Heroku Ruby Support

Last updated 07 September 2016

Table of Contents

Heroku is capable of running Ruby applications across a variety of Ruby implementations and includes support for framework-specific workflows.

This document describes the general behavior of Heroku as it relates to the recognition and execution of Ruby applications. For framework specific tutorials please see:

- Getting Started with Ruby on Heroku (https://devcenter.heroku.com/articles/getting-started-with-ruby)
- Getting Started with Rails 4 on Heroku (https://devcenter.heroku.com/articles/getting-started-with-rails4).

General support

The following support is provided, irrespective of the type of Ruby application deployed.

Activation

The Heroku Ruby Support will be applied to applications only when the application has a `Gemfile` in the root directory. Even if an application has no gem dependencies it should include an empty `Gemfile` to document that your app has no gem dependencies.

Particular actions, documented in subsequent sections, are taken depending on the type of application deployed, which is determined by the following rules:

- Presence of `Gemfile` indicates a Ruby application
- Presence of `config.ru` indicates a Rack application
- Presence of `config/environment.rb` indicates a Rails 2 application
- Presence of `config/application.rb` containing the string `Rails::Application` indicates a Rails 3 application

Libraries

The following libraries are used by the platform for managing and running Ruby applications and cannot be specified.

- Bundler v1.11.2: Application dependency resolution and management.

For more information on available settings see Bundler configuration (https://devcenter.heroku.com/articles/bundler-configuration).

Environment

The following environment variables will be set:

- `GEM_PATH => vendor/bundle/#{RUBY_ENGINE}/#{RUBY_ABI_VERSION}`
- `LANG => en-us`
- `PATH => bin:vendor/bundle/#{RUBY_ENGINE}/#{RUBY_ABI_VERSION}/bin:/usr/local/bin:/usr/bin:/bin`

`GEM_PATH` is set to the bundler gem vendor directory.

Process types

The following two process types are always made available:

```
rake: bundle exec rake
console: bundle exec irb
```

Build behavior

When an application is deployed, the build phase configures the underlying Rack or Rails application to use the provisioned database if a `config` directory exists, and a `RAILS_ENV` or `RACK_ENV` environment variable is present. Prior to ActiveRecord 4.1, which is used by Rails 4.1+, a `database.yml` file will be created . If a `database.yml` file already exists, it will be replaced up to ActiveRecord 4.1. The `database.yml` file is created as Ruby code that dynamically creates its output by parsing the `DATABASE_URL` environment variable. In ActiveRecord 4.1+ `DATABASE_URL` support is baked in. For more information see configuring database connections (http://guides.rubyonrails.org/configuring.html#configuring-a-database) and the pull request that added DATABASE_URL support (https://github.com/rails/rails/pull/13582)

JRuby

For JRuby, you can customize the options passed to the JVM during the build by setting the config var `JRUBY_BUILD_OPTS` . A common value is `--dev` , which optimizes the runtime for short processes like those executed by Bundler and Rake.

Ruby versions

Heroku makes a number of different Ruby implementations available. You can configure your app to select a particular runtime.

Default Ruby version for New Apps

If your `Gemfile` does not contain a `ruby` entry, you will get MRI `2.2.4`. Default rubies are locked into the app until you specify a Ruby version. For example, if you were previously on `1.9.3`, you will continue to stay on `1.9.3`. If you're locked into Ruby `2.0.0` or lower, you'll get the latest patchlevel available for that Ruby version.

The default Ruby for new apps, will be one minor version behind the most recent one. For instance, when Ruby `2.4.0` comes out, Ruby `2.3.x` will become the default.

Supported runtimes

Heroku supports the following Ruby versions and the associated Rubygems. A supported version means that you can expect our tools and platform to work with a given version. It also means you can receive technical support. Here are our supported Ruby versions:

MRI:

- `2.0.0` : patchlevel 648, Rubygems : `2.0.14.1`
- `2.1.10` : patchlevel 492, Rubygems : `2.2.5`
- `2.2.5` : patchlevel 319, Rubygems : `2.4.5.1`
- `2.3.1` : patchlevel 112, Rubygems : `2.5.1`

When a Ruby version reaches EOL security patches will no longer be available. We highly recommend running on a version of Ruby that is actively supported by Ruby core. Ruby 1.8.7, 1.9.2 and 1.9.3 have reached EOL. While you can still run them and receive technical support, security patches are not be applied. Ruby 2.0.0 will be EOL on Feb. 24th, 2016.

