# Getting Started with Rails 4.x on Heroku

Last updated 03 October 2016

### Table of Contents



The latest version of Rails available is **Rails 5 (https://devcenter.heroku.com/articles/getting-started-with-rails5)**. If you are starting a new application we recommend you use the most recently released version.

Ruby on Rails is a popular web framework written in Ruby (http://www.ruby-lang.org/). This guide covers using Rails 4 on Heroku. For information about running previous versions of Rails on Heroku, see Getting Started with Rails 3.x on Heroku (https://devcenter.heroku.com/articles/getting-started-with-rails3).

If you are already familiar with Heroku and Rails, reference the **simplifed Rails 4 on Heroku guide (https://devcenter.heroku.com/articles/rails4)** instead. For general information on how to develop and architect apps for use on Heroku, see **Architecting Applications for Heroku (https://devcenter.heroku.com/articles/architecting-apps)**.

For this guide you will need:

- Basic Ruby/Rails knowledge
- Locally installed version of Ruby 2.0.0+, Rubygems, Bundler, and Rails 4+
- Basic Git knowledge
- A Heroku user account: Signup is free and instant (https://signup.heroku.com/devcenter)

# Local Workstation Setup

Install the Heroku CLI (https://cli.heroku.com/) on your local workstation. This ensures that you have access to the Heroku command-line client (https://devcenter.heroku.com/categories/command-line), Foreman, and the Git revision control system. You will also need Ruby and Rails installed (http://guides.railsgirls.com/install).

Once installed, you'll have access to the `$ heroku` command from your command shell. Log in using the email address and password you used when creating your Heroku account:

> Note that `$` symbol before commands indicates they should be run on the command line, prompt, or terminal with appropriate permissions. Do not copy the `$` symbol.

```
$ heroku login
Enter your Heroku credentials.
Email: schneems@example.com
Password:
Could not find an existing public key.
Would you like to generate one? [Yn]
Generating new SSH public key.
Uploading ssh public key /Users/adam/.ssh/id_rsa.pub
```

Press enter at the prompt to upload your existing `ssh` key or create a new one, used for pushing code later on.

# Write your App

> To run on Heroku your app must be configured to use the Postgres database, have all dependencies declared in your `Gemfile` , and have the `rails_12factor` gem in the production group of your `Gemfile`

You may be starting from an existing app, if so upgrade to Rails 4 (http://edgeguides.rubyonrails.org/upgrading_ruby_on_rails.html#upgrading-from-rails-3-2-to-rails-4-0) before continuing. If not, a vanilla Rails 4 app will serve as a suitable sample app. To build a new app make sure that you're using the Rails 4.x using `$ rails -v` . You can get the new version of rails by running,

```
$ gem install rails -v 4.2.6 --no-ri --no-rdoc
Successfully installed rails-4.2.6
1 gem installed
```

Note: There may be a more recent version of Rails (https://rubygems.org/gems/rails/versions) available, we recommend always running the latest. You may want to run Rails 5 on Heroku (https://devcenter.heroku.com/articles/getting-started-with-rails5).

Then create a new app:

```
$ rails new myapp --database=postgresql
```

Then move into your application directory.

```
$ cd myapp
```

If you experience problems or get stuck with this tutorial, your questions may be answered in a later part of this document. Once you experience a problem try reading through the entire document and then going back to your issue. It can also be useful to review your previous steps to ensure they all executed correctly.

If already have an app that was created without specifying `--database=postgresql` you will need to add the `pg` gem to your Rails project. Edit your `Gemfile` and change this line:

```
gem 'sqlite3'
```

To this:

```
gem 'pg'
```

We highly recommend using PostgreSQL during development. Maintaining **parity between your development (http://www.12factor.net/dev-prod-parity)** and deployment environments prevents subtle bugs from being introduced because of differences between your environments. **Install Postgres locally (https://devcenter.heroku.com/articles/heroku-postgresql#local-setup)** now if it is not allready on your system.

