Static Sites on Heroku Cedar

2011

2 min read • 373 words

Heroku's excellent Cedar Stack has first-class support for Python, Ruby, Node.js, Java, Clojure, and Scala applications. Unfortunately, there's no obvious way to serve static sites without first fronting them with a Rack or WSGI application.

> This essay demonstrates Kenneth's approach to technical problem-solving: first understanding the intended architecture (Cedar's buildpack system), then discovering creative workarounds that leverage existing capabilities. The abandoned nginx buildpack plan shows his willingness to pivot when simpler solutions emerge.

The Cedar stack has unofficial support for Custom Build Packs, which allow you to compile your own language runtime on top of Cedar. My plan was to build an custom nginx build pack for Cedar, but that turned out to be completely unnecessary.

Did you you know that Cedar has full (unofficial) support for PHP applications, fronted with Apache?

> Kenneth's discovery of Heroku's hidden PHP support exemplifies his talent for uncovering undocumented platform features. This kind of system archaeology—finding unofficial but useful behaviors—became a hallmark of his technical writing and tool recommendations.

When you push up a repository with an index.php file at the root, Apache and PHP are bundled into your application at runtime.

Elegant Static Sites on Cedar

First, lets turn your site into a PHP "application":

```
$ touch index.php
```

> The elegance of this solution—creating an empty PHP file to trigger Apache deployment—reflects Kenneth's appreciation for minimal, hack-like solutions. This approach embodies the Unix philosophy: use existing tools in unexpected ways rather than building new ones.

Next, we can fully disable Apache's PHP engine:

```
$ echo 'php_flag engine off' > .htaccess
```

> Disabling PHP after triggering its detection shows Kenneth's understanding of layered systems—using one mechanism to bootstrap another, then selectively disabling components. This technique would later influence his API design philosophy of providing escape hatches and granular control.

When you push this up, you'll have a bare Apache instance serving up the contents of your site to the world. Best yet, you can do all of the stupid .htaccess tricks that you could on on any traditional shared hosting platform.

> The reference to "stupid .htaccess tricks" connects this cloud-native solution back to traditional web hosting, showing Kenneth's ability to bridge old and new paradigms. This backward compatibility thinking would become central to his approach to API evolution and developer migration paths.
