The Pragmatic Programmers

Deploying with JRuby 9k

Deliver Scalable Web Apps Using the JVM

Joe Kutner

Edited by Brian P. Hogan

Early praise for *Deploying with JRuby 9K*

Joe has pulled together a great collection of deployment knowledge from his years of experience building and supporting JRuby applications. He's an expert on this subject and *Deploying with JRuby 9k* is the definitive text for getting JRuby applications up and running.

➤ **Charles Oliver Nutter**
JRuby co-lead

Deploying with JRuby 9k answers the most frequently asked questions about real-world use of JRuby. Whether you're coming to JRuby from Ruby or Java, Joe fills in all the gaps you'll need to deploy JRuby with confidence.

➤ **Tom Enebo**
JRuby co-lead

I've been working with JRuby for years and I still learned several immediately actionable steps to improve the performance and maintenance of real-world JRuby apps.

➤ **Matt Margolis**
director, application development at Getty Images

Deploying with JRuby 9k is full of practical and actionable advice about how to get the most benefit out of the JVM when running your Ruby app on JRuby.

➤ **Chris Seaton**
Oracle Labs and JRuby contributor

Deploying with JRuby 9k is the essential guide for anyone building Ruby applications on the JVM. It's loaded with tips, tricks, and best practices that newcomers and experts can learn from.

➤ **Terence Lee**
Ruby task force member at Heroku

As a developer of MRI, I get super jealous reading about the JVM ecosystem and tooling. With this book, Joe has finally made that ecosystem approachable for JRuby applications.

➤ **Zachary Scott**
Ruby-core member and maintainer of Sinatra

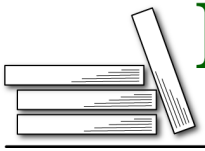
Deploying with JRuby 9k

Deliver Scalable Web Apps Using the JVM

Joe Kutner

The Pragmatic Bookshelf

Raleigh, North Carolina



Pragmatic Bookshelf

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:

Brian P. Hogan (editor)
Potomac Indexing, LLC (index)
Linda Recktenwald (copyedit)
Gilson Graphics (layout)
Janet Furlow (producer)

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

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

Printed in the United States of America.
ISBN-13: 978-1-68050-169-8
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—July 2016

Contents

Acknowledgments

Writing a book is a lot like writing code. You need to know the rules, recognize patterns, and occasionally know when to break the rules. Both writing and coding are crafts. And like with any craft, you improve by getting advice from more experienced professionals and being critiqued by your peers. I'm so fortunate to have had this kind of help.

I'm inexpressibly thankful to those who reviewed this book prior to its publication. I was humbled by the attention to detail and wise feedback they provided in making it a finished product. Thank you, Jeff Holland, Margaret Le, Matt Margolis, Jay McGaffigan, Chris Seaton, and Tim Uckun. I consider you all to be my friends!

I'd also like to thank the staff at the Pragmatic Bookshelf: Susannah Pfalzer, Dave Thomas, Andy Hunt, and probably a whole bunch of other people I don't know about.

Above all, thank you, Brian P. Hogan, my editor. This is our fourth endeavor together, and as usual I've become a better writer because of it. I look forward to working on future projects together.

I must also thank the creators of the technologies I've written about. This book would not have been possible without your hard work. Thank you, Charles Nutter, Thomas Enebo, Karol Buček, Christian Meier, Chris Seaton, and the rest of the JRuby team. You're the most amazing group in all of the open source world. I owe you all my deepest gratitude and a free beverage.

Finally, I'd like to thank my wife and son. I could not have completed this project without your love and support.

Preface

Your website has just crashed, and you're losing money. The application is built on Rails, runs on MRI, and is served up with Unicorn and Apache. Having this kind of infrastructure means you're managing more processes than you can count on two hands.

The background jobs are run with Resque,[1] the scheduled jobs are run with `cron`, and the long-running jobs use Ruby daemons,[2] which are monitored by monit because they tend to crash.[3] It's going to take some time to figure out which component is the culprit because you have no centralized management interface. Standing up a new server will take almost as long because the infrastructure is so complex. But the website has to get back online if you're going to stay in business.

