PERFORMANCE TESTING CRASH COURSE

Dustin Whittle

- dustinwhittle.com
- @dustinwhittle
- San Francisco, California, USA
- Technologist, Traveler, Pilot, Skier, Diver, Sailor, Golfer

What I have worked on

- Developer Evangelist @ APPDYNAMICS
- Consultant & Trainer @ SensioLabs
- Developer Evangelist @ YAHOO!

Why does performance matter?

Microsoft found that **Bing** searches that were 2 seconds slower resulted in a **4.3% drop** in revenue per user

When **Mozilla** shaved **2.2** seconds off their landing page, Firefox downloads **increased 15.4%**

Making **Barack Obama's** website **60% faster** increased donation conversions by **14%**

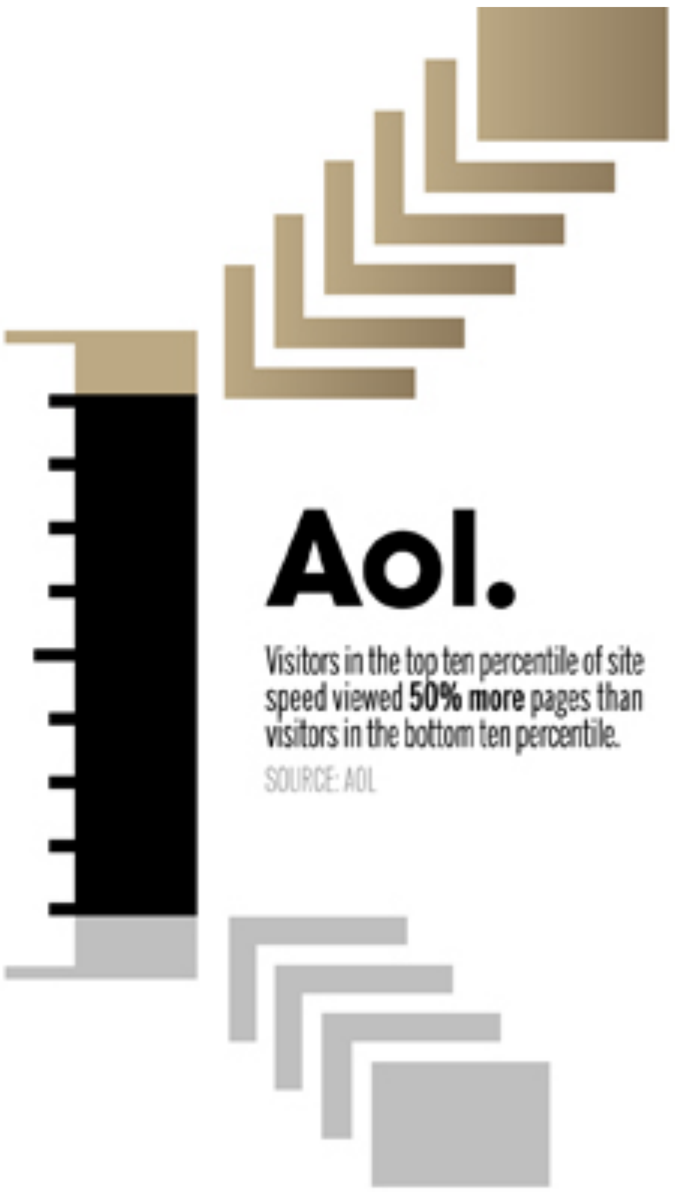
Amazon and **Walmart** increase revenue 1% for every 100ms of improvement

100 MILLISECONDS

amazon.com

Increased revenue by 1% for every 100 milliseconds of improvement.

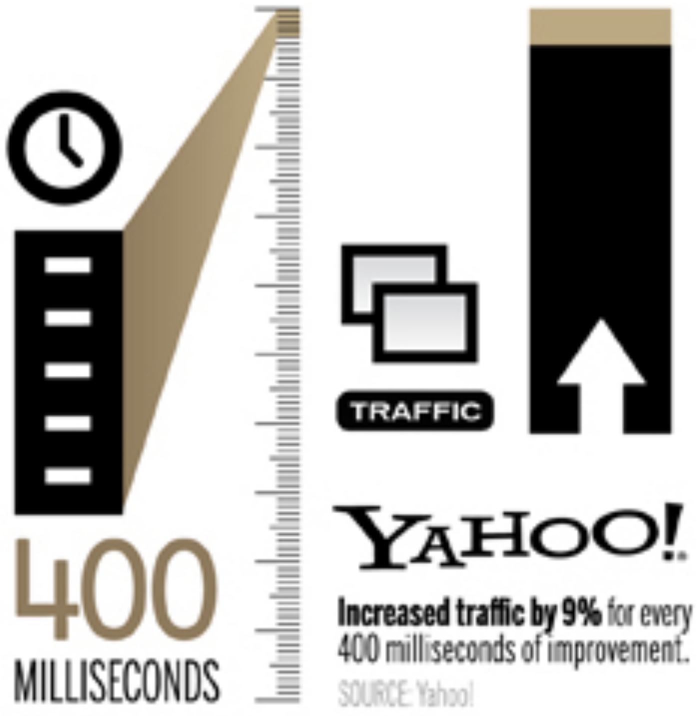
SOURCE: Amazon



Aol.

Visitors in the top ten percentile of site speed viewed **50% more** pages than visitors in the bottom ten percentile.

SOURCE: AOL



400 MILLISECONDS

YAHOO!

Increased traffic by 9% for every 400 milliseconds of improvement.

SOURCE: Yahoo!

strangeloop

Performance directly impacts the bottom line

HOW FAST is fast enough?

