O'REILLY®

Efficient Node.js

A Beyond-the-Basics Guide

Samer Buna

O'REILLY®

Efficient Node.js

A Beyond-the-Basics Guide

Samer Buna

Efficient Node.js

A Beyond-the-Basics Guide

Samer Buna

Efficient Node.js

by Samer Buna

Printed in the United States of America.

Published by O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Acquisitions Editor: Amanda Quinn

Development Editor: Jeff Bleiel

Production Editor: Gregory Hyman

Copyeditor: Miah Sandvik

Proofreader: Stephanie English

Indexer: Judith McConville

Interior Designer: David Futato

Cover Designer: Karen Montgomery

Illustrator: Kate Dullea

January 2025: First Edition

Revision History for the First Edition

2025-01-08: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781098145194* for release details.

978-1-098-14519-4

[LSI]

Preface

I’ve been using Node.js since its early days, and it has never failed me. With every piece of code I’ve written for Node, my appreciation for it has only increased. With every new skill I’ve developed for Node, I’ve felt the productivity gain.

Node.js is nothing short of revolutionary. It’s a great platform with impressive power. Once you get comfortable with its fundamentals and how it handles asynchrony, the rest is easy. You’ll get better with it quickly, and you’ll be able to build and scale backend services faster than you’d imagine.

Who Should Read This Book

This book is my attempt at helping you learn Node.js efficiently. It naturally dips into a few JavaScript concepts, but in general, you need a good basic understanding of the JavaScript language to get the most value out of this book.

If you’re not comfortable working with JavaScript objects, functions, operators, and iterators, reading an introductory book about JavaScript before this book would help.

This is the book that I wished existed when I started learning Node.js. At that time, I was mainly focusing on the frontend. Naturally, this book is a good fit for a frontend developer wanting to expand their experience to the backend.

Why I Wrote This Book

When it comes to learning Node.js, many tutorials, books, and courses tend to focus on the libraries and tools available within the Node.js ecosystem, rather than the Node.js runtime environment itself. They prioritize teaching how to utilize popular Node.js libraries and frameworks, instead of starting from the native capabilities of Node.js.

This approach is understandable because Node.js is a low-level runtime environment. It does not offer comprehensive solutions but rather a collection of small essential modules that makes creating solutions easier and faster. For example, a full-fledged web server will have options like serving static files (like images, CSS files, etc.). With the Node.js built-in

http module, you can build a web server that serves binary data, and with the Node.js built-in fs module, you can read the content of a file from the filesystem. You can combine

both of these features to serve static assets by using your own JavaScript code. There's no built-in Node.js way to serve static assets under a web server.

Popular Node.js libraries that are not part of Node.js itself (such as Express.js, Next.js, and many others with *.js* in their names) aim to provide nearly complete solutions within specific domains. For example, Express.js specializes in creating and running a web server (and serving static assets, and many other neat features). Practically, most developers will not be using Node.js on its own, so it makes sense for educational materials to focus on the libraries offering comprehensive solutions, so learners can skip to the good parts. The common thinking here is that only developers whose job is to write these libraries need to understand the underlying base layer of Node.js.

However, I would argue that a solid understanding of the built-in power of Node.js is essential before utilizing any of its external libraries and tools. Having a deep understanding of Node.js allows developers to make informed decisions when choosing which libraries to use and how to use them effectively. This book is my attempt to prioritize first learning the native capabilities of Node.js and then using that knowledge to efficiently utilize the powerful libraries and tools in its ecosystem.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

Indicates new terms, URLs, email addresses, filenames, and file extensions.

Constant width

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

Constant width italic

Shows text that should be replaced with user-supplied values or by values determined by context.

TI P

This element signifies a tip or suggestion.

N O TE

This element signifies a general note.

WARNING

This element indicates a warning or caution.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at *https://oreil.ly/EfficientNodeCode*.

If you have a technical question or a problem using the code examples, please send email to *support@oreilly.com*.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but generally do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Efficient Node.js* by Samer Buna (O'Reilly). Copyright 2025 Samer Buna, 978-1-098-14519-4."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

O'Reilly Online Learning

NOTE

For more than 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, visit *https://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

800-889-8969 (in the United States or Canada)

707-827-7019 (international or local)

707-829-0104 (fax)

support@oreilly.com

https://oreilly.com/about/contact.html

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *https://oreil.ly/EfficientNodeJS*.

For news and information about our books and courses, visit *https://oreilly.com*.

Find us on LinkedIn: *https://linkedin.com/company/oreilly-media*.

Watch us on YouTube: *https://youtube.com/oreillymedia*.

Acknowledgments

I am deeply grateful to the many incredible people whose efforts helped shape and refine this book. A heartfelt thank you to the O'Reilly Media team for their patience, guidance, and unwavering support throughout the writing process. I am especially thankful to Jeff Bleiel, my development editor, whose thoughtful insights greatly improved the clarity and presentation of this work. To Amanda Quinn, my editor; Miah Sandvik, my copyeditor; and Gregory Hyman, my production editor: your contributions are greatly appreciated.

