**Rails Heroku Tutorial**

**by Daniel Kehoe**

*Last updated 30 August 2012*

Here’s how to set up an app with Rails 3.2 and Ruby 1.9.3 on Heroku. Heroku provides low cost, easily configured Rails 3.2 application hosting.

An article, [Heroku Isn’t for Idiots](http://rdegges.com/heroku-isnt-for-idiots), describes why you might want to use Heroku. And a discussion on Quora explains, [What are the potential downsides of using Heroku?](http://www.quora.com/What-are-the-potential-downsides-of-using-Heroku) (not many).

If you’re new to Rails, see [What is Ruby on Rails?](http://railsapps.github.io/what-is-ruby-rails.html), the book [Learn Ruby on Rails](http://learn-rails.com/learn-ruby-on-rails.html), and recommendations for a [Rails tutorial](https://tutorials.railsapps.org/rails-tutorial).

**RailsApps Examples and Tutorials**

This is a guide for developers using the example apps from the RailsApps repository. The [RailsApps project](http://railsapps.github.io/) provides open source apps and detailed tutorials for Rails developers. Next time you build an app, try using our [Rails Composer](http://railsapps.github.io/rails-composer/) tool; it’s a better way to build a starter app.

[Follow on Twitter](http://www.twitter.com/rails_apps)Follow [@rails\_apps](http://twitter.com/rails_apps) on Twitter for updates and timely Rails tips.

**Before You Start**

To deploy an app to Heroku, you must have a Heroku account. Visit <http://heroku.com/> to set up an account. Or you can use someone else’s account (such as a client’s or employer’s if you have login credentials); see details below.

Before you start, you should review the advice offered in the article [Installing Rails](http://railsapps.github.io/installing-rails.html) to be sure your development environment is set up correctly on your local machine.

**Ruby 1.9.3 on Heroku**

Heroku’s newest stack, “Celadon Cedar,” supports Ruby 1.9.3 and Rails 3.2. As of 9 May 2012, Heroku officially supports Ruby 1.9.3 (see [Multiple Ruby Version Support on Heroku](http://blog.heroku.com/archives/2012/5/9/multiple_ruby_version_support_on_heroku/)).

You configure the Heroku environment to use Ruby 1.9.3 by specifying the Ruby version in your application Gemfile. This requires [Bundler](http://gembundler.com/) 1.2.0 or newer. You must have Bundler 1.2.0 or newer if you plan to deploy to Heroku (see instructions below).

Heroku provides the most secure patch level of whatever minor version number you request. So if you specify Ruby 1.9.3, Heroku will provide ruby-1.9.3-p194.

*Note:* You may have found outdated instructions elsewhere. To clarify: Installing Ruby 1.9.3 on Heroku previously (before March 19, 2012) required installation of the heroku-labs plugin. The heroku-labs plugin is deprecated and its functionality is now incorporated in the heroku client gem. Before May 9, 2012, installing Ruby 1.9.3 on Heroku required the [user\_env\_compile](http://devcenter.heroku.com/articles/labs-user-env-compile) feature; this is now deprecated. Furthermore, it is no longer necessary to set the RUBY\_VERSION variable with $ heroku config:add RUBY\_VERSION=ruby-1.9.3-p125 or to set the Heroku PATH config variable to include “bin” when you specify the default Ruby 1.9.3 for the Cedar stack. Prior to Rails 3.1.0.rc5, Heroku required an additional gem therubyracer-heroku. This gem is no longer needed for Rails 3.2 and should not be used.

The following Heroku articles are helpful:

* [Heroku Ruby Support](https://devcenter.heroku.com/articles/ruby-support)
* [Getting Started with Rails 3.x on Heroku/Cedar](https://devcenter.heroku.com/articles/rails3)
* [Rails 3.1+ Asset Pipeline on Heroku Cedar.](https://devcenter.heroku.com/articles/rails3x-asset-pipeline-cedar)

**Updating Bundler for Heroku**

These instructions assume you are using rvm, the [Ruby Version Manager](https://rvm.io/). Bundler and rubygems-bundler are installed in rvm’s “global” gemset.