57%

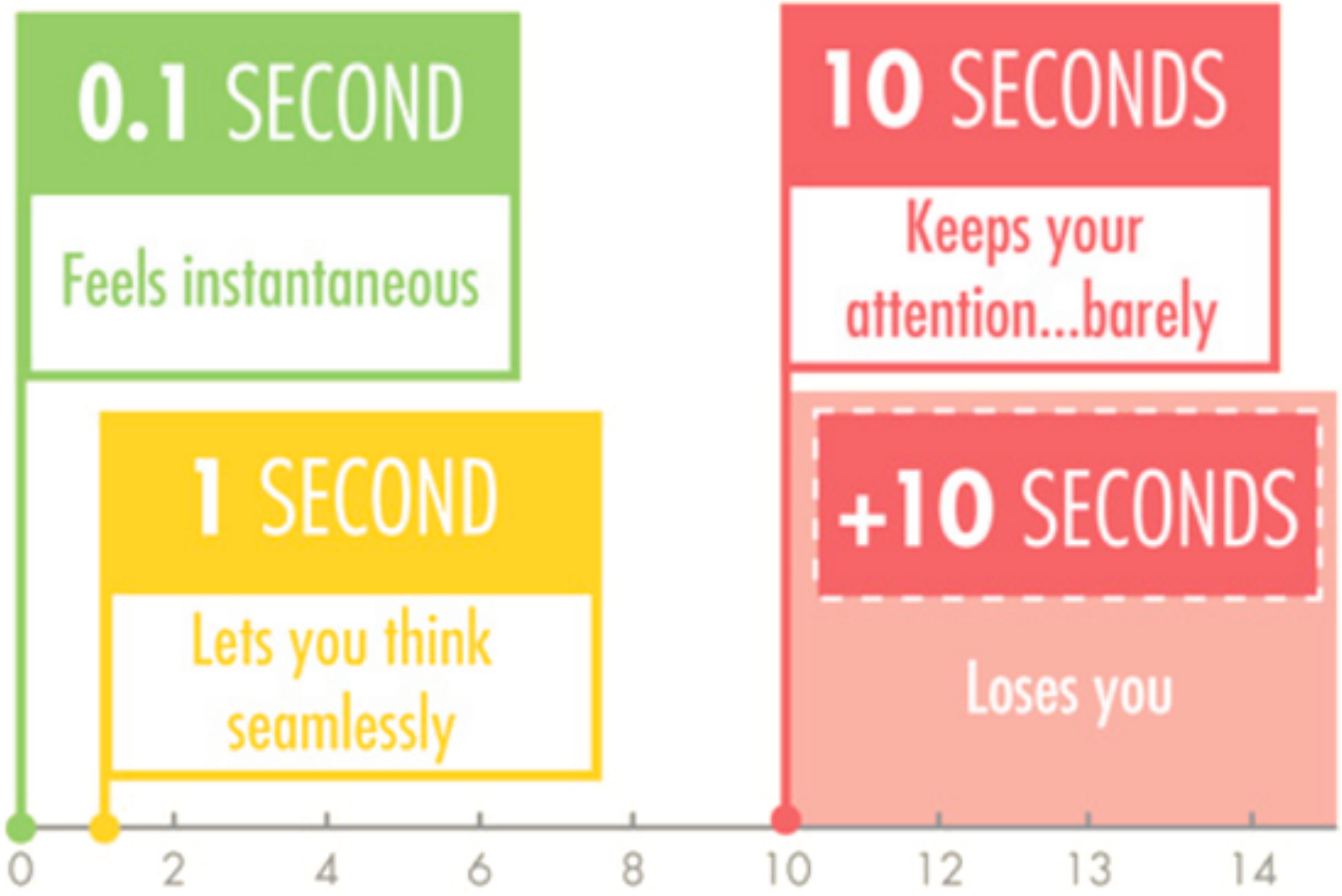
of online consumers WILL ABANDON A SLOW PAGE AFTER 3 SECONDS

> We want you to be able to flick from one page to another **as quickly as you can flick a page in a book.**

SO WE'RE aiming very, very HIGH ...*at something like* 100 MILLISECONDS

Urs Hölzle,
Senior VP Operations, Google

HOW YOUR BRAIN
perceives page load times



0.1 SECOND
Feels instantaneous

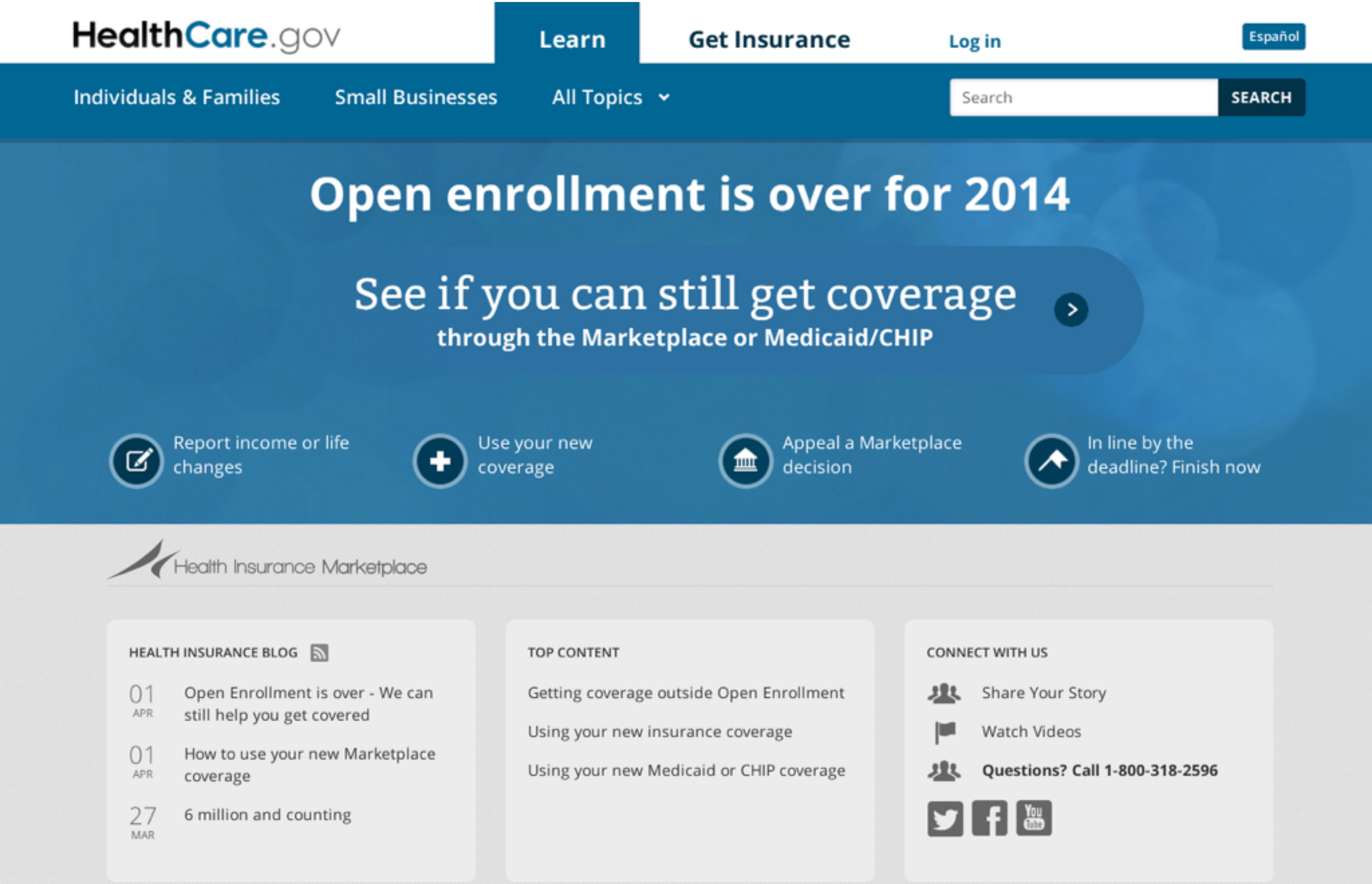
1 SECOND
Lets you think seamlessly

10 SECONDS
Keeps your attention...barely

+10 SECONDS
Loses you

slow websites lead to WEB STRESS

Studies show that we have to **CONCENTRATE** UP TO **50%** **HARDER** when using **SLOW WEBSITES**