JRuby:

- `1.7.25` , Ruby Versions: [`1.9.3`]
- `9.0.5.0` , Ruby Versions: [`2.2.3`]
- `9.1.5.0` , Ruby Versions: [`2.3.1`]

JRuby versions support multiple ruby versions listed below. You need to specify one in your Gemfile. JRuby runs on the JVM which is also installed alongside JRuby. For a list of supported Java versions and details on how to configure specific versions, see the Java support article (https://devcenter.heroku.com/articles/java-support#supported-java-versions).

For more information on the JVM environment and available JVM options (such as `JAVA_TOOL_OPTIONS`) see Heroku's Java Support on Dev Center (https://devcenter.heroku.com/articles/java-support#environment).

Available runtimes

Additional, unsupported runtimes, are also available for JRuby:

MRI:

JRuby:

- `1.7.13` , Ruby Versions: [`1.8.7` , `1.9.3` , `2.0.0` (experimental)]
- `1.7.14` , Ruby Versions: [`1.8.7` , `1.9.3` , `2.0.0` (experimental)]
- `1.7.15` , Ruby Versions: [`1.8.7` , `1.9.3` , `2.0.0` (experimental)]
- `1.7.16` , Ruby Versions: [`1.8.7` , `1.9.3` , `2.0.0` (experimental)]
- `1.7.16.1` , Ruby Versions: [`1.8.7` , `2.0.0` (experimental)]
- `1.7.17` , Ruby Versions: [`1.8.7` , `1.9.3`]
- `1.7.18` , Ruby Versions: [`1.8.7` , `1.9.3` , `2.0.0` (experimental)]
- `1.7.19` , Ruby Versions: [`1.8.7` , `1.9.3` , `2.0.0` (experimental)]
- `1.7.20` , Ruby Versions: [`1.8.7` , `1.9.3` , `2.0.0` (experimental)]
- `1.7.21` , Ruby Versions: [`1.8.7` , `1.9.3` , `2.0.0` (experimental)]
- `1.7.22` , Ruby Versions: [`1.8.7` , `1.9.3` , `2.0.0` (experimental)]
- `1.7.23` , Ruby Versions: [`1.8.7` , `1.9.3` , `2.0.0` (experimental)]
- `1.7.24` , Ruby Versions: [`1.8.7` , `1.9.3` , `2.0.0` (experimental)]
- `1.7.25` , Ruby Versions: [`1.8.7` , `2.0.0` (experimental)]
- `9.0.0.0` , Ruby Versions: [`2.2.2`]
- `9.0.1.0` , Ruby Versions: [`2.2.2`]
- `9.0.2.0` , Ruby Versions: [`2.2.2`]
- `9.0.3.0` , Ruby Versions: [`2.2.2`]
- `9.0.4.0` , Ruby Versions: [`2.2.2`]
- `9.0.5.0` , Ruby Versions: [`2.2.3`]
- `9.1.0.0` , Ruby Versions: [`2.3.0`]
- `9.1.1.0` , Ruby Versions: [`2.3.0`]
- `9.1.2.0` , Ruby Versions: [`2.3.0`]
- `9.1.3.0` , Ruby Versions: [`2.3.1`]
- `9.1.4.0` , Ruby Versions: [`2.3.1`]

Selecting a runtime

Please see our Ruby Versions (https://devcenter.heroku.com/articles/ruby-versions) document for instructions on how to specify your Ruby version.

The Ruby runtime that your app uses will be included in your slug (https://devcenter.heroku.com/articles/slug-compiler), which will affect the slug size (https://devcenter.heroku.com/articles/limits#build).

Ruby applications

Pure Ruby applications, such as headless processes and evented web frameworks like Goliath, are fully supported on Cedar.

Activation

When a deployed application is recognized as a pure Ruby application, Heroku responds with `-----> Ruby app detected` .

```
$ git push heroku master
-----> Ruby app detected
```

Add-ons

A dev database add-on is provisioned if the Ruby application has the `pg` gem in the `Gemfile` . This populates the `DATABASE_URL` environment var. For more information see Ruby Database Provisioning (https://devcenter.heroku.com/articles/ruby-database-provisioning).