Now re-install your dependencies (to generate a new `Gemfile.lock` ):

```
$ bundle install
```

You can get more information on why this change is needed and how to configure your app to run postgres locally see why you cannot use Sqlite3 on Heroku (https://devcenter.heroku.com/articles/sqlite3).

In addition to using the `pg` gem, you'll also need to ensure the `config/database.yml` is using the `postgresql` adapter.

The development section of your `config/database.yml` file should look something like this:

```
$ cat config/database.yml
# PostgreSQL. Versions 8.2 and up are supported.
#
# Install the pg driver:
#   gem install pg
# On OS X with Homebrew:
#   gem install pg -- --with-pg-config=/usr/local/bin/pg_config
# On OS X with MacPorts:
#   gem install pg -- --with-pg-config=/opt/local/lib/postgresql84/bin/pg_config
# On Windows:
#   gem install pg
#       Choose the win32 build.
#       Install PostgreSQL and put its /bin directory on your path.
#
# Configure Using Gemfile
# gem 'pg'
#
default: &default
  adapter: postgresql
  encoding: unicode
  # For details on connection pooling, see rails configuration guide
  # http://guides.rubyonrails.org/configuring.html#database-pooling
  pool: 5
```

Be careful here, if you omit the `sql` at the end of `postgresql` in the `adapter` section your application will not work.

# Welcome page

Rails 4 no longer has a static index page in production. When you're using a new app, there will not be a root page in production, so we need to create one. We will first create a controller called `welcome` for our home page to live:

```
$ rails generate controller welcome
```

Next we'll add an index page.

In file `app/views/welcome/index.html.erb` write:

```
<h2>Hello World</h2>
<p>
  The time is now: <%= Time.now %>
</p>
```

Now we need to have Rails route to this action. We'll edit `config/routes.rb` to set the index page to our new method:

In file `config/routes.rb` , on line 2 add:

```
  root 'welcome#index'
```

You can verify that the page is there by running your server:

```
$ rails server
```

And visiting http://localhost:3000 (http://localhost:3000) in your browser. If you do not see the page, use the logs that are output to your server to debug.

# Heroku gems

Heroku integration has previously relied on using the Rails plugin system, which has been removed from Rails 4. To enable features such as static asset serving and logging on Heroku please add `rails_12factor` gem to your `Gemfile`.

At the end of `Gemfile` add:

```
gem 'rails_12factor', group: :production
```

Then run:

```
$ bundle install
```

We talk more about Rails integration on our Ruby Support page (https://devcenter.heroku.com/articles/ruby-support#injected-plugins).

# Specify Ruby version in app

Rails 4 requires Ruby 1.9.3 or above. Heroku has a recent version of Ruby installed, however you can specify an exact version by using the `ruby` DSL in your `Gemfile`. For this guide we'll be using Ruby 2.

At the end of `Gemfile` add:

```
ruby "2.3.1"
```

You should also be running the same version of Ruby locally. You can verify by running `$ ruby -v`. You can get more information on specifying your Ruby version on Heroku here (https://devcenter.heroku.com/articles/ruby-versions).

# Store your App in Git

Heroku relies on git (http://git-scm.com/), a distributed source control managment tool, for deploying your project. If your project is not already in git first verify that `git` is on your system:

```
$ git --help
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]
```

If you don't see any output or get `command not found` you will need to install it on your system, verify that the Heroku CLI (https://cli.heroku.com/) is installed.

Once you've verified that git works, first make sure you are in your Rails app directory by running `$ ls` :

The output should look like this:

```
$ ls
Gemfile
Gemfile.lock
README.rdoc
Rakefile
app
bin
config
config.ru
db
lib
log
public
test
tmp
vendor
```

Now run these commands in your Rails app directory to initialize and commit your code to git:

```
$ git init
$ git add .
$ git commit -m "init"
```

You can verify everything was committed correctly by running:

```
$ git status
On branch master
nothing to commit, working directory clean
```

Now that your application is committed to git you can deploy to Heroku.