Open enrollment is over for 2014

See if you can still get coverage

through the Marketplace or Medicaid/CHIP

Report income or life changes

Use your new coverage

Appeal a Marketplace decision

In line by the deadline? Finish now

HEALTH INSURANCE BLOG

01 APR	Open Enrollment is over - We can still help you get covered
01 APR	How to use your new Marketplace coverage
27 MAR	6 million and counting

TOP CONTENT

Getting coverage outside Open Enrollment

Using your new insurance coverage

Using your new Medicaid or CHIP coverage

CONNECT WITH US

Share Your Story

Watch Videos

Questions? Call 1-800-318-2596

Tools of the trade for performance testing

Understand your baseline performance

Static

vs

Hello World

vs

Applications

Apache Bench

```
ab -c 10 -t 10 -k http://dustinwhittle.com/
```

```
Benchmarking dustinwhittle.com (be patient)
Finished 286 requests


Server Software:        nginx
Server Hostname:        dustinwhittle.com
Server Port:            80

Document Path:          /
Document Length:        6642 bytes

Concurrency Level:      10
Time taken for tests:   10.042 seconds
Complete requests:      286
Failed requests:        0
Write errors:           0
Keep-Alive requests:    0
Total transferred:      2080364 bytes
HTML transferred:       1899612 bytes
Requests per second:    28.48 [#/sec] (mean)
Time per request:       351.133 [ms] (mean)
Time per request:       35.113 [ms] (mean, across all concurrent
requests)
Transfer rate:          202.30 [Kbytes/sec] received
```

Siege

```
siege -c 10 -b -t 10S http://dustinwhittle.com/
```

```
** SIEGE 2.72
** Preparing 10 concurrent users for battle.
The server is now under siege...
Lifting the server siege...      done.

Transactions:               263 hits
Availability:               100.00 %
Elapsed time:               9.36 secs
Data transferred:           0.35 MB
Response time:              0.35 secs
Transaction rate:           28.10 trans/sec
Throughput:                 0.04 MB/sec
Concurrency:                9.82
Successful transactions:    263
Failed transactions:         0
Longest transaction:         0.54
Shortest transaction:        0.19
```

Crawl the entire app
to discover all urls

```
sproxy -o ./urls.txt
```

```
SPROXY v1.02 listening on port 9001
...appending HTTP requests to: ./urls.txt
...default connection timeout: 120 seconds
```

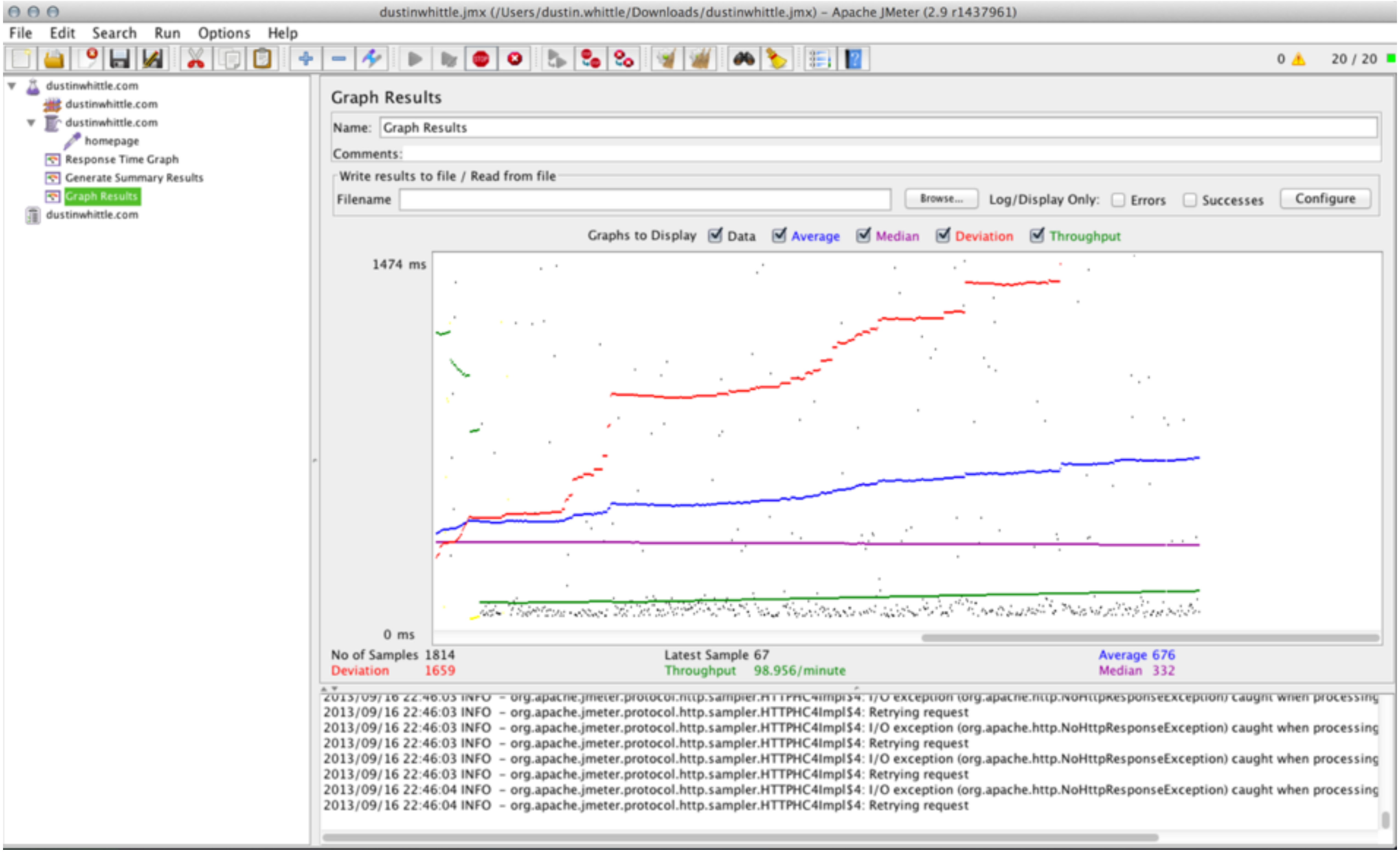
```
wget -r -o verbose.txt -l 0 -t 1 --spider -w 1 -e
robots=on
-e "http_proxy = http://127.0.0.1:9001"
"http://dustinwhittle.com/"
```

```
sort -u -o urls.txt urls.txt
```

