. Is that tedious? Possibly. But it's tedium with a purpose. Reading code is nice, but I want to get the concepts in this book into your brain, and typing code will do so much more of that transfer than merely reading. In addition, you should also be sure to complete the exercises in the chapters.

Using Node.js

For better or for worse, we use JavaScript and the Node.js runtime to implement our system. The principles we cover absolutely transcend programming languages, but bringing them down to the mundane requires implementing them in *something*. We use Node.js because it's a fine language when you stay away from its warts—which we do—and because most developers who have written code to run on the web can at least grok JavaScript.

Strapping In

Well, then—let's buckle our seat belts and get rolling. I'm glad you're along for the journey.