# Deploy your application to Heroku

Make sure you are in the directory that contains your Rails app, then create an app on Heroku:

```
$ heroku create
Creating app... done, safe-mesa-35727
https://safe-mesa-35727.herokuapp.com/ | https://git.heroku.com/safe-mesa-35727.git
```

You can verify that the remote was added to your project by running

```
$ git config --list | grep heroku
remote.heroku.url=https://git.heroku.com/safe-mesa-35727.git
remote.heroku.fetch=+refs/heads/*:refs/remotes/heroku/*
```

If you see `fatal: not in a git directory` then you are likely not in the correct directory. Otherwise you may deploy your code. After you deploy your code, you will need to migrate your database, make sure it is properly scaled and use logs to debug any issues that come up.

Deploy your code:

```
$ git push heroku master
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Ruby app detected
remote: -----> Compiling Ruby/Rails
remote: -----> Using Ruby version: ruby-2.3.1
remote: -----> Installing dependencies using bundler 1.11.2
remote:        Running: bundle install --without development:test --path vendor/bundle --binstubs vendo
remote:        Warning: the running version of Bundler is older than the version that created the lockf
remote:        Fetching gem metadata from https://rubygems.org/...........
remote:        Fetching version metadata from https://rubygems.org/...
remote:        Fetching dependency metadata from https://rubygems.org/..
remote:        Using json 1.8.3
remote:        Installing rake 11.1.2
remote:        Installing minitest 5.9.0
remote:        Installing i18n 0.7.0
remote:        Installing thread_safe 0.3.5
remote:        Installing erubis 2.7.0
remote:        Installing builder 3.2.2
remote:        Installing mini_portile2 2.0.0
remote:        Installing rack 1.6.4
remote:        Installing coffee-script-source 1.10.0
remote:        Installing mime-types-data 3.2016.0521
remote:        Installing arel 6.0.3
remote:        Installing execjs 2.7.0
remote:        Installing concurrent-ruby 1.0.2
remote:        Installing thor 0.19.1
remote:        Using bundler 1.11.2
remote:        Installing pg 0.18.4 with native extensions
remote:        Installing multi_json 1.12.1
remote:        Installing rails_serve_static_assets 0.0.5
remote:        Installing rails_stdout_logging 0.0.4
remote:        Installing sass 3.4.22
remote:        Installing tilt 2.0.4
remote:        Installing tzinfo 1.2.2
remote:        Installing rdoc 4.2.2
remote:        Installing nokogiri 1.6.7.2 with native extensions
remote:        Installing rack-test 0.6.3
remote:        Installing mime-types 3.1
remote:        Installing coffee-script 2.4.1
remote:        Installing uglifier 3.0.0
remote:        Installing sprockets 3.6.0
remote:        Installing rails_12factor 0.0.3
remote:        Installing activesupport 4.2.6
remote:        Installing sdoc 0.4.1
remote:        Installing mail 2.6.4
remote:        Installing rails-deprecated_sanitizer 1.0.3
remote:        Installing globalid 0.3.6
remote:        Installing activemodel 4.2.6
remote:        Installing jbuilder 2.4.1
remote:        Installing activejob 4.2.6
remote:        Installing activerecord 4.2.6
```

```
remote:        Installing rails-dom-testing 1.0.7
remote:        Installing loofah 2.0.3
remote:        Installing rails-html-sanitizer 1.0.3
remote:        Installing actionview 4.2.6
remote:        Installing actionpack 4.2.6
remote:        Installing railties 4.2.6
remote:        Installing sprockets-rails 3.0.4
remote:        Installing actionmailer 4.2.6
remote:        Installing rails 4.2.6
remote:        Installing coffee-rails 4.1.1
remote:        Installing jquery-rails 4.1.1
remote:        Installing sass-rails 5.0.4
remote:        Installing turbolinks 2.5.3
remote:        Bundle complete! 13 Gemfile dependencies, 53 gems now installed.
remote:        Gems in the groups development and test were not installed.
remote:        Bundled gems are installed into ./vendor/bundle.
remote:        Post-install message from rdoc:
remote:        Depending on your version of ruby, you may need to install ruby rdoc/ri data:
remote:        <= 1.8.6 : unsupported
remote:        = 1.8.7 : gem install rdoc-data; rdoc-data --install
remote:        = 1.9.1 : gem install rdoc-data; rdoc-data --install
remote:        >= 1.9.2 : nothing to do! Yay!
remote:        Bundle completed (30.50s)
remote:        Cleaning up the bundler cache.
remote:        Warning: the running version of Bundler is older than the version that created the lockf
remote: -----> Preparing app for Rails asset pipeline
remote:        Running: rake assets:precompile
remote:        I, [2016-05-26T19:33:05.012902 #1067]  INFO -- : Writing /tmp/build_14c78c3c9f70b1508642
remote:        I, [2016-05-26T19:33:05.013422 #1067]  INFO -- : Writing /tmp/build_14c78c3c9f70b1508642
remote:        I, [2016-05-26T19:33:05.025799 #1067]  INFO -- : Writing /tmp/build_14c78c3c9f70b1508642
remote:        I, [2016-05-26T19:33:05.026153 #1067]  INFO -- : Writing /tmp/build_14c78c3c9f70b1508642
remote:        Asset precompilation completed (4.20s)
remote:        Cleaning assets
remote:        Running: rake assets:clean
remote:
remote: ###### WARNING:
remote:        No Procfile detected, using the default web server.
remote:        We recommend explicitly declaring how to boot your server process via a Procfile.
remote:        https://devcenter.heroku.com/articles/ruby-default-web-server
remote:
remote: -----> Discovering process types
remote:        Procfile declares types     -> (none)
remote:        Default types for buildpack -> console, rake, web, worker
remote:
remote: -----> Compressing...
remote:        Done: 32.2M
remote: -----> Launching...
remote:        Released v5
remote:        https://safe-mesa-35727.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/safe-mesa-35727.git
 * [new branch]      master -> master
```