Process types

No default `web` process type is created if a pure Ruby application is detected.

Rack applications

Rack applications behave like Ruby applications, with the following additions.

Activation

A root-level `config.ru` file specifies the existence of a Rack application. Applications recognized as Rack apps are denoted with a `-----> Ruby/Rack app detected` at deploy-time.

```
$ git push heroku master
-----> Ruby/Rack app detected
```

Environment

The following additional environment variable will be set:

- `RACK_ENV` => “production”

Add-ons

A dev database add-on is provisioned if the Rack application has the `pg` gem in the `Gemfile`. This populates the `DATABASE_URL` environment variable. For more information see Ruby Database Provisioning (https://devcenter.heroku.com/articles/ruby-database-provisioning).

Process types

If you don't include a `Procfile` (https://devcenter.heroku.com/articles/procfile), Rack apps will define a web process type at deploy time:

```
web: bundle exec rackup config.ru -p $PORT
```


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`.

As a special case to assist in migration from Bamboo, Ruby apps which bundle the `thin` gem will get this web process type:

```
web: bundle exec thin start -R config.ru -e $RACK_ENV -p $PORT
```

Rails 2.x applications

Activation

An app is detected as a Rails 2.x app when the `Gemfile.lock` file contains a `rails` gem, and the gem version is greater than or equal to 2.0.0, and less than 3.0.0. Apps recognized as Rails 2.x apps are denoted with a `-----> Ruby/Rails app detected` at deploy-time.

```
$ git push heroku master
-----> Ruby/Rails app detected
```

Environment

The following additional environment variables will be set:

- `RAILS_ENV` => "production"
- `RACK_ENV` => "production"

Add-ons

A dev database add-on will be provisioned. This populates the `DATABASE_URL` environment variable. For more information see Ruby Database Provisioning (https://devcenter.heroku.com/articles/ruby-database-provisioning).

Process types

If you don't include a `Procfile` (https://devcenter.heroku.com/articles/procfile), Rails 2 apps will define a web process type at deploy time:

```
web: bundle exec ruby script/server -p $PORT
```


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`.

As a special case to assist in migration from Bamboo, Ruby apps which bundle the `thin` gem will get this web process type:

```
web: bundle exec thin start -e $RAILS_ENV -p $PORT
```

Two additional process types are declared for Rails 2:

```
worker: bundle exec rake jobs:work
console: bundle exec script/console
```

Plugin injection in Rails 2

- A Rails stdout plugin is injected.

Rails 3.x applications

Activation

An app is detected as a Rails 3.x app when the `Gemfile.lock` file contains a `railties` gem, and the gem version is greater than or equal to 3.0.0, and less than 4.0.0. Apps recognized as Rails 3.x apps are denoted with a `-----> Ruby/Rails app detected` at deploy-time.

```
$ git push heroku master
-----> Ruby/Rails app detected
```

Environment

The following additional environment variables will be set:

- `RAILS_ENV` => “production”
- `RACK_ENV` => “production”

Add-ons

A dev database add-on will be provisioned. This populates the `DATABASE_URL` environment variable. For more information see Ruby Database Provisioning (https://devcenter.heroku.com/articles/ruby-database-provisioning).

Process types

If you don’t include a `Procfile` (https://devcenter.heroku.com/articles/procfile), Rails 3 apps will define a web and console process type at deploy time:

```
web: bundle exec rails server -p $PORT
console: bundle exec rails console
```

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` .

As a special case to assist in migration from Bamboo, Ruby apps which bundle the `thin` gem will get this web process type:

```
web: bundle exec thin start -R config.ru -e $RACK_ENV -p $PORT
```

Compile phase

As a final task in the compilation phase, the `assets:precompile` Rake task is executed. This will compile all assets and put them in your public directory. For more information refer to Rails Asset Pipeline on Heroku Cedar (https://devcenter.heroku.com/articles/rails-asset-pipeline).

Plugin injection in Rails 3

- A Rails stdout plugin is injected.
- A static assets plugin is automatically injected.

If you include the `rails_12factor` (https://github.com/heroku/rails_12factor) gem, then no plugin injection is performed at all - all the configuration will be made via the gem. You should put this in your `:production` group to avoid a bug in development that will cause you to see duplicate logs. This has been fixed in later versions of Rails (https://github.com/rails/rails/pull/11060).