The problem I've just described is all too common. It has happened to everyone from small startups to large companies that use Rails to serve millions of requests. Their infrastructure is complex, and the myriad components are difficult to manage because they're heterogeneous and decentralized in nature. Even worse, Rubyists have become comfortable with this way of doing things, and some may think it's the only way of doing things. But that's not the case.

The recent growth and increased adoption of the Java Virtual Machine (JVM) as a platform for Ruby applications has opened many new doors. Deployment strategies that weren't possible with MRI Ruby are now an option because of the JVM's built-in management tools and support for native operating system threads. Ruby programmers can leverage these features by deploying their applications on JRuby.

It's common for Ruby programmers to think that JRuby deployment will look identical to deployment with MRI Ruby (that is, running lots of JVM processes

1. https://github.com/resque/resque
2. http://daemons.rubyforge.org/
3. http://mmonit.com/monit/

behind a load balancer and putting all asynchronous background jobs in a separate process). On the other hand, Java programmers tend to deploy JRuby applications the same way they deploy Java applications. This often requires lots of XML and custom build configurations, which negate many of the benefits of a more dynamic language such as Ruby. But there are much better options than both Ruby and Java programmers are used to.

In this book, you'll explore the most popular and well-supported methods for deploying JRuby. You have a surprising amount of flexibility in the processes and platforms to choose from, which allows Ruby and Java programmers to tailor their deployments to suit many different environments.

The No-Java-Code Promise

You won't have to write any Java code as you work your way through this book. That's not what this book is about. It's about deploying Ruby applications on the JVM. The technologies and tools you'll learn about in this book hide the XML and Java code from you. As the JRuby core developers like to say, "[They] write Java so you don't have to."[4]

You may want to include some Java code in your application. Or you may want to make calls to some Java libraries. That's entirely your choice. If you want to write your programs exclusively in Ruby and deploy them on the Java Virtual Machine—as so many of us do—then go ahead.

There are many reasons to deploy Ruby applications on the JVM, and using Java libraries and APIs is just one of them. In this book, you'll learn how to get the most out of the JVM without writing any Java code.

What's in This Book?

Over the course of this book, you're going to work on an application like the one described at the beginning of the preface. You'll port it to JRuby, add some new features, and simplify its infrastructure, which will improve its ability to scale.

The application's name is Twitalytics, and it's a rich Twitter client. (As you probably know, Twitter is a social networking website that's used to post short status updates, called *tweets*.) Twitalytics tracks an organization's tweets, annotates them, and performs analytic computations against data captured in those tweets to discover trends and make predictions. But it can't handle its current load.

4. http://vimeo.com/27494052

Twitalytics has several background jobs that are used to stream tweets into the application, perform analytics, and clean up the database as it grows. In addition, it receives a large volume of HTTP requests for traditional web traffic. But doing this on MRI means running everything in separate processes, which consumes more resources than its infrastructure can handle.

You'll begin working on the app in Chapter 1, *Getting Started with JRuby*, on page 1. You'll learn what makes JRuby a better deployment platform and why it's a good fit for this application. Then you'll extract a microservice from the Twitalytics monolith, port it to JRuby, and package it into an archive file with the Warbler gem. But before you can deploy it, you'll need to create an environment where it can run.

In Chapter 2, *Creating a Deployment Environment*, on page 17, you'll set up a containerization layer based on Docker and provision it with some essential components. You'll also learn how to automate this process to make it more reliable and reusable. You'll create a new server for each deployment strategy used in this book, and being able to reuse your configuration will save you time and prevent errors. In fact, this environment will apply not only to Twitalytics but to any JRuby deployment, so you're likely to reuse it on the job.

Once you've completed the production server setup, you'll be ready to deploy. You'll learn how JRuby deployment differs from the more common practice of traditional Ruby application deployment and how containerization technologies like Docker can simplify the process. In additional to using Docker, you'll deploy to the cloud on the Heroku platform as a service.

The Warbler gem gives you a quick way to get started with JRuby. But it's just a stepping-stone on your path to better performance. As the book progresses, you'll improve your deployment strategy by running Twitalytics on some other JRuby web servers.