It is always a good idea to check to see if there are any warnings or errors in the output. If everything went well you can migrate your database.

# Migrate your database

If you are using the database in your application you need to manually migrate the database by running:

```
$ heroku run rake db:migrate
```

Any commands after the `heroku run` will be executed on a Heroku dyno (https://devcenter.heroku.com/articles/dynos). You can obtain an interactive shell session by running `$ heroku run bash`.

# Visit your application

You've deployed your code to Heroku. You can now instruct Heroku to execute a process type. Heroku does this by running the associated command in a dyno (https://devcenter.heroku.com/articles/dynos) - a lightweight container which is the basic unit of composition on Heroku.

Let's ensure we have one dyno running the `web` process type:

```
$ heroku ps:scale web=1
```

You can check the state of the app's dynos. The `heroku ps` command lists the running dynos of your application:

```
$ heroku ps
=== web (Standard-1X): bin/rails server -p $PORT -e $RAILS_ENV (1)
web.1: starting 2016/05/26 14:33:18 -0500 (~ 15s ago)
```

Here, one dyno is running.

We can now visit the app in our browser with `heroku open`.

```
$ heroku open
```

You should now see the "Hello World" text we inserted above.

Heroku gives you a default web url for simplicty while you are developing. When you are ready to scale up and use Heroku for production you can add your own Custom Domain (https://devcenter.heroku.com/articles/custom-domains).

# View the logs

If you run into any problems getting your app to perform properly, you will need to check the logs.