Benchmark traffic across all unique urls with siege

```
siege -v -c 50 -i -t 3M -f urls.txt -d 10
```

Apache JMeter



Name: Graph Results

Comments:

Filename

Log/Display Only: □ Errors □ Successes

Graphs to Display ☑ Data ☑ Average ☑ Median ☑ Deviation ☑ Throughput

No of Samples 1814
Deviation 1659
Latest Sample 67
Throughput 98.956/minute
Average 676
Median 332

```
2013/09/16 22:46:03 INFO  - org.apache.jmeter.protocol.http.sampler.HTTPHC4Impl$4: I/O exception (org.apache.http.NoHttpResponseException) caught when processing
2013/09/16 22:46:03 INFO  - org.apache.jmeter.protocol.http.sampler.HTTPHC4Impl$4: Retrying request
2013/09/16 22:46:03 INFO  - org.apache.jmeter.protocol.http.sampler.HTTPHC4Impl$4: I/O exception (org.apache.http.NoHttpResponseException) caught when processing
2013/09/16 22:46:03 INFO  - org.apache.jmeter.protocol.http.sampler.HTTPHC4Impl$4: Retrying request
2013/09/16 22:46:03 INFO  - org.apache.jmeter.protocol.http.sampler.HTTPHC4Impl$4: I/O exception (org.apache.http.NoHttpResponseException) caught when processing
2013/09/16 22:46:03 INFO  - org.apache.jmeter.protocol.http.sampler.HTTPHC4Impl$4: Retrying request
2013/09/16 22:46:04 INFO  - org.apache.jmeter.protocol.http.sampler.HTTPHC4Impl$4: I/O exception (org.apache.http.NoHttpResponseException) caught when processing
2013/09/16 22:46:04 INFO  - org.apache.jmeter.protocol.http.sampler.HTTPHC4Impl$4: Retrying request
```

Multi-Mechanize

Multi-Mechanize is an open source framework for performance and load testing

```
pip install multi-mechanize
```

multimech-newproject demo

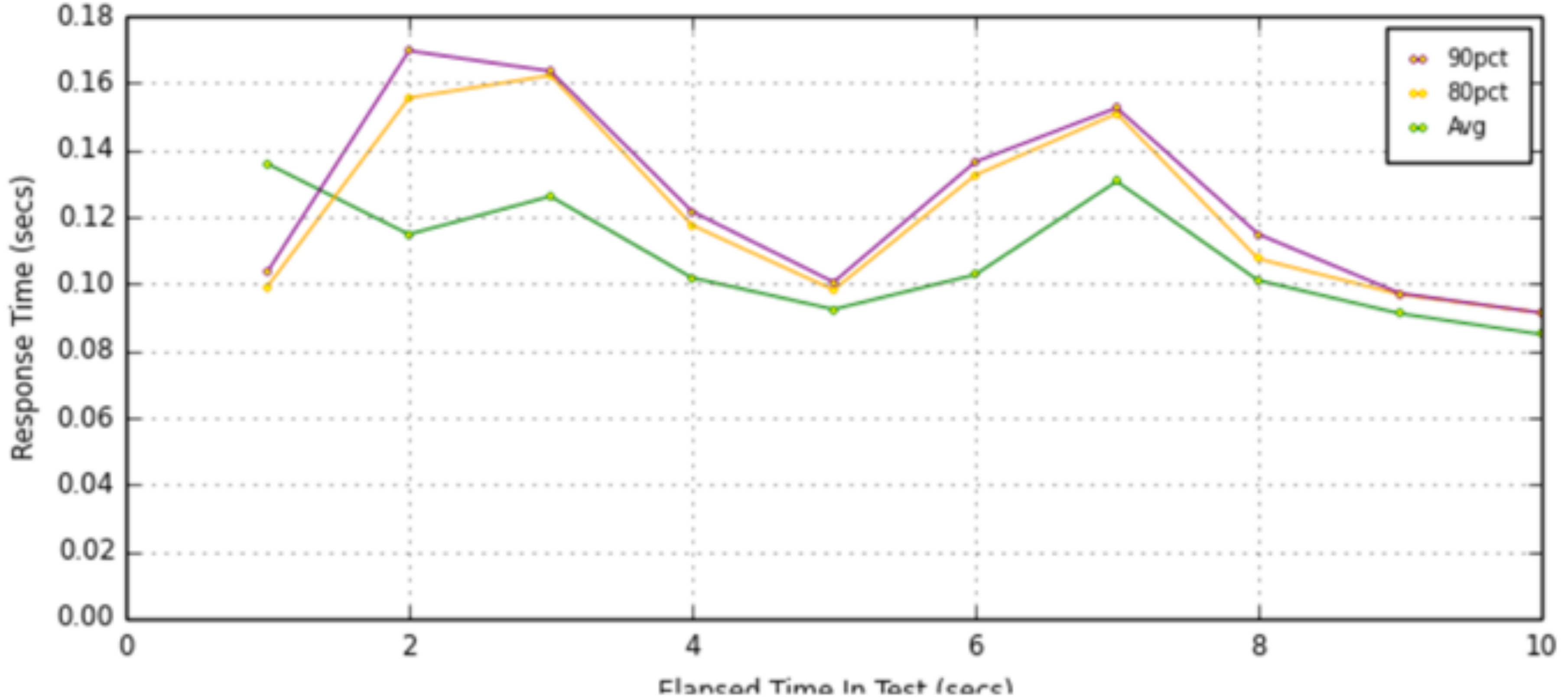
```python
import requests

class Transaction(object):
    def run(self):
        r = requests.get('http://dustinwhittle.com/')
        r.raw.read()
```

```
import mechanize
import time

class Transaction(object):
    def run(self):
        br = mechanize.Browser()
        br.set_handle_robots(False)

        start_timer = time.time()
        resp = br.open('http://www.dustinwhittle.com/')
        resp.read()
        latency = time.time() - start_timer
        self.custom_timers['homepage'] = latency

        start_timer = time.time()
        resp = br.open('http://www.dustinwhittle.com/blog')
        resp.read()
        latency = time.time() - start_timer
        self.custom_timers['blog'] = latency

        assert (resp.code == 200)
```

