RELIABLY DEPLOYING RAILS APPLICATIONS

Reliable, repeatable deployment & provisioning

BY BEN DIXON

Reliably Deploying Rails Applications

Hassle free provisioning, reliable deployment

Ben Dixon

This book is for sale at http://leanpub.com/deploying_rails_applications

This version was published on 2021-04-02

Contents

Intro

Preface to the fifth edition (2021)

The first edition of Reliably Deploying Rails applications was released in November 2013 and was based on Ubuntu 12.04 and Ruby 1.9.2. Since then a lot has changed in the world of application deployment. The shift towards devops has hugely increased the number of people exposed to the deployment process, CI and SSL has become ubiquitous and containerisation has rapidly gained in popularity.

But even amongst all this change, the basic requirement, to be able to spin up a VPS, deploy a Rails application to it and then largely forget about it, seems to live on. I remain a huge fan of Docker, Kubernetes and the ecosystems that surround them, but I've yet to find anything that comes close to the simplicity and reliability of provisioning a VPS with Chef and then using Capistrano to deploy a Rails application to it.

Systems I built 10 years ago have been running on this setup with minimal maintenance requirements on the infrastructure side and little or no downtime. And with the like of Hetzner Cloud providing perfectly usable virtual severs for less than 5 dollars a month, you can serve a surprising amount of traffic for an astonishingly low cost.

The fifth edition updates to the stack to cover Ruby 3.0. Rails 6 and Ubuntu 20.04 LTS. It adds automatic SSL with LetsEncrypt, migrates the application server from Unicorn to Puma and standardises on userspace systemd for managing the persistence of services without requiring sudo access. It will include chapters on using Capistrano to deploy to common CI systems and modern backups with Restic.

When I wrote the first edition 8 years ago, I thought at most, some 10's of people would read it, so I wrote it mainly for the enjoyment of the exercise. Since then thousands of people have purchased it, and even more gratifyingly, hundreds have gotten in touch with suggestions, improvements and stories of how they're using it and what they're building.

I can't thank everyone who's purchased this book and gotten in touch enough, and I hope the fifth edition continues to enable people to reliably and affordably deploy Rails applications while keeping their time, focus and money on building amazing things.

Ben

Current status of the fifth edition

As with the previous editions, the fifth edition has been released as a work in progress via Leanpub. This means that the fifth edition is not yet finished. If you purchase it now, you'll receive lifetime free updates of this and future editions as they are completed. Anybody who purchased previous editions will also receive these updates for free.

The chapter list below indicates the current status of each chapter with whether or not it has been updated yet for the fifth edition.

At a high level the "critical path" as been updated, so the sections on setting up the server and deploying to it with Capistrano are completed, with updated production ready sample code. So if your goal is "I need to get a Rails application deployed now", the book is ready for you.

What remains to complete is updating the supplementary chapters which go into the detail of what's happening behind the scenes and how to further customise the configuration. These will be completed over the coming months.

The purpose of this book

This book will show you from start to finish how to:

- Setup a VPS from Scratch
- Automate setting up additional servers in minutes
- Use Capistrano to deploy reliably
- Automate boring maintenance tasks

If you've got applications on Heroku which are costing you a fortune, this will provide you with the tools you need to move them onto a VPS.

If you're already running your app on a VPS but the deploy process is flaky - it sometimes doesn't restart or loads the wrong version of the code - this book provides a template for making the process robust.

I've spent hundreds of hours combing through blog posts, documentation and tweaking configuration files. This has got me to the stage where deploying to a VPS is as easy as - in fact often easier than - deploying to Heroku. If you want to do the same, this book will save you a lot of time.

Who is this book not for

This book is not for people who want to learn to deploy containerised Rails applications to the likes of Kubernetes. I remain a huge fan of containers and the ecosystem around them, but this book remains focused on the battle tested, simple as possible method of provisioning a VPS and deploying to it with Capistrano.

