The Complete Guide to Rails Performance course teaches you how to diagnose and optimize production Ruby on Rails applications. The course covers performance across the entire stack - frontend, backend, databases, environment, deployment and more. The Complete Guide to Rails Performance course is available online. Included in the course is **70+ lessons and 100+ action items**, access to **live Q&A**, over **70 screencasts**, **Ebook** versions of all material, and an **audiobook** recording for learning on the go.

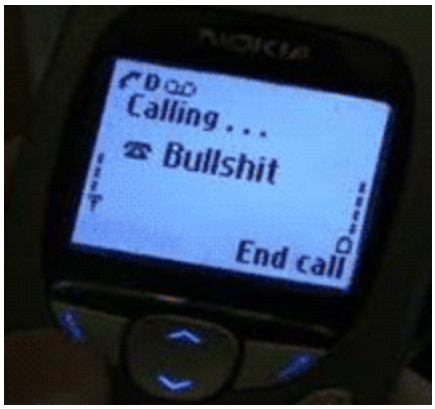
Performance is hard.

Why are some (even *most*) production Rails applications extremely slow?

They didn't start that way. Your app started as a lean, mean web machine.

When did pages start taking 5 seconds to load? Searches bog down, your caches are bursting at the seams, and your server is constantly out of memory.

Where did it all go wrong?



Hacker News will tell you it's because Ruby is fatally flawed. It's a slow language, doomed to failure in a land of Go, Javascript, Erlang and others. That's BS.

Look at Shopify, Github and Basecamp - all Rails applications in the top 1000 in the world, according to Alexa. How do these Rails applications achieve sub-100 millisecond response times at massive (dare I say, Web™) Scale?

Is it black magic? Is DHH sacrificing chickens in his backyard?

The secret is that Rails apps aren't slow by default - they die a slow death by a thousand papercuts. This course is about each one of those thousand papercuts.

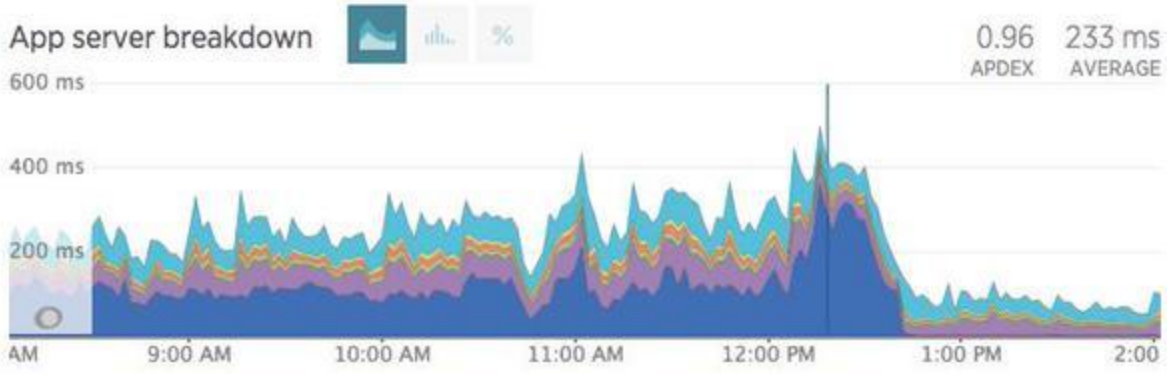
But you don't have a team of 30 or more experienced programmers, like Shopify, Github and Basecamp do. And let's be real - you're not DHH. Maybe you're a startup CTO with a business to run - you can't spend hundreds of hours tracking down performance bugs.

Maybe you've got a side project that's suffering under new growth and server load.

Maybe you *are* a bigger dev shop but you're looking for ways to train up your team on the latest performance techniques.

You just want someone to tell you what the problem areas are and how to fix them.

Almost every Rails application suffers from the same performance problems. This guide is about identifying and solving those problems.



What it looks like when you solve a problem.

Introducing The Complete Guide to Rails Performance

My name is Nate Berkopec. I'm a Rails developer.

Over the past 4 years, I've seen dozens of different Rails codebases. Many were for well-funded startups, from Ivy-League incubators like YCombinator or 500Startups. Others were the passion projects of developers looking to start a side business.

