Servers / Pagination / Code Smells / Gems

Devpoint Labs - Jake Sorce / Dave Jungst

Ruby’s Rack Middleware

Rack was created:

- to mix-and-match web servers with web applications and other technologies.
- out of the need of different environments for development, testing and/or production.

In 2007 Christian Neukirchen released Rack, what he then referred to as "a modular Ruby webserver interface". Today it is adapted by numerous web servers and web application development frameworks such as Espresso, Mack, Ruby on Rails, Sinatra etc.

Rack Middleware Cont...

The Rack Middleware Gem, implementing the Rack specification, works by dividing incoming HTTP requests into different pipelined stages and handles them in pieces until it sends back a response coming from your web application. It has two distinct components: a Handler and an Adapter, used for communicating with web servers and applications.

Common Servers That Implement Rack

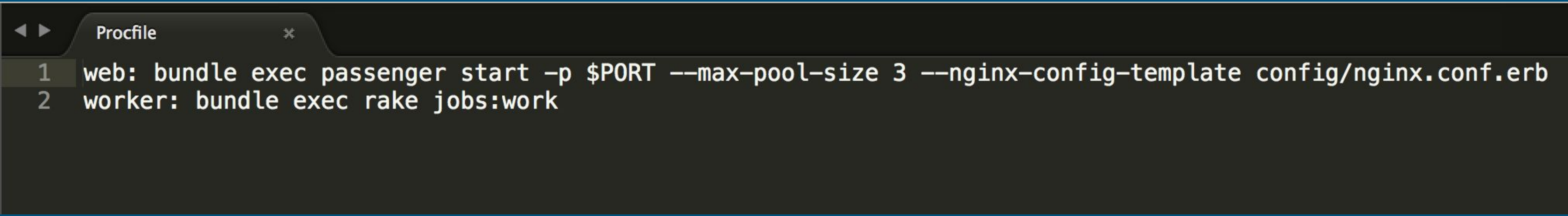
- WEBrick
  - Default Rails web server.
- Phusion Passenger: Fast web server & app server
  - A web server and application server for your web apps. Keeps your users happy, saves your business time and money.
- Puma: A Modern, Concurrent Web Server for Ruby
  - Built for speed and parallelism. (Heroku's recommended web server)
- Thin: Tiny, Fast, & Funny HTTP Server
  - A Ruby web server that glues together 3 of the best Ruby libraries in web history.
- Unicorn: Rack HTTP Server for Fast Clients and Unix
  - Fully-featured, however, it denies by design trying to do everything.

Heroku Procfiles

A Procfile is a mechanism for declaring what commands are run by your application’s dynos on the Heroku platform. It follows the process model. You can use a Procfile to declare various process types, such as multiple types of workers.

The Procfile lives in the root directory of your Rails application and has no extension

Procfile Example



```
web: bundle exec passenger start -p $PORT --max-pool-size 3 --nginx-config-template config/nginx.conf.erb
worker: bundle exec rake jobs:work
```

WEBrick - Rails Default Server

http://ruby-doc.org/stdlib-1.9.3/libdoc/webrick/rdoc/WEBrick.html

Phusion Passenger: Fast web server & app server

https://www.phusionpassenger.com/

- Mature
- Fast
- Most Recommended Web Server For Rails Deployments Not On Heroku
- Capable Of Handling Slow Clients
- Open Source
- Multi-Process Single-Threaded

Heroku Passenger Configuration

https://github.com/phusion/passenger-ruby-heroku-demo

Puma: A Modern, Concurrent Web Server for Ruby

http://puma.io/

- Rack exclusive Ruby web application server
- Small footprint both in size and execution resources consumption
- Simple, yet significant configuration options
- Heroku’s recommended web server

Heroku Puma Configuration

https://devcenter.heroku.com/articles/deploying-rails-applications-with-the-puma-web-server

Thin: Tiny, Fast, & Funny HTTP Server

http://code.macournoyer.com/thin/

- Thin HTTP server is designed to work with any framework implementing Rack
- Probably shouldn’t use for a production level application server unless you have a very small application and want a very small server footprint, otherwise I would go with Puma or Passenger depending on your hosting platform

Heroku Thin Configuration - NOT Recommended

In your Gemfile, create or change the production group to include thin:

```
group :production do
  gem 'thin'
end
```

Then, in your Procfile

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

Unicorn: Rack HTTP Server for Fast Clients and Unix

https://github.com/blog/517-unicorn

- Very mature
  - adapted to be used with Python as well
- Fully-featured
- Until recently Unicorn was Heroku's recommended web application server
  - a slow client attack vulnerability made heroku switch their recommendation to Puma

In a slow client attack, an attacker deliberately sends multiple partial HTTP requests to the server to carry out an HTTP DoS attack on the server. The client attempts to slow the request or response so much that it holds connections and memory resources open on the server for a long time, but without triggering session time-outs.