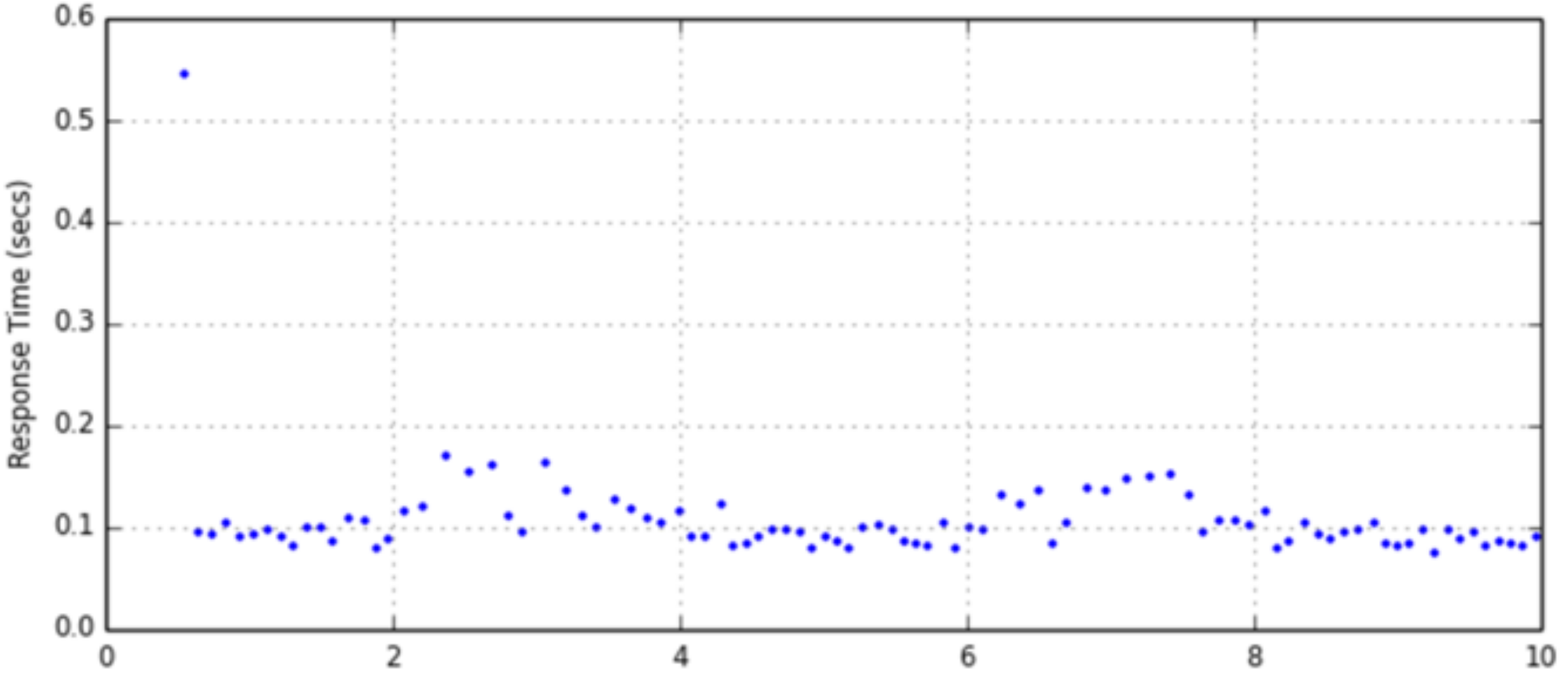
```
[global]
run_time = 10
rampup = 5
results_ts_interval = 1
progress_bar = on
console_logging = off
xml_report = on


[user_group-1]
threads = 1
script = demo.py
```

multimech-run demo

Response Time: 1 sec time-series



Response Time: raw data (all points)



What about when you need more than one machine?

Who lives in the cloud?

Bees with Machine Guns

A utility for arming (*creating*)
many bees (*micro EC2 instances*)
to attack (*load test*)
targets (*web applications*)

```
pip install beeswithmachineguns
```

```
# ~/.boto

[Credentials]

aws_access_key_id=xxx
aws_secret_access_key=xxx

[Boto]

ec2_region_name = us-west-2
ec2_region_endpoint = ec2.us-west-2.amazonaws.com
```

```
bees up -s 2 -g default -z us-west-2b -i ami-bc05898c -k appdynamics-dustinwhittle-aws-us-west-2 -l ec2-user
```

```
Connecting to the hive.
Attempting to call up 2 bees.
Waiting for bees to load their machine
guns...
.
.
.
.
Bee i-3828400c is ready for the attack.
Bee i-3928400d is ready for the attack.
The swarm has assembled 2 bees.
```

bees report

```
Read 2 bees from the roster.
Bee i-3828400c: running @ 54.212.22.176
Bee i-3928400d: running @ 50.112.6.191
```

```
bees attack -n 1000 -c 50 -u
http://dustinwhittle.com/
```

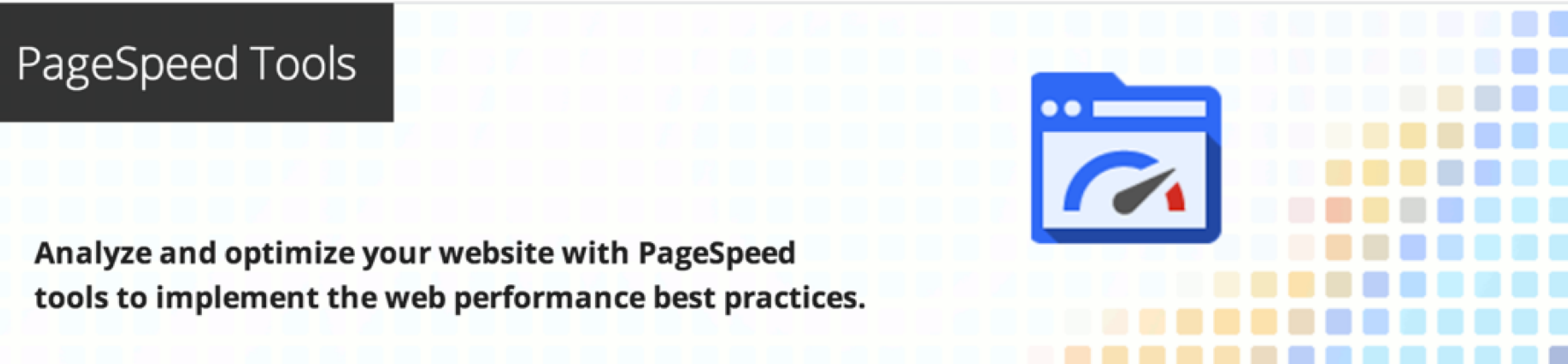
```
Read 2 bees from the roster.
Connecting to the hive.
Assembling bees.
Each of 2 bees will fire 50000 rounds, 125 at a time.
Stinging URL so it will be cached for the attack.
Organizing the swarm.
Bee 0 is joining the swarm.
Bee 1 is joining the swarm.
Bee 0 is firing his machine gun. Bang bang!
Bee 1 is firing his machine gun. Bang bang!
Bee 1 is out of ammo.
Bee 0 is out of ammo.
Offensive complete.
     Complete requests:      100000
     Requests per second: 1067.110000 [#/sec] (mean)
     Time per request:    278.348000 [ms] (mean)
     50% response time:      47.500000 [ms] (mean)
     90% response time:      114.000000 [ms] (mean)
Mission Assessment: Target crushed bee offensive.
The swarm is awaiting new orders.
```

bees down

What about the client side?

