Automating Web Performance Measurement

By Addy Osmani
Eng Manager, Web Developer Relations

Web performance can have a huge impact on your entire user experience. If you've been looking at improving your own site's perf lately, you've probably heard of PageSpeed Insights - a tool that analyzes pages and offers advice on how to make them faster based on best practices for mobile and desktop web performance.

PageSpeed's scores are based on a number of factors, including how well your scripts are minimized, images optimized, content gzipped, tap targets being appropriately sized and landing page redirects avoided.

With 40% of people potentially abandoning pages that take more than 3 seconds to load, caring about how quickly your pages load on your users devices is increasingly becoming an essential part of our development workflow.

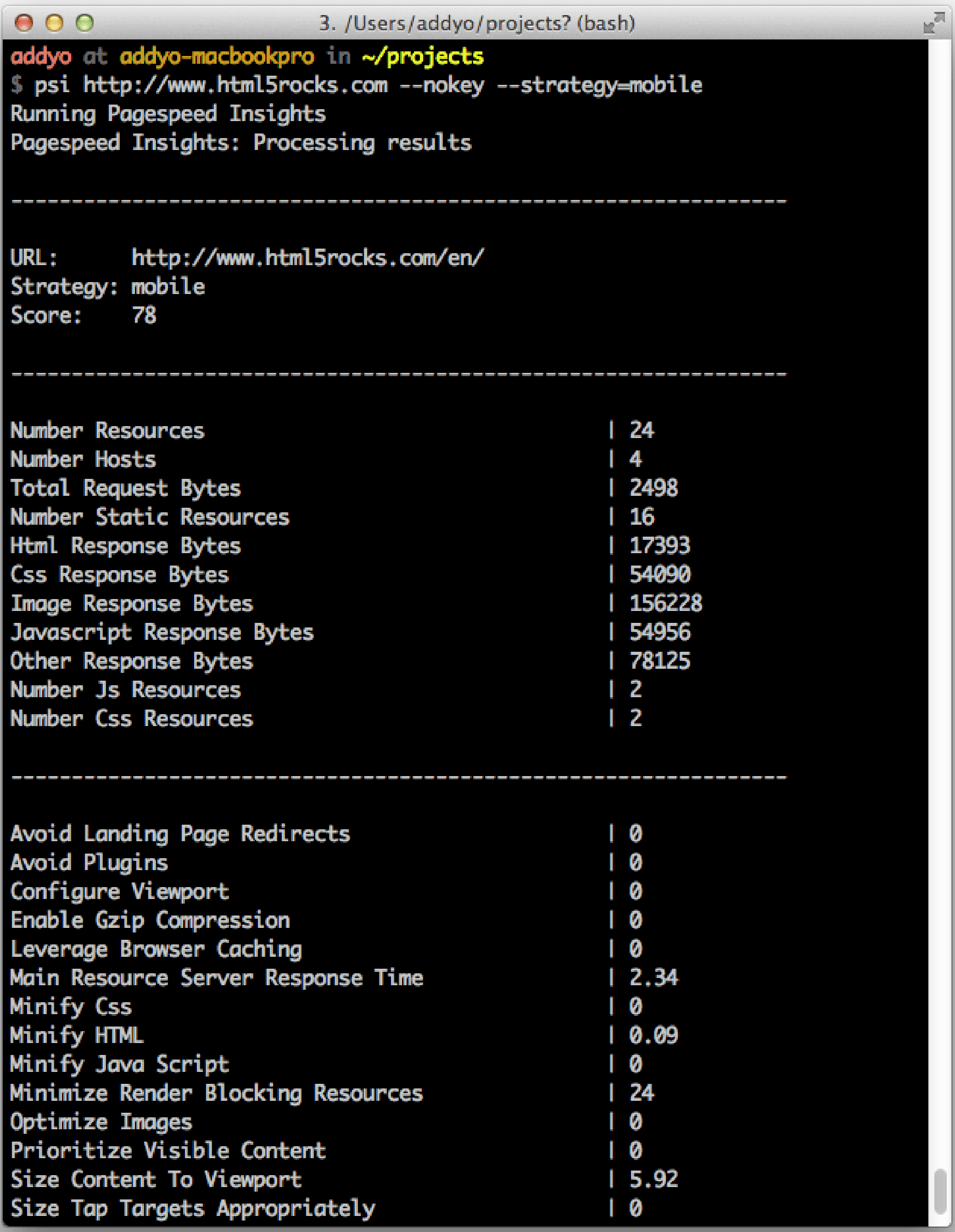
Performance metrics in your build process

Although manually going to the PageSpeed Insights to find out how your scores is fine, a number of developers have been asking whether it's possible to get similar performance scoring into their build process.