Heroku Unicorn Configuration

https://devcenter.heroku.com/articles/rails-unicorn

Servers Conclusion

Heroku Deployment:

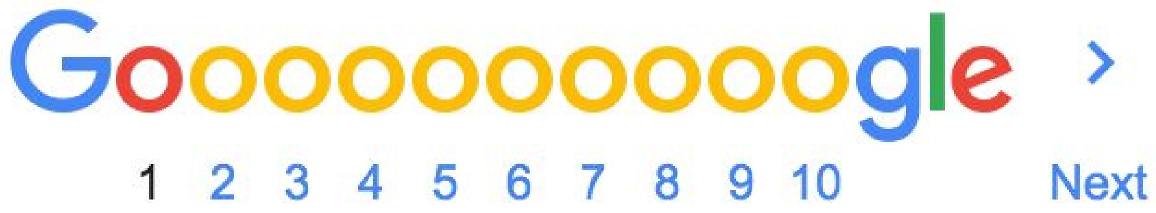
Even though Passenger’s recent updates address both speed and stability, Puma offers a lean and fast web server solution for most applications that require high concurrency. Of course, it always makes sense to run your own benchmarks and application tests.

Amazon / Custom Web Server Deployment:

Phusion Passenger would be the way to go

Pagination

Pagination is used for such things as displaying a limited number of results on search engine results pages, or showing a limited number of posts when viewing a forum thread. Pagination is used in some form in almost every web application to divide returned data and display it on multiple pages. Pagination also includes the logic of preparing and displaying the links to the various pages.



Pagination Gems

- Kaminari
  - Kaminari is a Scope Engine based, clean, powerful, agnostic, customizable and sophisticated paginator for Rails 3+
    - https://github.com/amatsuda/kaminari
- will_paginate
  - will_paginate provides a simple API for performing paginated queries with Active Record, DataMapper and Sequel, and includes helpers for rendering pagination links in Rails and Sinatra.
    - https://github.com/mislav/will_paginate/wiki
- ajax_pagination
  - Loads page content into AJAX sections with AJAX links, handling the details for you, load content with javascript into designated page containers.
    - https://github.com/ronalchn/ajax_pagination

will_paginate Example

1. build a new rails project with a postgres database
2. add a model
3. add the will_paginate gem
4. add the populator and faker gems
5. write a populate rake task
6. populate your database with a bunch of model records
7. implement will_paginate for performance
8. always think about and use pagination in production level applications!

Kaminari Example

1. build a new rails project with a postgres database
2. add a model
3. add the kaminari gem
4. add the populator and faker gems
5. write a populate rake task
6. populate your database with a bunch of model records
7. implement kaminari for performance
8. always think about and use pagination in production level applications!

Code Smells

A code smell is a surface indication that usually corresponds to a deeper problem in the system. The term was first coined by a guy named Kent Beck.

One of the nice things about smells is that it's easy for inexperienced people to spot them, even if they don't know enough to evaluate if there's a real problem or to correct them. I've heard of lead developers who will pick a "smell of the week" and ask people to look for the smell and bring it up with the senior members of the team. Doing it one smell at a time is a good way of gradually teaching people on the team to be better programmers.

Gems

- rubocop
  - Automatic Ruby code style checking tool. Aims to enforce the community-driven Ruby Style Guide.
  - http://github.com/bbatsov/rubocop
- Rails best practices
  - a code metric tool for rails codes, written in Ruby.
    - http://rails-bestpractices.com/
- Reek
  - Reek is a tool that examines Ruby classes, modules and methods and reports any code smells it finds.
    - https://github.com/troessner/reek

Gems Cont...

- Flog
  - Flog reports the most tortured code in an easy to read pain report. The higher the score, the more pain the code is in.
    - http://ruby.sadi.st/Ruby_Sadist.html
- Bullet
  - help to kill N+1 queries and unused eager loading
    - https://github.com/flyerhzm/bullet

Capstone Mini Project

1. Make sure you are running a production level web server like: puma, unicorn, or passenger, NEVER use WEBrick for production!

2. Add a few of the gems mentioned above for better insight and more peace of mind.

3. Make sure you are paginating index pages and places you are showing or could potentially show lots of data!