In modern web applications more latency comes from the client-side than the server-side.

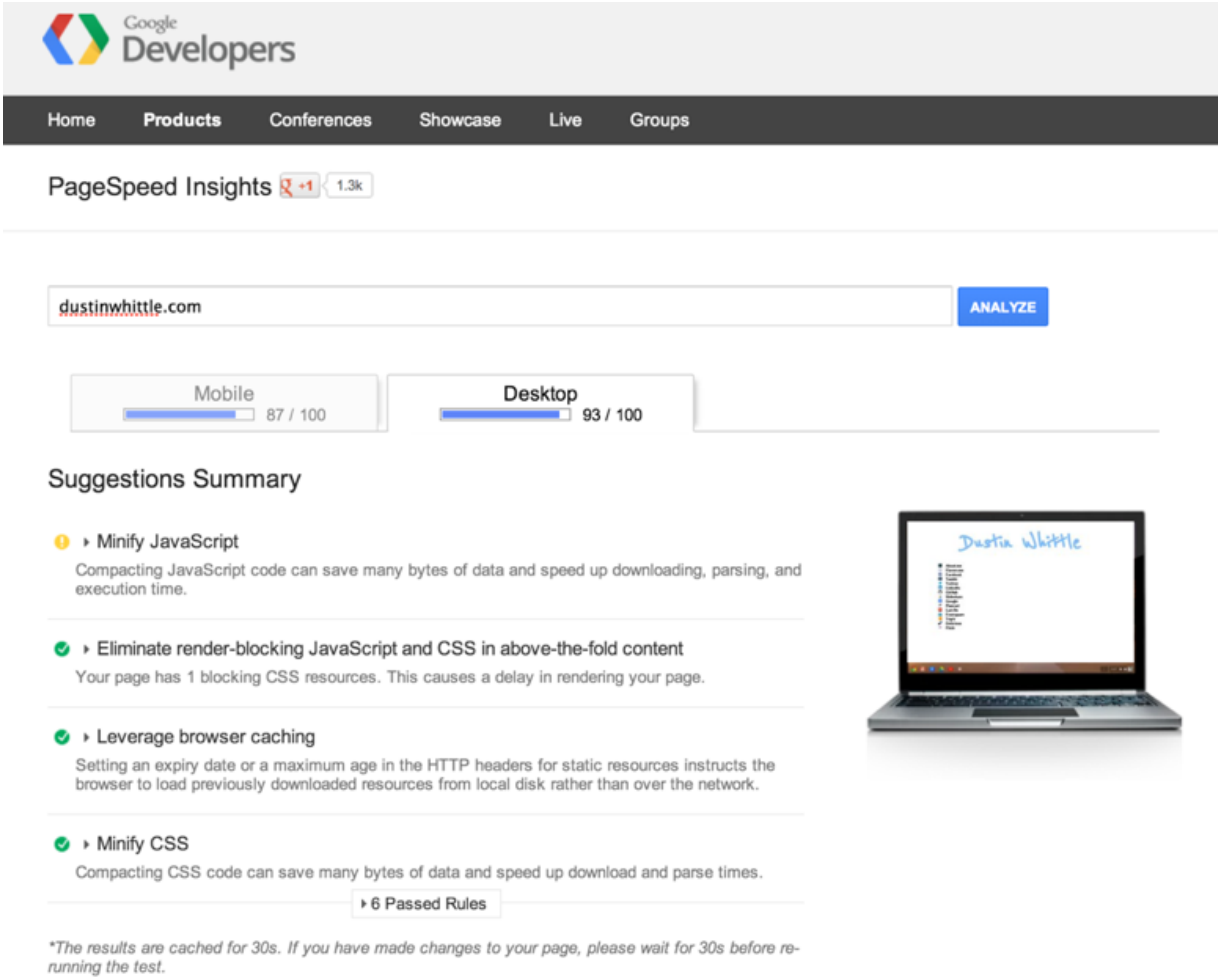
Google PageSpeed



PageSpeed Tools

Analyze and optimize your website with PageSpeed tools to implement the web performance best practices.

Google PageSpeed Insights



PageSpeed Insights

dustinwhittle.com

Mobile 87 / 100

Desktop 93 / 100

Suggestions Summary

- Minify JavaScript

  Compacting JavaScript code can save many bytes of data and speed up downloading, parsing, and execution time.

- ✓ Eliminate render-blocking JavaScript and CSS in above-the-fold content

  Your page has 1 blocking CSS resources. This causes a delay in rendering your page.