The next chapter, Chapter 3, *Deploying a Rails Application*, on page 29, is dedicated to the Puma web server. Puma allows you to deploy applications much as you would with MRI-based Rails applications. But you'll find that JRuby reduces the complexity of this kind of deployment environment while increasing its reliability and portability. You'll deploy the Puma-based Rails app using both Docker and Heroku. The resulting architecture will be friendly and familiar to Rubyists.

But you still won't be making the most of what the JVM has to offer. To do that, you'll need a new kind of platform.

In Chapter 5, *Deploying JRuby in the Enterprise*, on page 73, you'll learn about a Ruby application server. You'll use TorqueBox, a server based on the popular JBoss application server, to run Twitalytics. This kind of deployment is unique when compared to traditional Ruby deployments because it provides a complete environment to run any kind of program, not just a web application. You'll learn how this eliminates the need for external processes. In the end, you'll have the most advanced deployment environment available to any Ruby application.

An overview of each strategy covered in this book is listed here:

	Warbler	Puma	TorqueBox
Internals	Jetty	Pure-Ruby	JBoss AS
Deployment type	WAR file	Traditional	Mixed
Docker deployment	Yes	Yes	Yes
Heroku deployment	Yes	Yes	Yes
Background jobs	No	No	Yes

Deciding on the right platform for each application is a function of these attributes. But getting an application up and running on one of these platforms is only a part of the job. You also need to keep it running. Fortunately, one of the many advantages of JRuby is the built-in JVM tooling.

Chapter 6, *Managing a JRuby Application*, on page 87 presents some tools for monitoring, managing, and configuring a deployed JRuby application. These tools are independent of any deployment strategy and can be used to monitor the memory consumption, performance, and uptime of any Java process. The information you gain from these tools can be used to improve the performance of JRuby, which you'll learn in Chapter 7, *Tuning a JRuby Application*, on page 109. You'll learn about different kinds of memory and the various knobs you can turn to optimize how the JVM allocates that memory. You'll even learn how to change garbage collectors and benchmark them.

In Chapter 8, *Monitoring JRuby in Production*, on page 123, you'll learn how to capture the same kind of metrics from a production runtime. You'll use some third-party apps to instrument your code, capture performance data, and log errors. Finally, Chapter 9, *Using a Continuous Integration Server*, on page 133 will introduce a tool for producing reliable and consistent deployments.

Twitalytics is a Rails application, and you'll use this to your advantage as you deploy it. But all of the server technologies you'll use work equally well with

any Rack-compliant framework (such as Sinatra[5]). In fact, the steps you'll use to package and deploy Twitalytics would be identical for these other frameworks. Warbler, Puma, and TorqueBox provide a few hooks that make deploying a Rails application more concise in some cases (such as automatically packaging bundled gems). But the workflow is the same.

When you encounter Rails-specific features in this book, be aware that this is only for demonstration purposes and not because the frameworks being used work exclusively with Rails. Rails works with these servers because it's Rack based.

What's Not in This Book?

This book won't teach you how to write code in the Ruby language. You'll write a bit of Ruby code in the course of this book, but you won't learn about specific features of the Ruby language. In particular, this book doesn't cover continuations, `ObjectSpace`, fibers, and other topics that have subtle differences when applied to JRuby. This book is specifically about *deploying* JRuby applications and how JRuby affects your production environments.

Other topics not addressed include zero-downtime deployments, database migrations, the asset pipeline, and content delivery networks (CDN). These are important aspects of Ruby web application development, but they're not notably different between MRI and JRuby. You can learn about these topics in books on the Ruby language and Rails. The same concepts will apply to JRuby.

Who Is This Book For?

This book is for programmers, system administrators, and DevOps[6] professionals who want to use JRuby to power their applications but aren't familiar with how this new platform will change their infrastructure.

You're not required to have any experience with JRuby. This book is written from the perspective of someone who is familiar with MRI-based Ruby deployments but wants a modern deployment strategy for their applications. Some of the concepts we'll discuss may be more familiar to programmers with Java backgrounds, but it's not required that you have any experience with Java or its associated technologies.