If you're in a hurry

If what you need is just to get a Rails application deployed and working as quickly as possible then:

- Start with Chapter 4, installing tools

- Then complete Chapter 5, the Chef Quick Start to setup the server
- Finally complete Chapter 18, the Capistrano quickstart to configure your Rails application for deployment

The entire process should take less than half an hour and at the end of it you'll have a fully functional deployment to a VPS. Otherwise read on!

About Me

I've been developing web applications for over fifteen years, over the last few years specialising in Rails development and deployment as well as Elixir and Kubernetes. I'm co-founder & CTO of Catapult[1] a venture backed global SaaS platform for maximising worker engagement.

Previously I was the technical lead at a health and fitness startup who provide the timetabling for many of the UK's leisure operators as well as producing a globally recognised iOS app (Speedo Fit) for swimmers.

As part of these projects I've dealt with everything from the usual rapid growth from 10's of requests per minute to 10's per second to more unusual challenges such as expanding infrastructure into China and debugging obscure indexing issues.

I spoke about deploying Rails applications at Railsconf 2014 and integrating Docker with Rails in 2015. In 2015 I also spoke at Refresh! conf in Tallinn about common mistakes people make when deploying and scaling web applications.

Why This Book Exists

I came to Rails having worked a lot in PHP and then Python/ Django. I was converted to Rails by the way the convention over configuration orientated approach removed much of the boilerplate I was accustomed to writing, leaving me free to focus on the real functionality of the application.

The first real hiccup was when I came to deploy an application to something other than Heroku. This became a necessity to keep costs down on side projects with background jobs and on several larger production projects where more flexibility was needed.

Having Googled extensively on how to deploy Rails applications to a VPS and found many tutorials documenting the commands to be entered by hand, I went down this route. This felt slow and repetitive. Somehow however carefully I documented the process, it never quite worked the next time I needed to setup a server.

This grated compared to the rest of my work-flow where anything I needed on multiple projects was simply abstracted into a gem and re-used, but I persevered and by and large it worked. Throughout the process I talked to others at local Ruby groups and other companies and they seemed to be following a similar process.

Eventually the inevitable happened and a production server failed completely, necessitating a complete rebuild.

[1]https://www.catapult.com

Unfortunately at the time I was in rural France on a rare holiday where the only usable WiFi signal was obtained by sitting half in the wardrobe in the attic of a small cottage. After spending nearly 12 hours in a wardrobe, typing in commands over a connection barely capable of maintaining an SSH session, I realised this had to be a problem someone had already solved.

I then discovered configuration management, experimented with Chef, Puppet and Ansible and eventually settled on Chef. In a surprisingly short amount of time, I had a simple Chef configuration which allowed me to configure a Rails ready server with just a few local commands, with a near guarantee it would be functionally identical to all previous ones.

This guarantee meant I could also use a standard Capistrano configuration to deploy to it. Suddenly deploying a Rails application to a VPS for the first time went from something that took at least a day to something that took just a few minutes.

Talking to others about why they weren't using such a system, it became clear that configuration management was considered the realm of "Ops" with too steep of a learning curve for most developers who just wanted to deploy the odd application here and there. This didn't tally with my experience at all, Chef was just Ruby and the time taken to learn the basics had been barely longer than it usually took me to setup a server by hand.

I started doing more consulting work for companies wanting to get up and running with configuration management and sharing my notes on the process with colleagues and friends for their own projects. Someone suggested this would make a useful book. And here we are!

I've tried to strike a balance between providing sample code which works out of the box to get up and running quickly whilst still giving enough detail on the rationale behind it and how each component works so that anyone reading this will be able to gradually develop a template tailored to them.

Intended Audience

This book is intended for people who develop Ruby on Rails applications and have to be involved with or, completely manage, the infrastructure and deployment of these applications.

Whether deploying applications is new to you or you've been doing it by hand for a while and want to move to a more structured approach, I hope this book will be useful to you.