The answer is: absolutely.

Introducing PSI for Node

Today we're happy to introduce PSI for Node - a new module that works great with Gulp, Grunt and other build systems and can connect up to the PageSpeed Insights service and return a detailed report of your web performance. Let's look at a preview of the type of reporting it enables:



```
addyo at addyo-macbookpro in ~/projects
$ psi http://www.html5rocks.com --nokey --strategy=mobile
Running Pagespeed Insights
Pagespeed Insights: Processing results

----------------------------------------------------------------

URL:      http://www.html5rocks.com/en/
Strategy: mobile
Score:    78

----------------------------------------------------------------

Number Resources                                 | 24
Number Hosts                                     | 4
Total Request Bytes                              | 2498
Number Static Resources                          | 16
Html Response Bytes                              | 17393
Css Response Bytes                               | 54090
Image Response Bytes                             | 156228
Javascript Response Bytes                        | 54956
Other Response Bytes                             | 78125
Number Js Resources                              | 2
Number Css Resources                             | 2

----------------------------------------------------------------

Avoid Landing Page Redirects                     | 0
Avoid Plugins                                    | 0
Configure Viewport                               | 0
Enable Gzip Compression                          | 0
Leverage Browser Caching                         | 0
Main Resource Server Response Time               | 2.34
Minify Css                                       | 0
Minify HTML                                      | 0.09
Minify Java Script                               | 0
Minimize Render Blocking Resources               | 24
Optimize Images                                  | 0
Prioritize Visible Content                       | 0
Size Content To Viewport                         | 5.92
Size Tap Targets Appropriately                   | 0
```

The results above are good for getting a feel for the type of improvements that could be made. For example, a 5.92 for sizing content to viewport means “some” work can still be done whilst a 24 for minimizing render blocking resources may suggest you need to defer loading of JS using the async attribute.

Lowering the barrier of entry to PageSpeed Insights

If you've tried using the PageSpeed Insights API in the past or attempted to use any of the tools we build on top of it, you probably had to register for a dedicated API key. We know that although this just takes a few minutes, it can be a turn off for getting Insights as part of your regular workflow.

We're happy to inform you that the PageSpeed Insights service supports making requests without an API key for up to 1 request every 5 seconds (plenty for anyone). For more regular usage or serious production builds, you'll probably want to register for a key.

The PSI module supports both a `nokey` option for getting it setup in less than a few minutes and the `key` option for a little longer. Details on how to register for an API key are documented.

Getting started

You have two options for how you integrate PSI into your workflow. You can either integrate it into your build process or run it as a globally installed tool on your system.

Build process

Using PSI in your Grunt or Gulp build-process is fairly straight-forward. If you're working on a Gulp project, you can install and use PSI directly.

Install

Install PSI using npm and pass `--save-dev` to save it to your package.json file.

```
npm install psi --save-dev
```

Then define a Gulp task for it in your gulpfile as follows (also covered in our Gulp sample project):

```
var psi = require('psi');
gulp.task('psi', function (cb) {
  psi({
      nokey: 'true', // or use key: ‘YOUR_API_KEY’
      url: site,
      strategy: 'mobile',
  }, cb);
});
```

For the above, you can then run the task using:

```
gulp psi
```

And if you’re using Grunt, you can use grunt-pagespeed by James Cryer which now uses PSI to power it’s reporting.

Install

```
npm install grunt-pagespeed --save-dev
```

Then load the task in your Gruntfile:

```
grunt.loadNpmTasks('grunt-pagespeed');
```

and configure it for use:

```
pagespeed: {
  options: {
    nokey: true,
    url: "https://www.html5rocks.com",
    strategy: "mobile"
  }
}
```

You can then run the task using:

```
grunt pagespeed
```

Installing as a global tool

You can also install PSI as a globally available tool on your system. Once again, we can use npm to install the tool:

```
$ npm install -g psi
```

And via any terminal window, request PageSpeed Insights reports for a site (with the **nokey** option or an API specific **key** as follows):

```
psi http://www.html5rocks.com --nokey --strategy=mobile
```

or for those with a registered API key:

```
psi http://www.html5rocks.com --key=YOUR_API_KEY --strategy=mobile
```

That’s it!

Go forth and make performance part of your culture

We need to start thinking more about the impact of our designs and implementations on user experience.

Solutions like PSI can keep an eye on your web performance on desktop and mobile and are useful when used as part of your regular post-deployment workflow.

We're eager to hear of any feedback you might have and hope PSI helps improve the performance culture on your team.