First, update to the [newest version of Rubygems-Bundler](http://rubygems.org/gems/rubygems-bundler):

|  |
| --- |
| $ rvm gemset use global  $ gem update rubygems-bundler  $ gem list |

Update to the [newest version of Bundler](http://rubygems.org/gems/bundler):

|  |
| --- |
| $ rvm gemset use global  $ gem update bundler  $ gem list |

If you encounter any issues when updating rubygems-bundler and Bundler, review the detailed instructions in the article [Installing Rails](http://railsapps.github.io/installing-rails.html).

Switch back to your application-specific gemset to proceed:

|  |
| --- |
| $ rvm gemset use <whatever> |

**Preparing a Gemfile for Heroku**

You’ll add the Heroku gem and specify a Ruby version in your application’s gemfile.

**Specifying a Ruby Version**

Specify a Ruby version before specifying any gems:

|  |
| --- |
| source 'http://rubygems.org'  ruby '1.9.3'  gem 'rails', '3.2.13' |

As explained above, specifying ruby '1.9.3' requires Bundler 1.2.0 or newer.

**Heroku Gem**

Add the Heroku gem to your application’s Gemfile [(check rubygems.org for the latest heroku gem)](http://rubygems.org/gems/heroku):

|  |
| --- |
| gem "heroku" |

Install your gems with:

|  |
| --- |
| $ bundle install |

If you haven’t done so, add your public key after installing the heroku gem so that you can use git to push or clone Heroku app repositories. For more information, see Heroku’s documentation [Managing Your SSH Keys](http://devcenter.heroku.com/articles/keys).

*Note:* As an alternative to installing the Heroku gem, Heroku offers a stand-alone “Heroku Toolbelt” which installs the Heroku command-line client and the Git revision control system. You can use either the Heroku gem or the Heroku Toolbelt. For more information, see Heroku’s documentation [Installing the Heroku CLI](http://devcenter.heroku.com/articles/heroku-command). It’s probably easier to simply use the Heroku gem.

**Database Configuration**

Heroku uses PostgreSQL as its default database. Heroku does not support SQLite or MySQL, the databases most commonly used for local development.

**Use SQLite Locally and PostgreSQL on Heroku**

As a general practice, you should use the same environment for local development as you do for production, which means using the same database systems locally and in production. In practice, with ActiveRecord isolating the database system, this may not be critically important.

If you don’t want to develop using PostgreSQL locally, you can set up your Gemfile to use SQLite for development and PostgreSQL for production. To switch from SQLite to PostgreSQL for deployment to Heroku, edit your Gemfile and change this line:

|  |
| --- |
| gem 'sqlite3' |

To this:

|  |
| --- |
| group :development, :test do  gem 'sqlite3'  end  group :production do  gem 'pg'  end |

Run bundle install --without production to update your gems. The flag --without production allows you to skip local installation of the pg gem; otherwise, you will have to install PostgreSQL locally (the pg gem won’t install if PostgreSQL is not installed).

**Use PostgreSQL Locally and on Heroku**

If you’d like to use PostgreSQL both locally and on Heroku for production, edit your Gemfile and change this line:

|  |
| --- |
| gem 'sqlite3' |

To this:

|  |
| --- |
| gem 'pg' |

You’ll have to install PostgreSQL locally before running bundle install.

**The MongoDB Alternative**

MongoDB simplifies development by eliminating the schemas and migrations required by SQLite, MySQL, or PostgreSQL. The [MongoHQ](http://addons.heroku.com/mongohq) add-on is offered by Heroku and provides a hosted MongoDB datastore.

If you decide to try MongoDB, take a look at the example apps (with detailed tutorials) that combine [Mongoid with Devise](https://github.com/RailsApps/rails3-mongoid-devise) or [Mongoid with OmniAuth](https://github.com/RailsApps/rails3-mongoid-omniauth).

**Configure Mongoid**

By default, the file **config/mongoid.yml** contains this:

|  |
| --- |
| development:  # Configure available database sessions. (required)  sessions:  # Defines the default session. (required)  default:  # Defines the name of the default database that Mongoid can connect to.  # (required).  database: <something>  # Provides the hosts the default session can connect to. Must be an array  # of host:port pairs. (required)  hosts:  - localhost:27017 |

To use MongoHQ with your app, add the following to the file **config/mongoid.yml**:

|  |
| --- |
| <% if ENV['MONGOHQ\_URL'] %>  production:  sessions:  default:  uri: <%= ENV['MONGOHQ\_URL'] %>  options:  skip\_version\_check: true  safe: true  <% end %> |

**Specifying a Rails Server**

By default, when you run rails server locally, Rails launches the simple Webrick web server. For production apps Heroku recommends using a more robust webserver such as [Thin](http://code.macournoyer.com/thin/). You could also use [Unicorn](http://unicorn.bogomips.org/), [Rainbows!](http://rainbows.rubyforge.org/index.html), or [Puma](http://puma.io/).