Pre-requisites

It's assumed anyone reading this is already proficient in Ruby and Rails development.

Some basic knowledge of the unix command line is assumed, in particular that you can:

- Use SSH to connect to remote servers
- Navigate around the file system from the shell (cd, ls, dir, mv, rm etc)
- Have a basic understanding of web architecture. E.g. what a server is, how to setup DNS etc.

Structure of Part 1 - Setting up the server

Part 1 focuses on automated provisioning and uses Chef as the configuration management tool.

Chapter 2 - The Stack

An overview of the components which will make up our production Rails configuration, their purpose and a high level look at the rationale between choosing each one.

This chapter has been updated for the fifth edition.

Chapter 3 - Tools and Terminology

A brief look at why using a tool to automate the setting up of servers is important and some high level terminology we'll encounter when using this books automation tool of choice; Chef.

This chapter has been updated for the fifth edition.

Chapter 4 - Installing Tools

How to install Chef and supporting tools using ChefDK and a brief look at how ChefDK works.

This chapter has been updated for the fifth edition.

Chapter 5 - Quick Start

A chapter for the "read the instructions afterwards" types among us. This chapter provides step by step instructions for setting up a fully working, reproducible Rails server using the books sample code and a few simple commands. You'll have a server setup and ready within 30 minutes (and most of that is just waiting for Ruby to compile!).

This chapter has been updated for the fifth edition.

Chapter 6 - Creating a new project

How to create a Chef project from scratch. Think `rails new` but for server configurations.

This chapter has been updated for the fifth edition.

Chapter 7 - Using Knife

How to use Knife to interact with Chef so that we can setup our desired node end state and apply changes.

This chapter has been updated for the fifth edition.

Chapter 8 - Creating a Chef Cookbook

A detailed guide to how to create custom Chef cookbooks. Cookbooks are like gems but for re-usable bits of functionality on a server. We'll begin with the commands we'd type in by hand to install a piece of software on the server and convert this into the simplest possible cookbook to automate the process. We'll then iterate on this to take advantage of some of Chefs more powerful features.

This chapter has not yet been updated for the fifth edition, the concepts and primitives remain the same, but the specific example used needs to be updated.

Chapter 9 - Node and Role Definitions

First a look at the concept of a "node definition", a JSON file which defines everything about a server to be created. Then a look at how Chef allows us to create re-usable pieces of functionality which can then be applied to multiple servers or re-used in future projects using "roles".

This chapter has not yet been updated for the fifth edition, the concepts and primitives remain the same, but it builds heavily on Chapter 8 so will need updating following that.

Chapter 10 - A Template for Rails Servers

A more detailed introduction to the sample configuration which was used in Chapter 4's quick-start to setup a fully working server in just a few minutes.

This chapter has not yet been updated for the fifth edition.

Chapter 11 - Basic Server Setup

The bare bones components needed on any server, how - and when - to setup automatic package updates, how to have the systems time kept automatically updated and why so many people get caught out by locales.

This chapter has not yet been updated for the fifth edition.

Chapter 12 - Security

Starting off with a look at some common security gotchas when deploying to a fresh VPS. Then some more detail around how to lock down SSH access, manage firewall rules and automatically set up users and public keys.

This chapter has not yet been updated for the fifth edition.

Chapter 13 - Rails Server Setup

Making sure we have some basic packages required for installing common gems, then a more detailed look at how to install Ruby and make sure we have the correct version available to our Rails application.

This chapter has not yet been updated for the fifth edition.

Chapter 14 - Monit

When something crashes, in a perfect world, it will fix itself. This chapter covers how to use Monit to automatically monitor the health of services and restart them when they fail. In the event this fails, how to have Monit alert us via email so we can intervene and save the day.

This chapter has not yet been updated for the fifth edition.

Chapter 15 - Nginx

Nginx will be the first port of call when a request comes in. How to setup virtual hosts, serve static assets and a look at why we need Nginx at all.

