Extracted from:

Docker for Rails Developers

Build, Ship, and Run Your Applications Everywhere

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Docker for Rails Developers

Build, Ship, and Run Your Applications Everywhere

Rob Isenberg
edited by Adaobi Obi Tulton

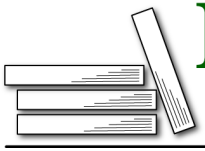
Docker for Rails Developers

Build, Ship, and Run Your Applications Everywhere

Rob Isenberg

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
Managing Editor: Susan Conant
Development Editor: Adaobi Obi Tulton
Copy Editor: Nicole Abramowitz
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: 978-1-68050-273-2
Book version: P1.0—February 2019

Ruth. In hindsight, writing a book whilst having a baby and renovating a house probably wasn't the best idea—who knew? Thank you for your patience, love, and support. None of this would have been possible without you.

Sammy. I couldn't have imagined the joy and love you'd bring into our life. Be kind, be brave, and be willing to take risks in pursuit of your happiness and passions. I love you so much.

Mum and Dad. Thank you for everything.

Acknowledgements

Thanks to Adaobi, my editor at The Pragmatic Bookshelf, for her constant positivity and encouragement, as well as excellent editing feedback on the book to help make it as good as it can be. I will miss our updates and bonding over our mutual love of Gordon Ramsay.

I owe a great deal of thanks to the following people who gave up their valuable time to read and provide feedback on the book (a thousand apologies if I have left anyone out):

- John Paul Ashenfelter
- David L. Bean
- Erin Dees
- Chris Johnson
- David Landry
- Nigel Lowry
- Alex Lynham
- Lee Machin
- Rory McCune
- Noel Rappin
- Chris Thorn
- John Yeates

The book is immeasurably better as a result of their contributions.

I'd also like to thank everyone who purchased a beta copy of the book while it was still being written. In particular, I'd like to thank people who submitted errata during this process—your confusion, frustration, and pain have hopefully saved others from suffering the same fate.

Finally, a huge thank you to the entire Pragmatic Bookshelf team for taking on and supporting this title.

Introduction

If you love Ruby on Rails, you're going to love Docker. They are kindred spirits, born out of similar ideals.

For me, the allure of Rails was its Big Ideas: generators, migrations, testing as a first-class citizen, convention over configuration, multi-environment setups built in, live-code reloading. While, individually, these features may not have been new, the combination made Rails more than the sum of its parts: it gave us superpowers.

Docker is doing for DevOps what Rails did for web development. It too is packed with Big Ideas: a holistic view of your app (hint: your app is more than just your Rails code), containerization (lighter-weight, faster, and more efficient than VMs), software delivery that doesn't suck (for example, Ruby installs the first time you run a Ruby script), fault-tolerant clustering and scaling out of the box (spin up production-like clusters on your local machine), expert-level security features baked in (for example, automatic key rotation). The list goes on.

Docker is lowering the barrier to entry, making DevOps tasks that previously would have been unthinkable suddenly within our grasp. It gives us a *new* set of superpowers.

That said, Docker is not a panacea or a silver bullet to solve all your DevOps challenges. As with all technologies, there are trade-offs (I'll try to point these out as we go). However, despite the trade-offs, as you'll discover in this book, there is value in adopting Docker.

What Is Docker?

Docker, the technology, is a set of tools built around the idea of packaging and running software in small, sandboxed environments known as *containers* (we'll get to the nitty gritty of these in *What Is a Container?*, on page ?).

At a high level, Docker provides five capabilities:

- *Packaging.* The ability to package software into a reusable, shareable format known as *images*.
- *Distribution.* The ability to easily share packaged software (images) with other people and deploy it to different machines.
- *Runtime.* The ability to run, pause, restart, or stop packaged software in a reliable, repeatable way.
- *Infrastructure creation.* Creating virtual machines ready to run our Docker containers.
- *Orchestration and scaling.* Managing the release of software to a single Docker node or across an entire cluster.

Together, these five things combine to enable a new way of delivering and running software.

Why Use Docker?

To build a Rails app, we typically develop on our local machine. Rather than each team member manually maintaining their own local development environments, we can use Docker to provide a common, standardized environment. This saves on repeated effort and helps avoid many forms of the "works on my machine" issues that can waste hours.

Other benefits of using Docker for your development environment include:

- *A holistic view of your app.* Rails apps typically need a database and other external dependencies like Redis and Elasticsearch. With Docker, these dependencies are no longer an afterthought or "add-on" like in Heroku; they are described and managed as fundamental parts of your app.
- *Single-command app installation and setup.* Have you ever set up a Rails app on your machine and spent an excessive amount of time installing specific versions of its software dependencies? Docker's built-in delivery mechanism means that new team members can go from zero to a running app in minutes. No laborious, error-prone, manual setup steps here.
- *Easy version management of dependencies.* Want to make sure everything works before switching to a new version of Ruby or upgrading the database? No problem: running containers is cheap. Just change the image version and away you go.