I also want to thank the talented software developers who reviewed drafts of this book and provided invaluable feedback. Your suggestions were instrumental in improving the content. Special thanks to Hazem Twair for spotting issues I would have missed, and to Tamas Piros and Aniket Wattamwar for their meticulous technical proofreading. Your input made this book stronger and more polished.

To my mentors and peers in the Node.js community: I am forever grateful for your wisdom, inspiration, and encouragement.

To everyone who contributed: your dedication and expertise have elevated this book far beyond what I could have achieved on my own. Thank you for your help—it means the world to me!

Chapter 1. Node Fundamentals

Node is an open source, cross-platform runtime environment in which developers can create backend services using the JavaScript language. It's built on top of V8, the JavaScript engine of the Chrome web browser, and it has dozens of built-in modules that are designed to be used asynchronously with an event-driven approach that's commonly known as the non-blocking model. Node developers can use events and handler functions to efficiently perform multiple operations in parallel, without having to deal with the complexity of multiple processes and threads.

There's a lot to unpack here, and that's what we will be doing in this first chapter. We'll start with an introduction to Node, how it works, and why it's popular. We'll learn the basics of the Node CLI, how to use modules and packages, and how to perform synchronous and asynchronous operations. We'll discuss the fundamentals of Node's event-driven, non-blocking model and learn how callbacks, promises, and events can be used to handle the result of an asynchronous operation.

NOTE

Throughout the book, I use the term *Node* instead of *Node.js* for brevity. The official name of the runtime environment is Node.js, but referring to it as just Node is common.

Introducing Node

Ryan Dahl started the Node project in 2009 after he was inspired by the performance of the V8 JavaScript engine in the Google Chrome web browser. V8 uses an *event-driven model*, which makes it efficient at handling concurrent connections and requests. Ryan wanted to bring this same high-performance, event-driven architecture to server-side applications. The event-driven model is the first and most important concept you need to understand about Node (and the V8 engine as well). I'll explain it briefly in this chapter, and we'll expand on it in Chapter 3.

TI P

I decided to give Node a spin and learn more about it after watching the presentation Ryan Dahl gave to introduce it. I think you'll benefit by starting there as well. Search YouTube for "Ryan Dahl introduction to Node". Node has changed significantly since then, so don't focus on the examples but rather the concepts and explanations.

In its core, Node enables developers to use the JavaScript language on any machine without needing a web browser. Node is usually defined as "JavaScript on backend servers." Before Node, that was not a common or easy thing. JavaScript was mainly a frontend thing.

However, this definition isn't completely accurate. Node offers a lot more than the ability to execute JavaScript on servers. In fact, the actual execution of JavaScript is done by the V8 JavaScript engine, not Node. Node is just an interface to V8 when it comes to executing JavaScript code.

V8 is Google’s open source JavaScript engine that can compile and execute JavaScript code. It’s used in Node as well as in Chrome and a few other browsers. It’s also used in Deno, the new JavaScript runtime that was created by Ryan Dahl in 2018.

NOTE

There are other JavaScript engines, like SpiderMonkey, which is used by Firefox, and JavaScriptCore, which is used by the Safari web browser and in Bun, an all-in-one JavaScript runtime, package manager, and bundler.

Node is better defined as a server runtime environment that wraps V8 and provides modules to help developers build and run efficient software applications with JavaScript.

The key word in this definition is *efficient*. Node adopts and expands on the same event-driven model that V8 has. Most of Node’s built-in modules are event-driven and can be used asynchronously without blocking the main *thread* of execution that your code runs in.

A thread is basically a small process within a larger one. A process can create multiple threads of execution that are each associated with a CPU core. Threads can share memory and resources within the larger process.

In multithreaded programming, slow operations are executed in separate threads. In Node, you get a single main thread for your code, and all the slow operations are executed outside of that main thread, asynchronously.

You need to read the content of an external file? You can do that asynchronously without blocking the single main thread. You need to start a web server? Work with network sockets? Parse, compress, or encrypt data? Every low-level slow operation has an asynchronous API for you to use without blocking your other operations.

You don't need to deal with multiple threads to do things in parallel in Node. You don't waste resources on manual threads being idle waiting on slow operations. You code in one thread and use asynchronous APIs, and Node takes care of executing the asynchronous operations efficiently outside of your main thread.

Any code that needs to be executed after a slow operation can be managed with *events* and *event handlers*. An event is a signal that something has happened and a certain action needs to be performed. The action can be defined in an event handler function that gets associated with the event. Every time the event is signaled, its handler function will be executed.

That's basically the gist of what *event-driven* means.

We'll expand on these important concepts once we learn the basics of running Node code and using its modules and packages.

The JavaScript Language

After considering programming languages like Python, Lua, and Haskell, Ryan Dahl picked the JavaScript language for Node because it was a good fit. It's simple, flexible, and popular, but more importantly, JavaScript functions are first-class citizens that we can treat like any other objects (numbers or strings). We can store them in variables, pass them to other functions via arguments, and even return them from other functions, all while preserving