A Comparison of (Rack) Web Servers for Ruby Web Applications

Authored by: ***ASPHostServer Administrator*** *[asphostserver@gmail.com]*
Saved From: http://faq.asphosthelpdesk.com/article.php?id=232

Meet Ruby's Rack

Nowadays, giving the opportunity to mix-and-match web servers with web applications and other technologies [to developers] is a very powerful, necessary and indeed in-demand aspect of any programming language and framework. A very basic example for this is the need of different environments for development, testing and/or production.

With the goal of making this happen, 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.

Despite the naming conventions and assorted explanations across [programming] languages and domains, specifications for interfacing [applications] are very similar, if not homologous (although unlike Python's WSGI -- an inspiration to Rack and others, Rack here is not only a specification but -- perhaps a little confusingly -- a middleware application as well).

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 (controller). It has two distinct components: a Handler and an Adapter, used for communicating with web servers and applications (frameworks) respectively.***

About The Comparison

In this comparison, we will talk about some of the popular and available web application server choices today. We will go through what makes one stand out against the other and how they differ in certain aspects from the rest. The aim here is to enable you to have a better overall understanding in order to be able to match servers with your application's needs to craft the solution you require.

Web Servers (in Alphabetical Order)

Phusion Passenger: Fast web server & app server

Passenger today has become the recommended server for Ruby on Rails applications. It is a mature, feature rich product which aims to cover necessary needs and areas of application deployment whilst greatly simplifying the set-up and getting-started procedures. It eliminates the traditional middleman server set up architecture by direct integration within **Apache** and **NGINX** (front-facing) web servers. It is also referred to as mod_rails / mod_rack for this very reason amongst the Ruby (Rails) community.

Notes

Passenger provides the ability to work with **multiple** applications hosted on the *same* server.

It is capable of handling slow clients. Requests and responses with Passenger are fully buffered, rendering itself immune to such attacks targeting to clog systems' resources.

Passenger is highly popular and used widely in many [production] scenarios. It is therefore possible to reach out and find experts and also have your issues addressed in online communities. It also has a dedicated company developing the product, which offers efficient commercial support for those interested.

Its open-source version has a multi-process single-threaded operation mode, whilst the Enterprise version can be configured to work either single-threaded or multi-threaded.

For enterprise users, Passenger provides some further "advanced" features such as: concurrency and multithreading, deployment en masse, resource control and limiting, et al.

To learn more about Passenger, you can visit its official website located at https://www.phusionpassenger.com/. For Passenger Enterprise, please visit enterprise section on the same page.

Puma: A Modern, Concurrent Web Server for Ruby

Puma is a Rack exclusive Ruby web application server. Its creation and birth is heavily inspired by the legacy Mongrel web server, which had revolutionized a lot during the time it was released and affected how Ruby applications were served for many generations (of servers) to come. Puma's developer, Evan Phoenix, deciding to move things forward, transformed Mongrel's operational structure to solely operate on Rack (and thus eliminating complexities causing performance decay) and designed the application to support *true parallelism*, which allows concurrency (explained further below).

Notes

The application has quite a small foot print, both in size and execution resources consumption.

Puma has several working modes: it allows you to set the amount of minimum and maximum threads it can use to do its job and also works in a clustered mode whereby you can use forked processes to handle requests concurrently.

It is based on Mongrel's parser -- and a lot of rewrites of its codebase.

Despite being designed as "the go-to server" for Rubinius, it also works well with JRuby.

It comes with a simple, yet significant configuration option set to adapt the web server both for production and development needs in many ways.

Although Puma does not directly support hosting multiple applications out-of-the-box, it has Jungle: a (Puma as a service) tool to help with the production needs of multiple applications.

To learn more about Puma, you can visit its official code repository located at https://github.com/puma/puma where you can also find instructions to configure and get started.

Thin: Tiny, Fast, & Funny HTTP Server

Thin is a very popular application server which claims itself to be the most "secure, stable, fast and extensible Ruby web server". Still actively developed, it is based on, and hence the product of, three decisive

Ruby libraries:

- Mongrel's parser
- Event Machine network I/O library
- Rack middleware

Notes

Thin HTTP server is designed to work with any framework implementing Rack specification, which covers the majority today. It does this by loading Rack configuration files directly.

Being an Event / Machine based application server, Thin is capable of handling long running requests unlike some other choices without the help of a front facing reverse-proxy solution.

To learn more about Thin, you can visit its official website located at http://code.macournoyer.com/thin/.

Unicorn: Rack HTTP Server for Fast Clients and Unix

Unicorn is a very mature [Ruby] web application server, so much so that it was adapted to be used with Python as well. It is fully-featured, however, it denies by design trying to do everything: Unicorn's principal is doing what needs to be done [by a web application server] and delegating rest of the responsibilities to those which do them better.

Notes

Unicorn's master process spawns workers, as per your requirements, to serve the requests. This process also monitors the workers in order to prevent memory and process related staggering issues. What this means for system administrators is that it will kill a process if (for example) it takes too much time to complete a task or memory issues occur.

As mentioned above, one of the areas in which Unicorn delegates tasks is using the operating system for load balancing. This allows the requests not to pile up against busy workers spawned.

Much like NGINX, with Unicorn you can perform and deploy your applications (think of updates/upgrades) without dropping alive connections and clients.

Some of its other advanced features:

- All workers run within a given isolated address space, serving one request at a time.
- **before_fork and after_fork hooks for dealing with forked processes.**
- Ability to be used with copy-on-write friendly memory management to save memory.
- Ability to listen to multiple interfaces.

To learn more about Unicorn and its great features, you can visit its official website located at http://unicorn.bogomips.org/.