But I've never met a Rails application that couldn't be faster.

Performance isn't easy. Delivering an HTML response to the browser actually involves hundreds of technologies and levels of the stack.

Any of these levels or stacks can go wrong, causing a slowdown in your application:

- **The browser**: HTML, CSS and JavaScript, rendering quirks and browser preloaders.
- **The network**: SSL/TLS, HTTP, HTTP/2, TCP, WebSockets.
- **The Ruby stack:** Rails, the numerous gems you use in your application, your runtime, your memory allocator.
- **The database**: Redis, Memcache, SQL and NoSQL.
- **The server**: Virtualized environments have their own concerns. Will Docker solve all of your problems?

This makes performance optimization something like looking for a needle in a haystack - any one of a thousand things could be the bottleneck for your application. Where do you even start?

Worse, no full-stack developer that has to write code for a living can keep on top of all of these developments at once. No one has ever written a guide for optimizing Rails applications that covers the entire stack.

Wouldn't it be amazing if you could look over the shoulder of an experienced, full-stack Rails developer and watch as they diagnose and solve the performance problems of one of the most popular Rails applications in the world?

That's exactly what I've done.

Watch me optimize RubyGems.org, and apply the lessons back to your own site.

I've created what is basically an enormous checklist for speeding up Rails applications:

- Implemented 26 strategies that have been tested by more than 1 million unique users.
- Identified the 4 strategies that account for the most performance impact. These strategies have solved 80% of performance problems in most applications.
- Used more than 20 different SaaS providers, performance-oriented Rubygems, backend service providers and more and found the perfect technical stack to make the entire performance optimization process simple (even if you're a new developer or someone with a deep technical background).

These strategies have been tested and proven. But not just on my clients' sites.

While preparing this course, I've implemented each of my performance strategies on Rubygems.org, a top-20,000 website whose API you use every time you install a Ruby gem.

You'll get to watch, through screencasts and video lessons, exactly how (and more importantly, why) I did it.

Rubygems.org, one of the most popular Rails applications in the world, serves as your example and guide to each and every one of the strategies in this guide.

Think of it as an opportunity to pair program with an experienced Rails developer and work on a high-scale application.

You'll see the before and after, comparing the impact of each strategy through actual production metrics. It's broken down into simple, step-by-step processes that cover every layer of a Rails application's stack.

At the end of each lesson is a simple checklist so you can implement what you've learned. There's no guesswork. Just follow the plan.

What Topics Does This Course Cover?

This course covers:

- **Profiling and Benchmarking** - how to identify what areas are causing bottlenecks. Don't waste your time by "spraying and praying", optimizing every thing on your site. Learn how to identify what areas account for 80% of the slowdown.
- **Frontend** - I cover all the performance problems you can encounter in Javascript, HTML and CSS. In addition, we'll deep dive on HTTP and how the browser works, so that you can reduce your frontend load times by more than 80%.
- **Backend** - We'll discuss everything in the Ruby and Rails world that can make your application faster - what gems to use, which to avoid, and how to write Ruby that's zippy quick. You'll learn about how Rails works while avoiding the common pitfalls that make Rails apps slow.
- **Environment** - This is truly a full-stack book - we'll discuss asset delivery, DNS, server hardware and more.

10 More Things You'll Learn in this Course:

1. Will JRuby (or Rubinius or Opal) make your application faster?
2. Which application server (Unicorn, Puma, Passenger, etc.) is best for my application?
3. How can I make sure I'm not over-scaled and running too many servers?
4. How can I debug a slow ActiveRecord query?
5. What special areas do I need to look at as my application scales?
6. Which cache backend should I choose?
7. How can I track down and fix a memory leak?
8. How can I use Turbolinks without breaking Google Analytics or any of my other Javascript?
9. How do I make my Rails application as fast as a Sinatra app?
10. What's the optimal order of elements in the `<head>` tag?