- *Huge Docker ecosystem.* We frequently need to incorporate other technologies as part of our Rails apps: NGINX, Redis, Postgres, MySQL, Memcached, Elasticsearch, HAProxy, RabbitMQ, Node, and so on. All these and more are already packaged and ready to go with Docker.
- *Simulate production-like environments locally.* We know that how our Rails app performs in development isn't exactly the same as in production. With Docker, you can simulate production scenarios by running your app in multi-node, production-like environments on your local machine.

Docker can also help once you move beyond development. It provides a consistent interface, whether you're running locally, on a continuous integration (CI) server, or deploying to production. Once built, the same image is run at every stage of your continuous integration/delivery pipeline, giving us confidence that our tested application will perform the same in each environment.

If you need to manage and deploy to your own production infrastructure, there are further benefits:

- *Deployment standardization.* Docker provides a standard way of packaging and delivering applications: each part of your app is a container, and each app is a collection of containers. From a DevOps perspective, one Docker app is deployed and managed in the same way as any other.
- *Reliability and resiliency features built in.* Ever been woken at 3 a.m. by a cranky CEO because your app's gone kaput? Docker clusters are self-healing: if an instance dies, new copies of your app will be spawned on the remaining nodes.
- *Reducing infrastructure costs especially at scale.* Containers are much lighter-weight than virtual machines (VMs), allowing resources to be used more efficiently. They also let you scale up the number of containers on a single host rather than spinning up an entire new instance.
- *Room to grow.* If your app is (or becomes) wildly successful, it's good to know that Docker has been battle-tested at massive scale. Google Compute Engine, for example, is built on Docker containers, using Google's open source orchestration tool, Kubernetes.

Who Should Read This Book?

This book is for experienced Rails developers who want to learn how to use Docker. I'm going to assume, throughout the book, that you're proficient at using Rails; this will allow us to focus on learning and applying Docker.

This book doesn't aim to be a comprehensive manual on Docker: several other books serve that aim. Rather, this book is your field manual to *building Rails applications with Docker.* We'll cover the most useful commands and features that you'll need, and I'll refer you to reference material as needed.

If you're curious to discover how Docker can fit into your day-to-day workflow as a Rails developer, you've come to the right place.

What's in This Book?

In Part I, you'll learn everything you need to know about using Docker for local Rails development, including core concepts like containers and images. You'll build up real-world knowledge, step by step, through a series of practical tasks. We'll start with the basics—running a Ruby script and generating a new Rails project—before learning how to run our Rails app by building our own custom image.

We'll quickly move on to Compose, a higher-level Docker tool for declaratively describing an entire app, and how it all fits together. As you learn more, we'll gradually layer up services like a database and Redis. We'll cover how to set up and run your tests so that you're fully proficient at using Docker for Rails development.

In Part II, we'll explore the process of deploying and running an application in production. We'll start by giving you an overview of the production landscape—the tools, platforms, and technologies that can be used. Next, using Docker's own tools, we'll provision machines, create a cluster, and deploy our app. We'll also scale our app's resources to meet its changing needs.

How to Read This Book

Docker has a challenging learning curve. It's a vast tool and ecosystem, and there's a lot to understand. Hopefully this book will help—it's carefully structured to avoid introducing too many new things at once.

Each chapter builds on the one preceding it, so, especially if you're unfamiliar with Docker, I recommend reading the book in sequence to get the most benefit. Even if you have more Docker experience under your belt already, this is the recommended approach.

Docker IDs and Following Along Yourself

Docker generates various unique IDs. When following the examples, it's important to remember that the IDs generated for you will be different from those shown in the output. Don't worry, though; I'll point this out where it's particularly relevant.

Which Operating Systems Are Supported?

Although Docker is supported on all major platforms (macOS, Windows, and Linux)—and we'll lead you through the process of installing it on these in *Installing Docker*, on page ?—there are some minor differences between the platforms, particularly around file permissions and networking.

For that reason, I've chosen Docker for Mac as the default platform in the examples and discussion, but I'll point out any differences between other platforms when they come up.

Some Linux/Unix Knowledge Is Recommended

Even with Docker on Windows or Mac, there's no avoiding the need to understand some Linux basics. Docker evolved out of Linux kernel features, so explanations and examples often rely on Linux concepts and programs. I'm going to assume you have this knowledge already. If not, there are plenty of free resources online you can use to learn more or brush up if you need to.

Online Resources

You can find useful resources related to the book online,[1] including:

- The source code used throughout the book (you're free to use this in any way you'd like)
- An errata page, which lists corrections for the current edition

Let's get started!

1. http://pragprog.com/book/ridocker