To use Thin, add the gem to your Gemfile:

|  |
| --- |
| gem 'thin' |

If you want to continue to use Webrick as your local web server, with Thin on Heroku, set up your Gemfile like this:

|  |
| --- |
| group :production do  gem 'thin'  end |

Run bundle install --without production to update your gems.

**Procfile Not Needed**

Heroku recommends adding a Procfile to your app. For simplicity, there’s no need to create a Procfile if you use Thin. If you declare the Thin gem in your Gemfile, Heroku sets up an appropriate web process without requiring a Procfile.

Procfile is a mechanism for declaring what commands are run when your web app runs on the Heroku platform. Heroku recommends it for greater control and flexibility over your app. If you add a Procfile, you can specify Thin as your web process in the Procfile.

**Precompile Assets**

Rails 3.1 introduced the asset pipeline which enables proper organization of CSS and JavaScript. Assets must be available as Javascript and CSS files when anyone visits your website. That means multiple files in various formats (CoffeeScript, SASS, etc.) must be compiled to simple Javascript and CSS files. During development, assets are compiled on the fly so you can easily make changes and see the results.

For deployment on Heroku, you have three options:

* do nothing and allow Heroku to attempt to precompile assets when you deploy to Heroku (slug compilation)
* do nothing and allow Heroku to attempt to compile assets when the app is generated (runtime compilation)
* precompile the assets before deployment to Heroku

While runtime asset compilation will work, it should be used as a last resort. Using runtime compilation will require Rails to compile your assets each time a dyno boots up increasing the wait time for a new dyno to become available. If you do nothing and rely on Heroku to perform slug compilation (when you deploy to Heroku), you may find Heroku is unable to compile the assets and will default to runtime compilation. It’s best to precompile the assets before deployment to Heroku.

Heroku doesn’t check files in the **config/initializers** folder before attempting to precompile assets. That’s fine if you precompile assets before deploying to Heroku. Or if you don’t use the initializer files to set constants that are used in the assets files. You can avoid these issues by adding this configuration parameter to the **config/application.rb** file:

|  |
| --- |
| # Heroku requires this to be false  config.assets.initialize\_on\_precompile=false |

To precompile assets before deployment to Heroku:

|  |
| --- |
| rake assets:precompile |

See the article [Rails 3.1+ Asset Pipeline on Heroku Cedar.](https://devcenter.heroku.com/articles/rails3x-asset-pipeline-cedar).

**Commit to Git**

Your application must be committed to a Git repository before you can use Heroku. See [Git and Rails](http://railsapps.github.io/rails-git.html) to learn how to set up Git for your application.

Store the modified application in the Git repository:

|  |
| --- |
| $ git commit -am "Configured for deployment to Heroku" |

**Heroku Account**

To deploy an app to Heroku, you must have a Heroku account. Visit <http://heroku.com/> to set up an account. Heroku will ask you for your email address; it’s advisable to use the same email address you use for your GitHub account (see the article [Git and Rails](http://railsapps.github.io/rails-git.html)).

**Using a Client’s Account**

You may want to deploy an app to an account that belongs to someone other than yourself (for example, your employer or a client). If you are using a client’s or employer’s Heroku account, you won’t be able to push your application to Heroku until you’ve added yourself as a collaborator. After you create an app on Heroku, log in to your client’s account on Heroku. Under “My Apps” you’ll see the name of the app you’ve created; click “General Info” and you’ll find a form where you can add yourself as a collaborator.

**Deploying for Multiple Accounts**

David Dollar’s [heroku-accounts plugin](https://github.com/ddollar/heroku-accounts) will help if you must use multiple accounts on Heroku. With the heroku-accounts plugin, you can configure each application to deploy to the appropriate account.