You can view information about your running app using one of the logging commands (https://devcenter.heroku.com/articles/logging), `heroku logs`:

```
$ heroku logs
2016-05-26T19:32:21.684314+00:00 heroku[api]: Release v2 created by developer@example.com
2016-05-26T19:32:21.684314+00:00 heroku[api]: Enable Logplex by developer@example.com
2016-05-26T19:33:15.104930+00:00 heroku[api]: Release v3 created by developer@example.com
2016-05-26T19:33:15.104930+00:00 heroku[api]: Set LANG, RACK_ENV, RAILS_ENV, RAILS_SERVE_STATIC_FILES, 
2016-05-26T19:33:16.191292+00:00 heroku[api]: Release v4 created by developer@example.com
2016-05-26T19:33:16.191292+00:00 heroku[api]: Attach DATABASE (@ref:postgresql-contoured-80543) by deve
2016-05-26T19:33:17.032028+00:00 heroku[api]: Deploy 80e85a6 by developer@example.com
2016-05-26T19:33:17.031094+00:00 heroku[api]: Scale to web=1 by developer@example.com
2016-05-26T19:33:17.032120+00:00 heroku[api]: Release v5 created by developer@example.com
2016-05-26T19:33:17.700203+00:00 heroku[slug-compiler]: Slug compilation finished
2016-05-26T19:33:17.700197+00:00 heroku[slug-compiler]: Slug compilation started
2016-05-26T19:33:22.085200+00:00 heroku[web.1]: Starting process with command `bin/rails server -p 3465
```

You can also get the full stream of logs by running the logs command with the `--tail` flag option like this:

```
$ heroku logs --tail
```

# Dyno sleeping and scaling

By default, your app is deployed on a free dyno. Free dynos will sleep after a half hour of inactivity (if they don't receive any traffic). This causes a delay of a few seconds for the first request upon waking. Subsequent requests will perform normally. Free dynos also consume from a monthly, account-level quota of free dyno hours (https://devcenter.heroku.com/articles/free-dyno-hours) - as long as the quota is not exhausted, all free apps can continue to run.

```
$ heroku ps:scale web=1
```

To avoid dyno sleeping, you can upgrade to a hobby or professional dyno type as described in the Dyno Types (https://devcenter.heroku.com/articles/dyno-types) article. For example, if you migrate your app to a professional dyno, you can easily scale it by running a command telling Heroku to execute a specific number of dynos, each running your web process type.

# Console

Heroku allows you to run commands in a one-off dyno (https://devcenter.heroku.com/articles/one-off-dynos) - scripts and applications that only need to be executed when needed - using the `heroku run` command. Use this to launch a Rails console process attached to your local terminal for experimenting in your app's environment:

```
$ heroku run rails console
irb(main):001:0> puts 1+1
2
```

# Rake

Rake can be run as an attached process exactly like the console:

```
$ heroku run rake db:migrate
```

# Webserver

By default, your app's web process runs `rails server`, which uses Webrick. This is fine for testing, but for production apps you'll want to switch to a more robust webserver. On Cedar, we recommend Puma as the webserver (https://devcenter.heroku.com/articles/deploying-rails-applications-with-the-puma-web-server). Regardless of the webserver you choose, production apps should always specify the webserver explicitly in the `Procfile`.

First, add Puma to your application `Gemfile`:

At the end of `Gemfile` add:

```
gem 'puma'
```

Then run

```
$ bundle install
```

Now you are ready to configure your app to use Puma. For this tutorial we will use the default settings of Puma, but we recommend generating a `config/puma.rb` file and reading more about configuring your application for maximum performance by reading the Puma documentation (https://devcenter.heroku.com/articles/deploying-rails-applications-with-the-puma-web-server)

Finally you will need to tell Heroku how to run your Rails app by creating a `Procfile` in the root of your application directory.

## Procfile

Change the command used to launch your web process by creating a file called Procfile (https://devcenter.heroku.com/articles/procfile) and entering this:

In file `Procfile` write:

```
web: bundle exec puma -t 5:5 -p ${PORT:-3000} -e ${RACK_ENV:-development}
```

Note: The case of `Procfile` matters, the first letter must be uppercase.

We recommend generating a puma config file based on our Puma documentation (https://devcenter.heroku.com/articles/deploying-rails-applications-with-the-puma-web-server) for maximum performance.