Rails 4.x applications

Activation

An app is detected as a Rails 4.x app when the `Gemfile.lock` file contains a `railties` gem, and the gem version is greater than or equal to 4.0.0.beta, and less than 5.0.0. Apps recognized as Rails 4.x apps are denoted with a `-----> Ruby/Rails app detected` at deploy-time.

```
$ git push heroku master
-----> Ruby/Rails app detected
```

Environment

The following additional environment variables will be set:

- `RAILS_ENV` => “production”
- `RACK_ENV` => “production”

Add-ons

A dev database add-on will be provisioned. This populates the `DATABASE_URL` environment variable. For more information see Ruby Database Provisioning (https://devcenter.heroku.com/articles/ruby-database-provisioning).

Process types

If you don’t include a `Procfile` (https://devcenter.heroku.com/articles/procfile), Rails 4 apps will define a web and console process type at deploy time:

```
web: bundle exec bin/rails server -p $PORT -e $RAILS_ENV
console: bundle exec bin/rails console
```

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` .

Compile phase

As a final task in the compilation phase, the `assets:precompile` Rake task is executed if you have a `assets:precompile` Rake task defined, and don’t have a `public/assets/manifest-*.json` file. This will compile all assets and put them in your public directory. If the rake detection or the asset compilation task fails, the deploy will fail as well. For more information refer to Rails Asset Pipeline on Heroku Cedar (https://devcenter.heroku.com/articles/rails-asset-pipeline).

Plugin injection in Rails 4

Rails 4 no longer supports plugin functionality, and so Heroku's Rail 4 no longer injects any. However, if you don't include the `rails_12factor` (https://github.com/heroku/rails_12factor) gem, a warning will be raised. This gem replaces the need for the plugins, and ensures that Rails 4 is optimally configured for executing on Heroku.

Rails 5.x applications

Activation

An app is detected as a Rails 5.x app when the `Gemfile.lock` file contains a `railties` gem, and the gem version is greater than or equal to 5.0.0.beta, and less than 6.0.0. Apps recognized as Rails 5.x apps are denoted with a `-----> Ruby/Rails app detected` at deploy-time.

```
$ git push heroku master
-----> Ruby/Rails app detected
```

Environment

The following additional environment variables will be set:

- `RAILS_ENV` => "production"
- `RACK_ENV` => "production"
- `RAILS_LOG_TO_STDOUT` => "enabled"
- `RAILS_SERVE_STATIC_FILES` => "enabled"

Add-ons

A dev database add-on will be provisioned. This populates the `DATABASE_URL` environment variable. For more information see Ruby Database Provisioning (https://devcenter.heroku.com/articles/ruby-database-provisioning).

Process types

If you don't include a `Procfile` (https://devcenter.heroku.com/articles/procfile), Rails 5 apps will define a web and console process type at deploy time:

```
web: bundle exec bin/rails server -p $PORT -e $RAILS_ENV
console: bundle exec bin/rails console
```

> Regardless of the webserver you choose, production apps should always specify the webserver explicitly in the `Procfile`.

Compile phase

As a final task in the compilation phase, the `assets:precompile` Rake task is executed if you have a `assets:precompile` Rake task defined, and don't have a `public/assets/manifest-*.json` file. This will compile all assets and put them in your public directory. If the rake detection or the asset compilation task fails, the deploy will fail as well. For more information refer to Rails Asset Pipeline on Heroku Cedar (https://devcenter.heroku.com/articles/rails-asset-pipeline).

Plugin injection in Rails 5

Rails 5 no longer supports plugin functionality, and so Heroku's Rail 5 no longer injects any. Previously in Rails 4 the gem `rails_12factor` (https://github.com/heroku/rails_12factor) was required to enable logging and static file service. New applications will now work out of the box. If you are upgrading an application see Rails 5 getting started guide (https://devcenter.heroku.com/articles/getting-started-with-rails5#heroku-gems).