To install the heroku-accounts plugin:

|  |
| --- |
| heroku plugins:install git://github.com/ddollar/heroku-accounts.git |

To add an account locally (which you’ll call “work”):

|  |
| --- |
| $ heroku accounts:add work --auto |

To list accounts that are set up locally:

|  |
| --- |
| $ heroku accounts |

To set an account to use for a project:

|  |
| --- |
| # in project root  heroku accounts:set work |

**Create Your Application on Heroku**

Use the Heroku create command to create and name your new app.

|  |
| --- |
| $ heroku create myapp |

Replace myapp with something unique. Heroku demands a unique name for every hosted application. If it is not unique, you’ll see an error, “name is already taken.”

As of June 20, 2012, Heroku’s [default stack changed to Cedar](http://blog.heroku.com/archives/2012/5/24/cedar_goes_ga). Before, you needed to specify $ heroku create myapp --stack cedar; that’s no longer necessary.

If you don’t specify your app name (myapp in the example above), Heroku will supply a placeholder name. You can easily change Heroku’s placeholder name to a name of your choice with the heroku apps:rename command (see [Renaming Apps from the CLI](https://devcenter.heroku.com/articles/renaming-apps)).

Don’t worry too much about getting the “perfect name” for your Heroku app. The name of your Heroku app won’t matter if you plan to set up your Heroku app to use your own domain name. You’ll just use the name for access to the instance of your app running on the Heroku servers; later, you’ll set up DNS to point your domain name to the app running on Heroku.

**Set Heroku Environment Variables**

Your application may be obtaining usernames, passwords, or API keys for external services from the Unix shell environment. It’s a recommended practice to avoid recording sensitive information in your application code where it might be exposed publicly on a GitHub repo.

After creating your Heroku app, you may need to set Heroku environment variables to provide the same data your application obtains from your local shell environment.

For example, for Gmail:

|  |
| --- |
| $ heroku config:add GMAIL\_USERNAME=myname@gmail.com GMAIL\_PASSWORD=secret |

Check Heroku Configuration

You can check that everything has been added correctly by running:

|  |
| --- |
| $ heroku info --app myapp |

**Using MongoDB**

If you are using MongoDB, you can use a Heroku add-on to deploy your app using the MongoHQ service. See [details about the service](http://addons.heroku.com/mongohq) and [details about installation](http://devcenter.heroku.com/articles/mongohq).

To use MongoDB with your app, enable the add-on with the following command:

|  |
| --- |
| $ heroku addons:add mongohq:free |

**Troubleshooting Problems Connecting to MongoHQ**

If you deploy your app and get the error message “failed to connect to any given host:port” or “Failed to connect to a master node at localhost:27017”, the **config/mongoid.yml** file may not have the correct MongoHQ connection parameters.

**Push Your Application to Heroku**

Push your application to Heroku:

|  |
| --- |
| $ git push heroku master |

Review the output carefully. If you precompiled assets before deploying, you should see:

-> Preparing app for Rails asset pipeline

Detected manifest.yml, assuming assets were compiled locally

*Note:* If you are using a client’s or employer’s Heroku account, you won’t be able to push your application to Heroku until you’ve added yourself as a collaborator. After you create the app on Heroku, log in to your client’s account on Heroku. Under “My Apps” you’ll see the name of the app you’ve created; click “General Info” and you’ll find a form where you can add yourself as a collaborator.

**Database Migrations**

You’ll need to run your database migrations (not applicable if you are using MongoDB):

|  |
| --- |
| $ heroku run rake db:migrate |

Initialize your application database if your application requires it:

|  |
| --- |
| $ heroku run rake db:seed |

These two commands can be combined as:

|  |
| --- |
| $ heroku run rake db:setup |

**PostgreSQL Database Reset**

Heroku doesn’t support rake db:reset.

Need to reset your Heroku PostgreSQL database? Here’s how:

|  |
| --- |
| $ heroku pg:reset SHARED\_DATABASE --confirm myapp |

Don’t forget to restart your application or you won’t see any changes:

|  |
| --- |
| $ heroku restart |

**Visit Your Site**

Open your Heroku site in your default web browser:

|  |
| --- |
| $ heroku open |

Your app will be running at <http://my-app-name.herokuapp.com/>.

**Enabling Email**

Unless you enable email on Heroku, you’ll get errors when your application tries to send email from Heroku.