This chapter has not yet been updated for the fifth edition.

Chapter 16 - PostgreSQL

How to setup PostgreSQL, manage the different types of authentication, ensure we have the desired version and manage importing and exporting data.

This chapter has not yet been updated for the fifth edition.

Chapter 17 - Redis and Memcached

A brief chapter on the two simplest parts of our stack. How to install Redis and Memcached, configuring whether or not they're bound to localhost and managing the maximum size they can reach.

This chapter has not yet been updated for the fifth edition.

Structure of Part 2 - Deploying to the server

Chapter 18 - Capistrano Quickstart

Like chapter 5, a chapter for those of us who prefer to read the instructions later. This chapter covers the minimum steps need to deploy an existing Rails application to our fresh new server.

This chapter has been fully updated for the fifth edition.

Chapter 19 - Deploying With Capistrano

A much more detailed introduction to how to deploy to our new server with Capistrano 3 whilst using only core Capistrano gems.

This chapter has not yet been updated for the fifth edition.

Chapter 20 - Writing Custom Capistrano Tasks

Our goal is to automate everything about interacting with our server. Eventually there will be something we need to do which no-one else has automated in a way we can re-use. This chapter covers writing custom Capistrano tasks. Once we've mastered this we can automate almost every interaction with our server imaginable.

This chapter has not yet been updated for the fifth edition although only minor changes are expected.

Chapter 21 - Unicorn Configuration and Zero Downtime

How to configure deployment so that when we deploy, there's a seamless switch from one version to the next, with no gap while the new version starts. Includes a detailed section on causes of potential problems when setting this up and how to troubleshoot if it doesn't work reliably.

This chapter is due to be removed entirely in the fifth edition and replaced with a chapter on Puma. Zero downtime deployment is configured out of the box in the Quickstart chapter.

Chapter 22 - Virtualhosts and SSL

A deeper look at Nginx virtualhosts which control where requests are routed after they reach NGinx. Then a look at how to setup SSL and manage certificate rotation.

This chapter has not yet been updated for the fifth edition. The section on SSL will be removed as this is now covered automatically in the quick start section.

Chapter 23 - Sidekiq

Sidekiq is one of the most popular and efficient background job processing libraries in the Rails ecosystem. Here we cover how to integrate starting and stopping Sidekiq workers as part of the deployment process and how to setup monitoring so they'll be automatically restarted in the event of a failure.

This chapter has not yet been updated for the fifth edition. Deploying Sidekiq is now covered in this Quickstart, this chapter will be updated to cover some advanced configuration.

Chapter 24 - Automated Backups

It's an unfortunate fact that eventually a server will fail. In this chapter we setup automatic database backups to Amazon S3 (or any other number of destinations) to ensure that in the event of a complete server failure, our data is recoverable.

This chapter has not yet been updated for the fifth edition, it will be replaced with a Restic based backup solution.

Chapter 25 - Deploying with CI

This chapter has not yet been written

Version

You are reading Version 5.1.0 of this book. The major and minor (x.x) numbers correspond to the version number in the sample code. The final number refers to minor improvements such as typos. Having purchased once, you can download the latest version of this book from Leanpub at any time.

The Stack

Overview

The stack used for examples in this book is just one of many possible configurations. I've picked a combination of components I've found to be the most common across clients but it's intended to be an example only.

If part of your intended stack is not included here don't despair, all of the general principles will apply.

If you know, or can find instructions on how to install the component in question, the techniques provided for automating with Chef can be adapted.

Ubuntu 20.04 LTS

Ubuntu has become an increasingly common distribution in a server environment. The primary reason I recommend it is the level of community support available.

The sheer number of people of all abilities using it means that you are almost never the first person to have a particular problem so, in 99% of cases, a quick Stack Overflow search will point you in the right direction.

For the remaining 1% the community is extremely friendly and helpful.