Class Details: Here Is What You Will Get

Here is what your course will include:

- The entire Complete Guide to Rails Performance system (4 modules with **over 70 lessons and 100+ action items**)
- Meticulous **step-by-step instruction** for every strategy you learn
- 6 live **Q&A Sessions**
- Access to **a private Slack channel** with me and your fellow participants.
- Over 70 **screencasts**
- **An interview series** with accomplished Rubyists
- **Kindle/Epub/Mobi** versions for reading on the go
- **Audiobook** versions of all written content

How will you read it? Immediately after purchasing, you'll be sent a private link where you can jump in and start the course immediately. You could literally be watching the first workshop 5 minutes from now.

How Much Does it Cost?

The entire Complete Guide to Rails Performance -- all strategies, checklists and screencasts -- are broken into four modules to ensure that you don't get overwhelmed and can easily and methodically chip away at your Rails application's performance.

Starter Package

One flat payment of $299

- The complete and unabridged Complete Guide to Rails Performance.
- Access to the private community of your fellow course participants.
- ~~Dozens of hours of video and audio content.~~

Web-Scale™ Package

One flat payment of $399

- The complete and unabridged Complete Guide to Rails Performance.
- Access to the private community of your fellow course participants.
- Dozens of hours of video and audio content.

Corporate Package

Ask about pricing.

- A corporate license for the complete and unabridged Complete Guide to Rails Performance, providing access for your entire team. No per-seat rigmarole.
- Dozens of hours of video and audio content.
- Access to the private community of your fellow course participants.
- Private Q&A session with me and your team.
- 1-Day Performance Workshop where I pair with *your* team on *your* application.

If you have any questions about the course, you can email me directly at XXX or call my personal phone at XXX-XXX-XXXX.

Note: The Complete Guide to Rails Performance is currently being pre-sold for a reduced priced and will be released in January 2016.

Guarantee

30-day 100% Satisfaction Guarantee. If you do what I show you, and don't get results, I will give you 100% of your money back. If you're not completely satisfied, I don't want your money.

Frequently Asked Questions

Is there anyone I can reach out during the course if I have questions about my site in particular?

Yes. All course participants receive access to a private Slack channel where you can discuss problems and solutions. I'll be sitting in that channel all day. I do not have a day job - making your site faster is my job.

Will this course continue to be updated? And will I get free access to future versions?

Yes. The software world changes quickly. Any changes to the course are free to you forever - you will never be asked to upgrade or buy "version two" or buy again when Rails 8 comes out.

How much more is in the course vs. what you already put out on your blog?

About 25% of the material in the course is broadly covered by my blog. However, even for material that I've covered on the blog already, this course goes much further in depth and expands on those posts.

Think of my blog as the introduction. This course is the college class.

All videos and screencasts are all-new material.

Do I need to be an advanced Rails programmer to understand this course?

No. This course assumes about 3 months of Rails experience, no more. I hate technical writing that assumes the reader is some kind of genius and doesn't explain (or even just link to an explanation) everything that's going on.

In addition, even if you're not *completely* sure you've understood a topic, you can ask me and your fellow participants on our private Slack channel.

Finally, if you buy the course and decide its over your head, I'll refund your money. No questions asked.

Can I afford this right now?

Think about it this way - can your business (or the business you work for) afford to be slow? Can you afford users quitting when they get frustrated with your site's speed?

In 5 years, will you wish your site was slower? No. You'll wish it was faster.

Does this course apply to my small site? I only get 10 requests per minute.

If you (or your customers) are not satisfied with the speed of your Rails application, this course will work for you. Not 100% will apply, of course, but 90% of it will.

What stack will be covered? What about frontend JS frameworks?

I'm going to focus the course on the typical Rails stack. IMO, that includes a SQL relational database. NoSQL is too far outside of my comfort area to speak meaningfully about it. If a lot of people ask me for it, maybe I can bring in an outside expert for that or something. I may include a specific section on Postgres, because it has several unique features and it's so widely used. Docker - yes. JS frameworks - not specifically (I won't tell you how to optimize React, for example), but I will cover the specific needs of an API-only application.

Is this course appropriate for legacy applications?

I've worked on a lot of what I consider "legacy" applications (2+ year codebases). Those are the ones that tend to be slow, not the greenfield ones, so I'd say this course *focuses* on legacy applications. My only caveat is I'm not going to talk about optimizing previous major versions of anything - Rails 3, Ruby 1.9, etc.