Installed binaries

A common dependency of Ruby applications that compile assets is Node.js. When we detect the `execjs` gem, Node.js version `0.10.30` will be installed in your application's `PATH`. If you need a specific version of node for your Ruby application you should use multiple buildpacks to first install node then install Ruby (https://devcenter.heroku.com/articles/using-multiple-buildpacks-for-an-app).

```
$ heroku buildpacks:add heroku/nodejs
$ heroku buildpacks:add heroku/ruby
```

Verify that your buildpacks are set correctly and that Node comes before Ruby:

```
$ heroku buildpacks
=== myapp Buildpack URLs
1. heroku/nodejs
2. heroku/ruby
```

Once you have done this you'll need a `package.json` file in the root of your app. For example to install version `6.2.2` your `package.json` could look like this:

```
{ "engines" : { "node": "6.2.2" } }
```

Commit to git:

```
$ git add pacakage.json
$ git commit -m 'specify node version'
```

Now the next time you deploy your Ruby application will use Node version 6.2.2.

Injected plugins

By default, Heroku will inject plugins in Rails 3.x applications to ensure applications get the most out of the Heroku platform. The two plugins that may be injected are documented below. To avoid this injection in Rails 3, include the `rails_12factor` (https://github.com/heroku/rails_12factor) gem in your application. In your Gemfile:

```
gem 'rails_12factor'
```

Stdout

The `rails_stdout_logging` (https://github.com/heroku/rails_stdout_logging) gem ensures that your logs will be sent to standard out.

Heroku treats logs as streams of events, rather than files (https://devcenter.heroku.com/articles/management-visibility#logging) - by piping logs to stdout, your logs will be captured and consolidated across multiple dynos by Logplex (https://devcenter.heroku.com/articles/logging), which in turn makes them available from the command line, `$ heroku logs --tail` , or from logging add-ons (https://elements.heroku.com/addons/#logging).

Static assets

The `rails3_serve_static_assets` (https://github.com/pedro/rails3_serve_static_assets) gem lets the web process serve static assets.

In the default Rails development environment assets are served through a middleware called sprockets (https://github.com/sstephenson/sprockets). In production however most non-heroku Rails deployments will put their ruby server behind reverse HTTP proxy server such as Nginx which can load balance their sites and can serve static files directly. When Nginx sees a request for an asset such as `/assets/rails.png` it will grab it from disk at `/public/assets/rails.png` and serve it. The Rails server will never even see the request.

On Heroku, Nginx is not needed to run your application. Our routing layer (https://devcenter.heroku.com/articles/http-routing) handles load balancing while you scale out horizontally. The caching behavior of Nginx is not needed if your application is serving static assets through an edge caching CDN (https://en.wikipedia.org/wiki/Content_delivery_network).

By default Rails 4 will return a 404 if an asset is not handled via an external proxy such as Nginx. While this default behavior will help you debug your Nginx configuration, it makes a default Rails app with assets unusable on Heroku. To fix this we've released a gem `rails_serve_static_assets` .

This gem, `rails_serve_static_assets` , enables your Rails server to deliver your assets instead of returning a 404. You can use this to populate an edge cache CDN, or serve files directly from your web app. This gives your app total control and allows you to do things like redirects, or setting headers in your Ruby code. To enable this behavior in your app we only need to set this one configuration option:

```
config.serve_static_assets = true
```

You don’t need to set this option since this that is what this gem does. Now your application can take control of how your assets are served.

Debugger gems fail to install

There are several gems that require very specific patch levels of the Ruby they are running on. This is detrimental to a production app as it locks you into a specific patch level of Ruby and does not allow you to upgrade to receive security fixes.

Heroku releases security patches for Ruby versions as the become available from Ruby core. After we have upgraded a Ruby version, your app will get the new version on next deploy. If your app was using one of these gems you will see a failure when installing gems if we have upgraded the version of Ruby you are using. A failure may look like this:

```
Installing debugger-linecache
Gem::Installer::ExtensionBuildError: ERROR: Failed to build gem native extension.
```

Or like this:

```
Installing debugger
Gem::Installer::ExtensionBuildError: ERROR: Failed to build gem native extension.
```

The best way to avoid this problem is to not use `debugger` or any other debugging gems in production. These gems are designed to hook into the low level components of a language for dynamically stopping and inspecting execution of running code. This is not something you should be doing in production. Instead move these gems to your `development` group of the `Gemfile`:

```
group :development do
  gem "debugger"
end
```

Then `bundle install`, commit to git, and re-deploy.