Rails 3.1+ Asset Pipeline on Heroku Cedar

Last Updated: 08 June 2012

cedar rails ruby

Table of Contents

Getting Started

While Rails 3.1 and 3.2 can run on the Bamboo stack without the asset pipeline, these versions of Rails runs best on Heroku's Cedar stack. For new users, we recommend reading our tutorial for creating a Rails 3.x app on Cedar before proceeding further.

The Asset Pipeline

The new Rails asset pipeline is supported on Heroku's Cedar stack. The new pipeline makes assets a first class citizen in the Rails stack. By default, Rails uses CoffeeScript for JavaScript and SCSS for CSS. DHH has a great introduction during his keynote for RailsConf.

The Rails asset pipeline provides an `assets:precompile` rake task to allow assets to be compiled and cached up front rather than compiled every time the app boots.

There are three ways you can use the asset pipeline on Heroku.

1. Compiling assets locally.
2. Compiling assets during slug compilation.
3. Compiling assets during runtime.

Compiling assets locally

If a `public/assets/manifest.yml` is detected in your app, Heroku will assume you are handling asset compilation yourself and will not attempt to compile your assets.

To compile your assets locally, run the `assets:precompile` task locally on your app. Make sure to use the `production` environment so that the production version of your assets are generated.

```
RAILS_ENV=production bundle exec rake assets:precompile
```

A `public/assets` directory will be created. Inside this directory you'll find a `manifest.yml` which includes the md5sums of the compiled assets. Adding `public/assets` to your git repository will make it available to Heroku.

```
git add public/assets
git commit -m "vendor compiled assets"
```

Now when pushing, the output should show that your locally compiled assets were detected:

```
-----> Preparing Rails asset pipeline
       Detected manifest.yml, assuming assets were compiled locally
```

Compiling assets during slug compilation

The app's config vars are not available in the environment during the slug compilation process. Because the app must be loaded to run the `assets:precompile` task, any initialization code that requires existence of config vars should gracefully handle the `nil` case.

If you have not compiled assets locally, we will attempt to run the `assets:precompile` task during slug compilation. Your push output will show:

```
-----> Preparing Rails asset pipeline
       Running: rake assets:precompile
```

Please see the Troubleshooting section below on explanations of how the rake task works during our slug compilation process.

Compiling assets during runtime

If the `assets:precompile` task fails, the output will be displayed and runtime compilation of assets will be enabled:

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.

```
-----> Preparing Rails asset pipeline
       Running: rake assets:precompile
       ERROR: Unable to connect to memcached
       Precompiling assets failed, enabling runtime asset compilation
       Injecting rails31_enable_runtime_asset_compilation
```

By default, Rails prevents assets from being compiled during runtime so we inject this plugin to enable runtime asset compilation.

Asset caching

Caching of static assets can be implemented in-application using the Rack::Cache middleware or in a more distributed fashion with a CDN. Serving assets from your application does require dyno-resources so please consider an appropriate asset caching strategy for your needs.

Troubleshooting

assets:precompile failures

There's no fix or workaround at this time if assets:precompile is failing during slug compilation. Below we describe common issues you might run into and the reasons why it isn't working.

The most common cause of failures in `assets:precompile` is an app that relies on having its environment present to boot. Your app's config vars are not present in the environment during slug compilation, so you should take steps to handle the `nil` case for config vars (and add-on resources) in your initializers.

If you see something similar to the following:

```
could not connect to server: Connection refused
Is the server running on host "127.0.0.1" and accepting
TCP/IP connections on port xxxx?
```

This means that your app is attempting to connect to the database as part of `rake assets:precompile`. Because the config vars are not present in the environment, we use a placeholder `DATABASE_URL` to satisfy Rails. The full command run during slug compilation is:

```
env RAILS_ENV=production DATABASE_URL=scheme://user:pass@127.0.0.1/dbname bundle exec rake assets:precompile 2>&1
```

- `scheme` will be replaced with an appropriate database adapter as detected from your `Gemfile`

While precompiling assets, in Rails 3.1.1 and up, you can prevent initializing your application and connecting to the database by ensuring that the following line is in your `config/application.rb`:

```
config.assets.initialize_on_precompile = false
```

If `rake assets:precompile` is still not working, you can debug this locally by configuring a nonexistent database in your local `config/database.yml` and attempting to run `rake assets:precompile`. Ideally you should be able to run this command without connecting to the database.

therubyracer

If you were previously using `therubyracer` or `therubyracer-heroku`, these gems are no longer required and strongly discouraged as these gems use a very large amount of memory.

Updating PATH

If you need to compile assets at runtime, you must add `bin` to your PATH to access the JavaScript runtime. Check your current configuration using `heroku config`:

```
$ heroku config
PATH => vendor/bundle/ruby/1.9.1/bin:/usr/local/bin:/usr/bin:/bin
```

If your PATH variable does not include `bin` on its own, update it by running:

```
$ heroku config:add PATH=bin:vendor/bundle/ruby/1.9.1/bin:/usr/local/bin:/usr/bin:/bin
Adding config vars:
  PATH => vendor/bundle/ru...usr/bin:/bin:bin
Restarting app... done, v7.
```

If this article is incorrect or outdated, or omits critical information, please let us know. For all other issues, please see our support channels.

Submit feedback

Your feedback has been sent to the Dev Center team. Thank you.

« Rack::Sendfile Scheduled Jobs and Custom Clock Processes in Ruby with Clockwork »