5. http://www.sinatrarb.com/
6. http://en.wikipedia.org/wiki/DevOps

Conventions

The examples in this book can be run on Linux, Mac, Windows, and many other operating systems. But some small changes to the command-line statements may be required for certain platforms.

We'll use notation from bash, which is the default shell on Mac OS X and many Linux distributions. The $ prompt will be used for all command-line examples. Windows command prompts typically use something like C:\> instead, so when you see a command like this

```
$ bundle install
```

you'll know not to type the dollar sign and to read it like this:

```
C:\> bundle install
```

Most commands will be compatible between Windows and bash systems (such as cd and mkdir). In the cases where they're not compatible, the appropriate commands for both systems will be spelled out. One case in particular is the rm command, which will look like this:

```
$ rm temp.txt
$ rm -rf tmp/
```

On Windows this should be translated to these two commands, respectively:

```
C:\> del temp.txt
C:\> rd /s /q tmp/
```

Another Unix notation that's used in this book is the ~ (tilde) to represent a user's home directory. When you see a command like this

```
$ cd ~/code/twitalytics
```

you can translate it to Windows 10 as this command:

```
C:\> cd C:\Users\yourname\code\twitalytics
```

On earlier versions of Windows, the user's home directory can be found in the Documents and Settings directory. You can also use the %USERPROFILE% environment variable. Its value is the location of the current user's profile directory.

Other than these minor notation changes, the examples in this book are compatible with Windows by virtue of the Java Virtual Machine.

Getting the Source Code

You're ready to set up the Twitalytics application. Start by downloading the source code from http://pragprog.com/titles/jkdepj2/source_code. Unpack the downloaded

file and put it in your home directory. This will create a `code` directory and inside that will be a `twitalytics` directory, which contains the baseline code for the application (in other words, the MRI-based code).

But you're not quite ready to run this code with JRuby. It needs to be ported first. You'll learn how to do that in the coming chapters.

Online Resources

Several online resources can help if you're having trouble setting up your environment or running any of the examples in this book.

For Java-related problems, the Java Community has forums[7] and numerous Java-related articles.

For JRuby-related problems, the official JRuby website[8] has links to several community outlets. The most useful of these are the mailing list[9] and the #jruby IRC channel on FreeNode.[10]

For TorqueBox-related problems, there are a mailing list,[11] extensive documentation,[12] and the #torquebox IRC channel on FreeNode.

7. https://community.oracle.com/community/java
8. http://jruby.org/community
9. https://github.com/jruby/jruby/wiki/MailingLists
10. http://freenode.net/
11. http://torquebox.org/community/mailing_lists/
12. http://torquebox.org/documentation/

CHAPTER 1

Getting Started with JRuby

JRuby is a high-performance platform that can scale to meet demand without the headaches of an MRI-based deployment. Those headaches are often the result of running a dozen or more processes on a single server that all need to be monitored, balanced, and occasionally restarted. JRuby avoids these problems by simplifying the architecture required to run an application. In this chapter, you're going to port a microservice to JRuby so that you can take advantage of this simplicity and scalability. But in order to run the microservice in production, you'll need a way to deploy it. For this, you'll use Warbler.[1]

Warbler is a gem used to package source code into an archive file you can deploy without the need for complicated configuration management scripts. This makes the process more flexible, portable, and faster.

In *Preface*, on page xi, you were introduced to Twitalytics, a Ruby on Rails app that needs help. Its infrastructure is too complex, and it can't handle the volume of requests the site is receiving. You don't have time to port the daemons and background jobs to a new framework, but you need to get one high-traffic HTTP service deployed on JRuby. If you can do that, you'll be able to handle lots of concurrent requests without hogging the system's memory. Later in the book, you'll consume this service from the main Rails app that makes up the bulk of Twitalytics.

Your time constraints make Warbler a great solution. It won't maximize your use of the JVM, but it will allow you to take advantage of the most important parts. You'll be able to service all of your site's web requests from a single process without changing much code. The drawback is that you'll have to

1. https://github.com/jruby/warbler