Set the `RACK_ENV` to development in your environment and a `PORT` to connect to. Before pushing to Heroku you'll want to test with the `RACK_ENV` set to production since this is the enviroment your Heroku app will run in.

```
$ echo "RACK_ENV=development" >>.env
$ echo "PORT=3000" >> .env
```

You'll also want to add `.env` to your `.gitignore` since this is for local enviroment setup.

```
$ echo ".env" >> .gitignore
$ git add .gitignore
$ git commit -m "add .env to .gitignore"
```

Test your Procfile locally using Foreman:

```
$ gem install foreman
```

You can now start your web server by running

```
$ foreman start
18:24:56 web.1  | I, [2013-03-13T18:24:56.885046 #18793]  INFO -- : listening on addr=0.0.0.0:5000 fd=7
18:24:56 web.1  | I, [2013-03-13T18:24:56.885140 #18793]  INFO -- : worker=0 spawning...
18:24:56 web.1  | I, [2013-03-13T18:24:56.885680 #18793]  INFO -- : master process ready
18:24:56 web.1  | I, [2013-03-13T18:24:56.886145 #18795]  INFO -- : worker=0 spawned pid=18795
18:24:56 web.1  | I, [2013-03-13T18:24:56.886272 #18795]  INFO -- : Refreshing Gem list
18:24:57 web.1  | I, [2013-03-13T18:24:57.647574 #18795]  INFO -- : worker=0 ready
```

Looks good, so press Ctrl-C to exit and you can deploy your changes to Heroku:

```
$ git add .
$ git commit -m "use puma via procfile"
$ git push heroku master
```

Check `ps`, you'll see the web process uses your new command specifying Puma as the web server

```
$ heroku ps
=== web (Standard-1X): bundle exec puma -t 5:5 -p ${PORT:-3000} -e ${RACK_ENV:-development} (1)
web.1: starting 2016/05/26 14:34:03 -0500 (~ 15s ago)
```

The logs also reflect that we are now using Puma:

```
$ heroku logs
```

# Rails Asset Pipeline

There are several options for invoking the Rails asset pipeline (http://guides.rubyonrails.org/asset_pipeline.html) when deploying to Heroku. For general information on the asset pipeline please see the Rails 3.1+ Asset Pipeline on Heroku Cedar (https://devcenter.heroku.com/articles/rails-asset-pipeline) article.

The `config.assets.initialize_on_precompile` option has been removed is and not needed for Rails 4. Also, any failure in asset compilation will now cause the push to fail. For Rails 4 asset pipeline support see the Ruby Support (https://devcenter.heroku.com/articles/ruby-support#rails-4-x-applications) page.

# Troubleshooting

If you push up your app and it crashes ( `heroku ps` shows state `crashed` ), check your logs to find out what went wrong. Here are some common problems.

## Runtime dependencies on development/test gems

If you're missing a gem when you deploy, check your Bundler groups. Heroku builds your app without the `development` or `test` groups, and if you app depends on a gem from one of these groups to run, you should move it out of the group.

One common example using the RSpec tasks in your `Rakefile` . If you see this in your Heroku deploy:

```
$ heroku run rake -T
Running `bundle exec rake -T` attached to terminal... up, ps.3
rake aborted!
no such file to load -- rspec/core/rake_task
```

Then you've hit this problem. First, duplicate the problem locally:

```
$ bundle install --without development:test
…
$ bundle exec rake -T
rake aborted!
no such file to load -- rspec/core/rake_task
```

Now you can fix it by making these Rake tasks conditional on the gem load. For example:

## Rakefile

```
begin
  require "rspec/core/rake_task"
  desc "Run all examples"
  RSpec::Core::RakeTask.new(:spec) do |t|
    t.rspec_opts = %w[--color]
    t.pattern = 'spec/**/*_spec.rb'
  end
rescue LoadError
end
```

Confirm it works locally, then push to Heroku.

# Done

You now have your first application deployed to Heroku. The next step is to deploy your own application. If you're interested in reading more you can read more about Ruby on Heroku at the Devcenter (https://devcenter.heroku.com/categories/ruby).