Nginx

Nginx is a web server and reverse proxy known for its high performance and small memory footprint.

A key benefit of nginx is that due to its event driven architecture, its memory usage is extremely predictable, even under heavy loads. This makes it ideal for projects that may begin on a small VPS for testing and grow to much larger machines as they scale.

Architecturally, when requests come into the server, they are handled by Nginx which then passes them back to our application server (for example Puma or Unicorn) which runs the rails application and returns the response.

Postgresql

Postgresql is a traditional relational database. It has increasingly become the default for new rails applications using relational databases, in part because it is the default database supported by Heroku. It's my preferred database primarily because with the addition of native JSON support it is increasingly able to combine the benefits of a traditional RDMS with those of a NoSQL solution such as Mongo.

Ruby + rbenv

This book covers and has been tested with Ruby 2.7.x and Ruby 3.0.x. Whilst a lot of the techniques may work with more exotic flavors such as JRuby, I have minimal experience with these and so will not cover them directly.

My preference for managing and installing ruby versions on production servers is rbenv. This is primarily because I find rbenvs operation simple to understand and therefore troubleshoot.

Redis

Redis is an extremely fast key value store. It performs well in use cases such as caching and API rate limiting as well as more advanced areas such as calculating intersections between series.

It has a few gotchas such as its behavior when its max memory limit is reached which are covered in the section on configuring Redis.

Memcached

Similar to Redis but entirely in memory (reboot the machine and you lose everything that was in Memcached). Great for caching, and like Redis, incredibly simple to install and maintain.

Why This Stack

These components cover a majority of the applications I've encountered over the last few years. I'm generally fairly neutral in the "which is the best stack argument." There are lots of other great combinations out there and the concepts in this book will apply to most of them.

Adapting to your stack

If your stack is not covered above, don't worry. The aim of this book is to show you how easy it is to use Chef to automate the provisioning of any Rails stack, the components here are just examples. By the end of the book you'll be able to either find community Chef recipes or write your own to setup almost anything.

Chef is used only for setting up the server, not for deployment. The second half of this book - deploying with Capistrano - will be relevant no matter which configuration management tool you use.

Tools And Terminology

Introduction

The ‘simplest’ way to provision a new server is to create a new VPS on something like Linode or Digital Ocean, login via SSH and start apt-get’ing the packages you need, dropping into vim to tweak config files and adding a few custom package sources where newer versions are needed.

When something new is needed we ssh back in, install or upgrade a package, building the server up in layers. Somewhere there’s a text file or wiki page with “all the commands we need” written down should we ever need to do it again.

This approach has a few key pitfalls;

1. It’s very hard to keep track of what’s been done. With the best will in the world the text file doesn’t quite get all of the commands. Nobody realises until the time comes to provision a new server (often under adverse conditions) and running all the commands in the file doesn’t quite yield a working server.
2. It’s slow (and therefore expensive). Even if the process is perfectly documented, someone is still needed to sit and run the commands. This is repetitive, boring work, but will often need to be done by an engineer whose time could be better spent working on the product itself.
3. It doesn’t scale. Having an engineer type in a list of commands might, with a lot of discipline, hold together when there’s only one server involved. Expand that to five or six servers and the cracks will soon start to show.

Automation

The goal of this section and of this book as a whole, is to take the manual processes and automate them. As a general rule, any process which I’d expect to repeat more than once a year in the life-cycle of deploying and managing infrastructure, I try and automate.

Automation relates to an approach more than it relates to any particular tool. If we’re doing anything which involves running more than one command or sshing into a remote server more than once in a month, it’s probably worth stopping and thinking, “how can this be automated?”

The benefits are often visible after a remarkably short period of time. Automating server deployments not only makes disaster recovery easier, it makes the creation of accurate test and staging environments easier, making the testing of new deployments easier and more efficient and so decreasing downtime.