- ✓ Leverage browser caching

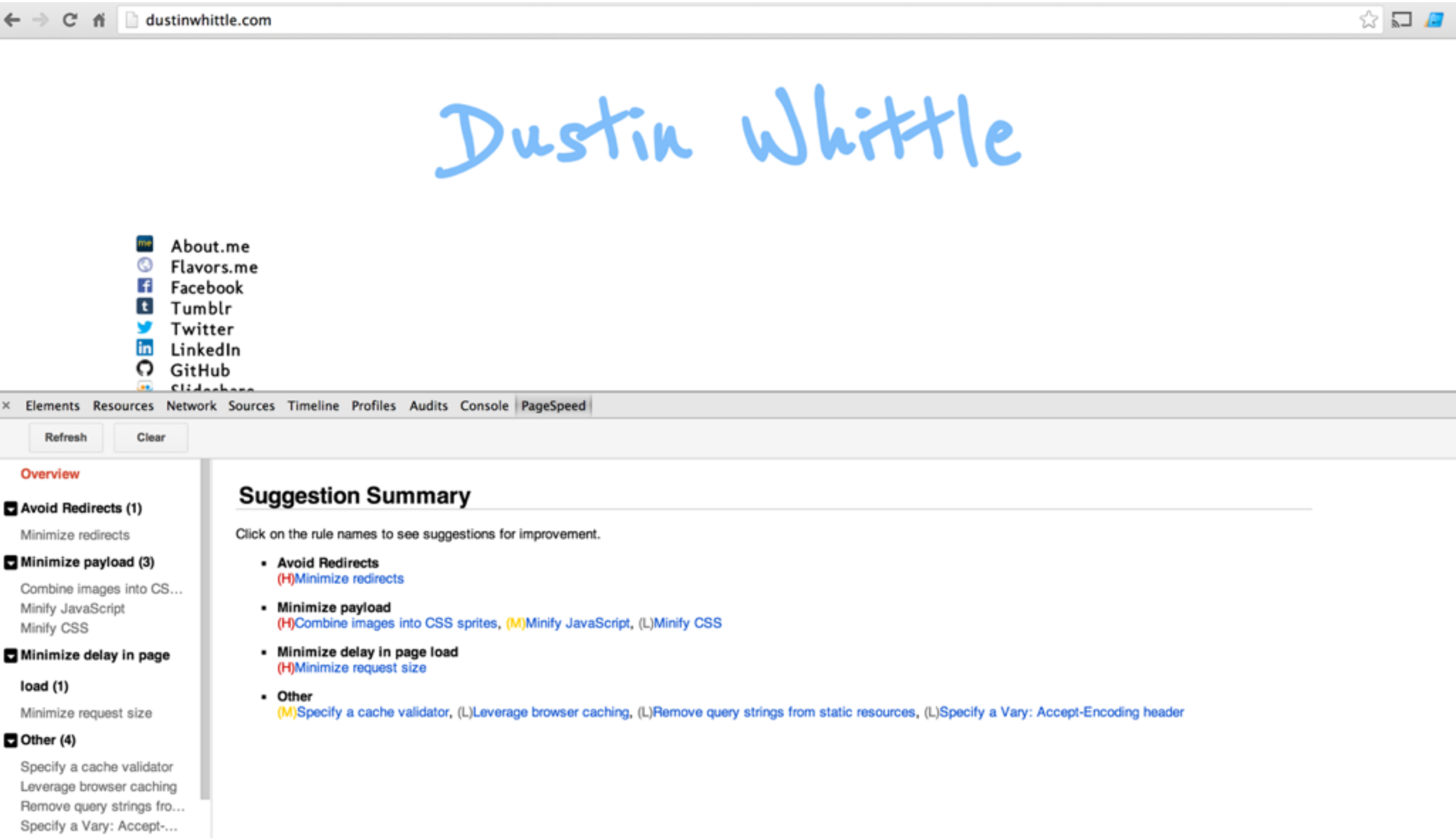
  Setting an expiry date or a maximum age in the HTTP headers for static resources instructs the browser to load previously downloaded resources from local disk rather than over the network.

- ✓ Minify CSS

  Compacting CSS code can save many bytes of data and speed up download and parse times.

6 Passed Rules

**The results are cached for 30s. If you have made changes to your page, please wait for 30s before re-running the test.*



Suggestion Summary

Click on the rule names to see suggestions for improvement.

- **Avoid Redirects**
  (H)Minimize redirects
- **Minimize payload**
  (H)Combine images into CSS sprites, (M)Minify JavaScript, (L)Minify CSS
- **Minimize delay in page load**
  (H)Minimize request size
- **Other**
  (M)Specify a cache validator, (L)Leverage browser caching, (L)Remove query strings from static resources, (L)Specify a Vary: Accept-Encoding header

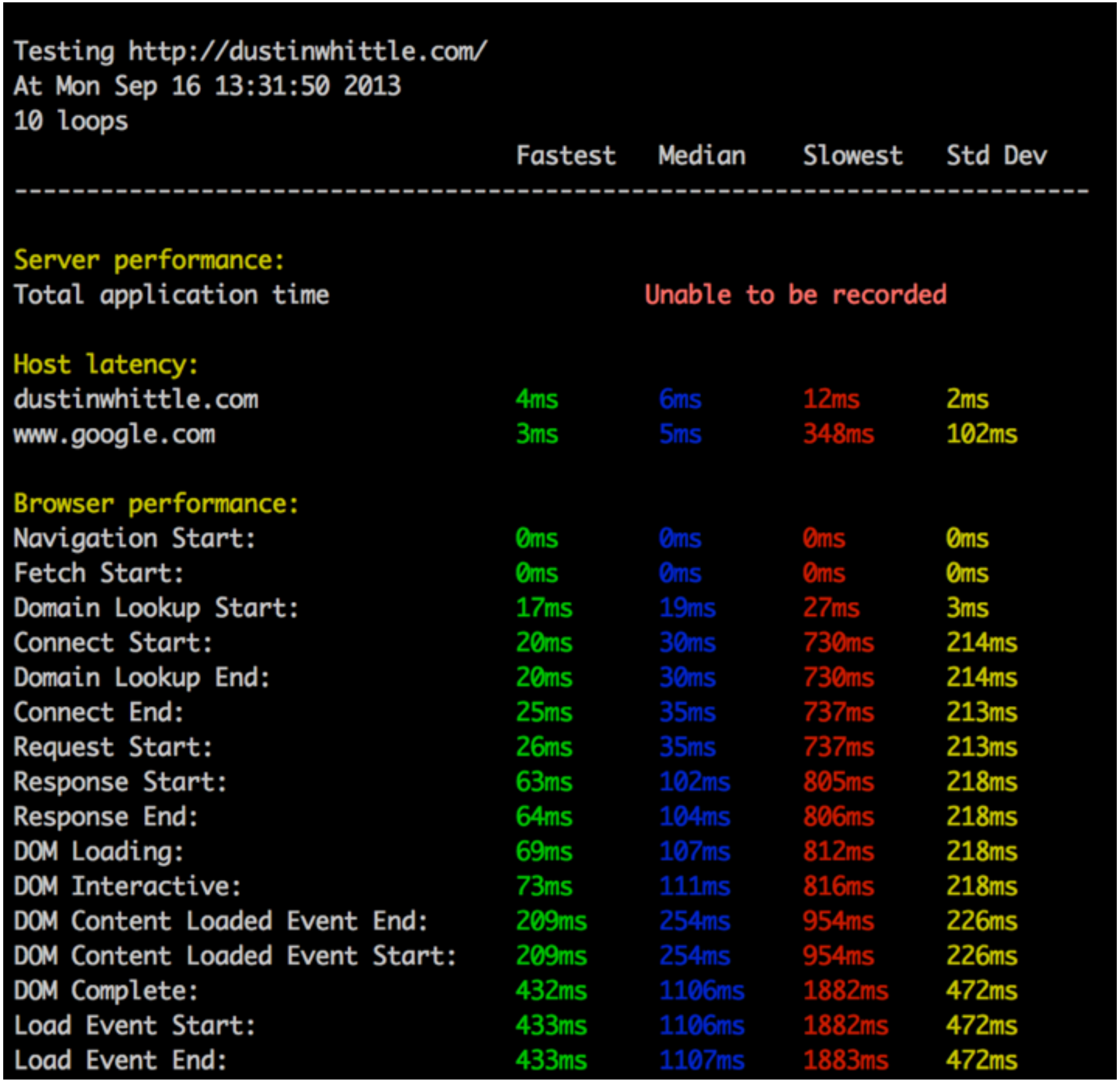
Google PageSpeed API

```
curl "https://www.googleapis.com/
pagespeedonline/v1/runPagespeed?
url=http://dustinwhittle.com/&key=xxx"
```