Heroku offers several options for sending email from your Rails application. See Heroku’s documentation [Sending Email from Your App](http://devcenter.heroku.com/articles/smtp).

**Email Service Providers**

For a production application, you’ll need to use an email service provider for your transactional email such as [SendGrid](http://sendgrid.com/pricing.html) or [Mandrill](http://mandrill.com/).

**Gmail**

If you’re sending test emails during development, Gmail is the easiest option.

To use Gmail from Heroku, add the following to your *config/environments/production.rb* file:

|  |
| --- |
| config.action\_mailer.default\_url\_options = { :host => 'myapp.herokuapp.com' }  config.action\_mailer.delivery\_method = :smtp  config.action\_mailer.perform\_deliveries = true  config.action\_mailer.raise\_delivery\_errors = false  config.action\_mailer.default :charset => "utf-8"  config.action\_mailer.smtp\_settings = {  address: "smtp.gmail.com",  port: 587,  domain: "myapp.herokuapp.com",  authentication: "plain",  enable\_starttls\_auto: true,  user\_name: ENV["GMAIL\_USERNAME"],  password: ENV["GMAIL\_PASSWORD"]  } |

To avoid storing your Gmail username and password in your Git repository, use environment variables to pass the username and password to your application. See Heroku’s documentation [Configuration and Config Vars](http://devcenter.heroku.com/articles/config-vars).

Set your Gmail username and password as Heroku environment variables:

|  |
| --- |
| $ heroku config:add GMAIL\_USERNAME=no-reply@example.com GMAIL\_PASSWORD=please |

**Your Own Domain**

You’ll likely want to use your own domain name for your app.

See [Heroku’s article about custom domains](https://devcenter.heroku.com/articles/custom-domains) for instructions.

**Why “www” Matters**

For simplicity, you might want your web app running on a “bare domain” like <http://example.com/>. You can do it, but Heroku explains [Why You Should Avoid Naked Domains](https://devcenter.heroku.com/articles/avoiding-naked-domains-dns-arecords) and always use a [fully qualified domain name](http://en.wikipedia.org/wiki/Fully_qualified_domain_name) such as <http://www.example.com/>.

**Set Your Domain Name**

This command tells Heroku that your app should respond to requests to “www.mydomain.com”:

|  |
| --- |
| $ heroku domains:add www.mydomain.com |

**Set a DNS Record**

Domain Name Service ([DNS](http://en.wikipedia.org/wiki/Domain_Name_System)) routes requests for your domain to the servers that host your application. Your name service may be provided by your domain registrar (for example, GoDaddy) or a managed DNS provider (for example, [DNSimple](https://dnsimple.com/) or [Zerigo](http://www.zerigo.com/managed-dns)). You must visit your DNS provider’s website to set a CNAME record to route requests for “http://www.mydomain.com/” to your app running on Heroku.

Each DNS provider has a different interface for managing DNS records. This article doesn’t show you how to set a CNAME record.

After you have set a CNAME record, as soon as DNS records propagate (usually a few minutes but sometimes longer), your app should be running at <http://www.mydomain.com/>.

**Using Subdomains**

You’ll need to enable “wildcard domains” if you want your application to accommodate arbitrary subdomains (for example, a different subdomain for each user account),

|  |
| --- |
| $ heroku domains:add \*.mydomain.com |

Then, visit your DNS provider to set a CNAME record to point “\*.example.com” at “myapp.herokuapp.com”.

**Checking Subdomains**

You can check that everything has been added correctly by running:

|  |
| --- |
| $ heroku info --app myapp |

As soon as DNS has propagated, your local computer will show that the domain name resolves properly:

|  |
| --- |
| $ host www.example.com  www.example.com has address ... |

**Troubleshooting**

When you get errors, troubleshoot by reviewing the log files:

|  |
| --- |
| $ heroku logs |

If necessary, use the Unix tail flag to monitor your log files. Open a new terminal window and enter:

|  |
| --- |
| $ heroku logs -t |

to watch the server logs in real time.

**Where to Get Help**

Your best source for help with Heroku is [Stack Overflow](http://stackoverflow.com/questions/tagged/heroku). Your issue may have been encountered and addressed by others.

You can also check the [Heroku Dev Center](http://devcenter.heroku.com/) or the [Heroku Google Group](http://groups.google.com/group/heroku/).