Automating the copying of production databases from production to development and staging environments makes it more likely developers will use current data in their tests. This makes tests in development more meaningful and so increases productivity and decreases costs.

Finally automation is usually easier than expected. Once we've mastered Chef for automating provisioning tasks and Capistrano for automating deployment tasks and subsequent interactions with production and staging servers, it becomes clear that the time taken to automate additional tasks is rarely significantly more than the time taken to perform them once.

Introducing Chef

Chef is an automation platform made by Opscode which uses a Ruby DSL (Domain Specific Language) to represent the commands required to provision a server in a reusable format.

Typical Chef Usage

A traditional Chef toolchain is made up of:

- Chef Server - This is where information about the intended setup of all nodes being managed resides
- Chef Client - This runs on every single node being managed and is responsible for executing the changes required to bring the node into the desired state
- Knife - This is the command line utility used to update the Chef Server with changes to the desired state of one or more nodes

So a minimal workflow might look like this:

1. Use Knife to tell Chef Server to `bootstrap` a node. This means installing Chef Client on the remote node and creating a "node definition" file for that node on the Chef server
2. Use Knife to tell Chef Server that we want certain "recipes" or "roles" added to the "run list" - for example the "postgesql::server" "recipe" from the "postgresql" "cookbook" - of the node
3. Use Knife to set "attributes" on our "node" which customise how the recipes in our "run list" behave. For example setting the port Postgres listens on.
4. Use Knife (or Berkshelf, explained later) to upload our Cookbooks to the Chef Server
5. Use Knife to tell Chef Server to "converge" our node
6. Chef Server will then connect to the Chef Client on the target node, copy across the relevant cookbooks and have Chef Client execute them
7. These cookbooks will lead to commands being executed which setup whatever was defined in the run-list - in the example from above, Postgres.

Terminology

The above example included some important terminology:

Bootstrap	The process of setting up a node for the first time to be used with Chef
Node	A server being managed by Chef
Recipes	A definition of how to get a target machine into a particular state. This is often how to install and configure a particular piece of software
Attributes	Values (variables) which customise the behaviour of recipes
Cookbooks	A grouping of related recipes
Roles	Re-usable groupings of run list entries and attributes
Run List	The list of roles and recipes which should be applied to a node
Converge	The process of applying the roles & recipes to a node

Working without a Chef server - Chef Zero

The above may sound excessive when all we want to do is setup one VPS for one Rails application. In scenarios like these we can use Chef Zero to complete the same workflow without any need for a central Chef Server.

Chef Zero replaces Chef Solo which was used in earlier editions of this book to accomplish the same thing. If you're already using Chef Solo, upgrading is fairly painless, with repository structure remaining almost identical.

Chef Zero runs an on-demand, in-memory Chef Server instance which persists its data to a local directory in the form of JSON files. We can interact with this using knife as if it was a central Chef server while in fact everything is being run from our local development machine.

This has the substantial advantage of meaning that everything we learn about how to interact with and use Chef, is applicable to both small standalone projects using Zero and larger projects using Chef Server.

This book will focus entirely on Chef Zero but the techniques describe work identically in a Chef Server environment.

Leveraging the Community

One possibility when working with Chef is that we create our own cookbooks and recipes which define how to install and configure everything we need. In practice it is often much more more efficient to use a Cookbook which has already been created by the community and then customise its behaviour using attributes.

In the same way there is a gem for almost anything, there is a cookbook for almost everything, from installing Postgres and NGINX to configuring locales and managing third party log aggregation tools.

Berkshelf

Berkshelf is like Bundler for these third party cookbooks. It allows us to specify a list of dependencies and their versions, and have them be downloaded automatically from a central index.

The chapter “Creaing a New Project & Berkshelf” covers the operation of this in more detail.

Installing Tools

Overview

This section covers installing the tools which are used throughout this book. All subsequent chapters assume that the tools covered in this chapter have already been installed.

Installing Tools

Install Chef Workstation

Install Chef Workstation from https://downloads.chef.io/products/workstation and ensure that running the command `chef` from a terminal does not return a command not found error.

Chef Workstation includes:

- The command line tool `chef` which is primarily used for generating new chef assets, for example repositories (projects) and cookbooks
- The cookbook dependency manager Berkshelf, used for managing both our own and third part cookbooks
- The command line interface Knife which we use for managing the interaction between Chef and the nodes we are configuring
- Chef Zero, a tool which allows us to use Chef without a separate centralised server

At time of writing the Chef Workstation `dmg` for MacOS can be downloaded with:

```
curl https://packages.chef.io/files/stable/chef-workstation/21.2.303/mac_os_x/11.\
0/chef-workstation-21.2.303-1.x86_64.dmg --output /tmp/chef-workstation.dmg
hdiutil attach /tmp/chef-workstation.dmg
```

And the `deb` for Debian based linux distributions with:

```
curl https://packages.chef.io/files/stable/chef-workstation/21.2.303/debian/10/ch\
ef-workstation_21.2.303-1_amd64.deb --output /tmp/chef-workstation.deb
sudo dpkg -i /tmp/chef-workstation.deb
```

This version of the book has been tested with version 21.2.303 of Chef Workstation.

We can check that the installation has been successful by executing the command `chef`, if it doesn't give a command not found error, then the installation succeeded.

Installing Knife Zero

Knife is the command line utility which we'll use to:

- Tell Chef what the desired end state of our nodes is
- Tell Chef to actually make changes to a node

This is installed by default by Chef workstation so nothing else is needed here.

Knife Zero adds some extra commands to knife to make it easy to interact with Chef Zero which is what allows us to use Chef without a centralised Chef server. We'll cover these commands in more detail in "Using Knife".

Chef Workstation sets up an isolated Ruby and Gem environment for itself. This means that we cannot simply `gem install knife-zero` even though Knife, like a majority of Chef components is distributed in the form of a Ruby Gem.

Instead, the `chef` command line utility we installed as part of Chef Workstation provides the `chef gem install` command which allows us to install any gem into the Chef Workstation enviornment.

To install Knife Zero simply execute:

```
chef gem install knife-zero
```

In a folder without a `Gemfile` and `Gemfile.lock` in it. If there is a `Gemfile` in the current folder when we execute the above command, we make encounter hard to debug "bundler not found" errors.

Note that when installing the gem to the Chef Ruby environment, we may see output like:

```
WARNING:  You don't have /home/ben/.chefdk/gem/ruby/2.7.0/bin in your PATH,
          gem executables will not run.
```

which can be safely ignored.

We can test that installation of `knife-zero` has worked by executing:

```
knife zero
```

Which should result in output such as:

```
FATAL: Cannot find subcommand for: 'zero'
Available zero subcommands: (for details, knife SUB-COMMAND --help)

** ZERO COMMANDS **
knife zero apply QUERY (options)
knife zero bootstrap [SSH_USER@]FQDN (options)
knife zero chef_client QUERY (options) | It's same as converge
knife zero converge QUERY (options)
knife zero diagnose # show configuration from file
```

We now have the core tools we're going to work with installed and are ready to proceed either directly to the "Quick Start" or, if we prefer to read the instructions, to "Create a New Project".

How To Buy

This book is published via LeanPub. You can purchase it here: https://leanpub.com/deploying_rails_applications/ with a no questions asked, 45 day money back guarantee.

Once you've purchased it, you'll automatically receive lifetime free updates as soon as they're published. The book is regularly updated to ensure it stays current as new versions of the tools covered are released.

If you've got any questions about what's in the book or feedback on what you've seen, please feel to reach out to me via email, ben@talkingquickly.co.uk[2] or on twitter; @talkingquickly[3].

Thanks for reading,

Ben.

[2]mailto:ben@talkingquickly.co.uk
[3]https://www.twitter.com/talkingquickly