WBench

```
gem install wbench
```

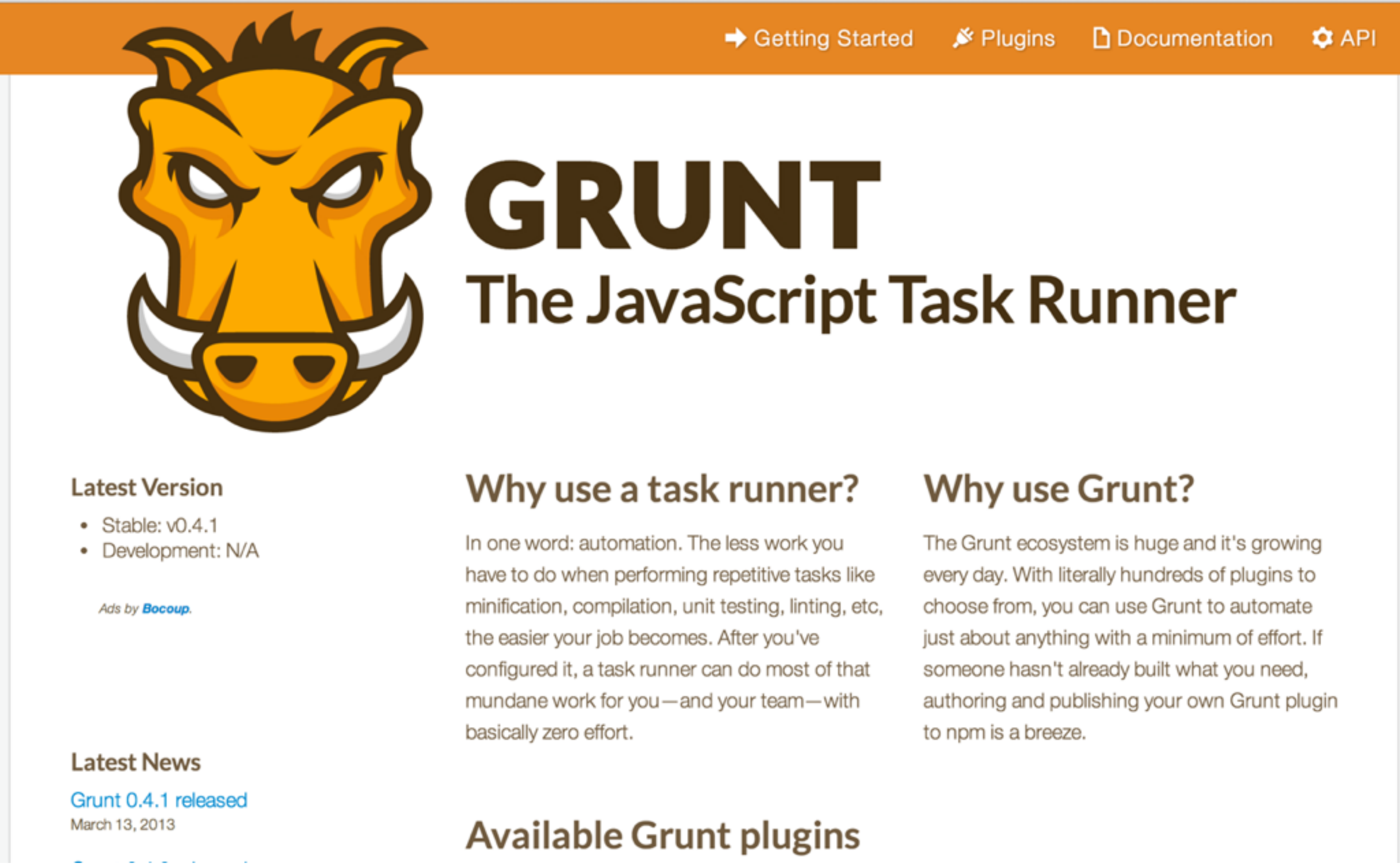
```
wbench http://dustinwhittle.com/
```



```
Testing http://dustinwhittle.com/
At Mon Sep 16 13:31:50 2013
10 loops
                                   Fastest   Median    Slowest   Std Dev
---------------------------------------------------------------------------

Server performance:
Total application time                      Unable to be recorded

Host latency:
dustinwhittle.com                  4ms       6ms       12ms      2ms
www.google.com                     3ms       5ms       348ms     102ms

Browser performance:
Navigation Start:                  0ms       0ms       0ms       0ms
Fetch Start:                       0ms       0ms       0ms       0ms
Domain Lookup Start:               17ms      19ms      27ms      3ms
Connect Start:                     20ms      30ms      730ms     214ms
Domain Lookup End:                 20ms      30ms      730ms     214ms
Connect End:                       25ms      35ms      737ms     213ms
Request Start:                     26ms      35ms      737ms     213ms
Response Start:                    63ms      102ms     805ms     218ms
Response End:                      64ms      104ms     806ms     218ms
DOM Loading:                       69ms      107ms     812ms     218ms
DOM Interactive:                   73ms      111ms     816ms     218ms
DOM Content Loaded Event End:      209ms     254ms     954ms     226ms
DOM Content Loaded Event Start:    209ms     254ms     954ms     226ms
DOM Complete:                      432ms     1106ms    1882ms    472ms
Load Event Start:                  433ms     1106ms    1882ms    472ms
Load Event End:                    433ms     1107ms    1883ms    472ms
```

Automate client-side performance testing with Grunt

Use Bower (for dependencies),
Grunt (for automation),
and Yeoman (for bootstrapping)



GRUNT

The JavaScript Task Runner

Latest Version

- Stable: v0.4.1
- Development: N/A

Latest News

Grunt 0.4.1 released
March 13, 2013

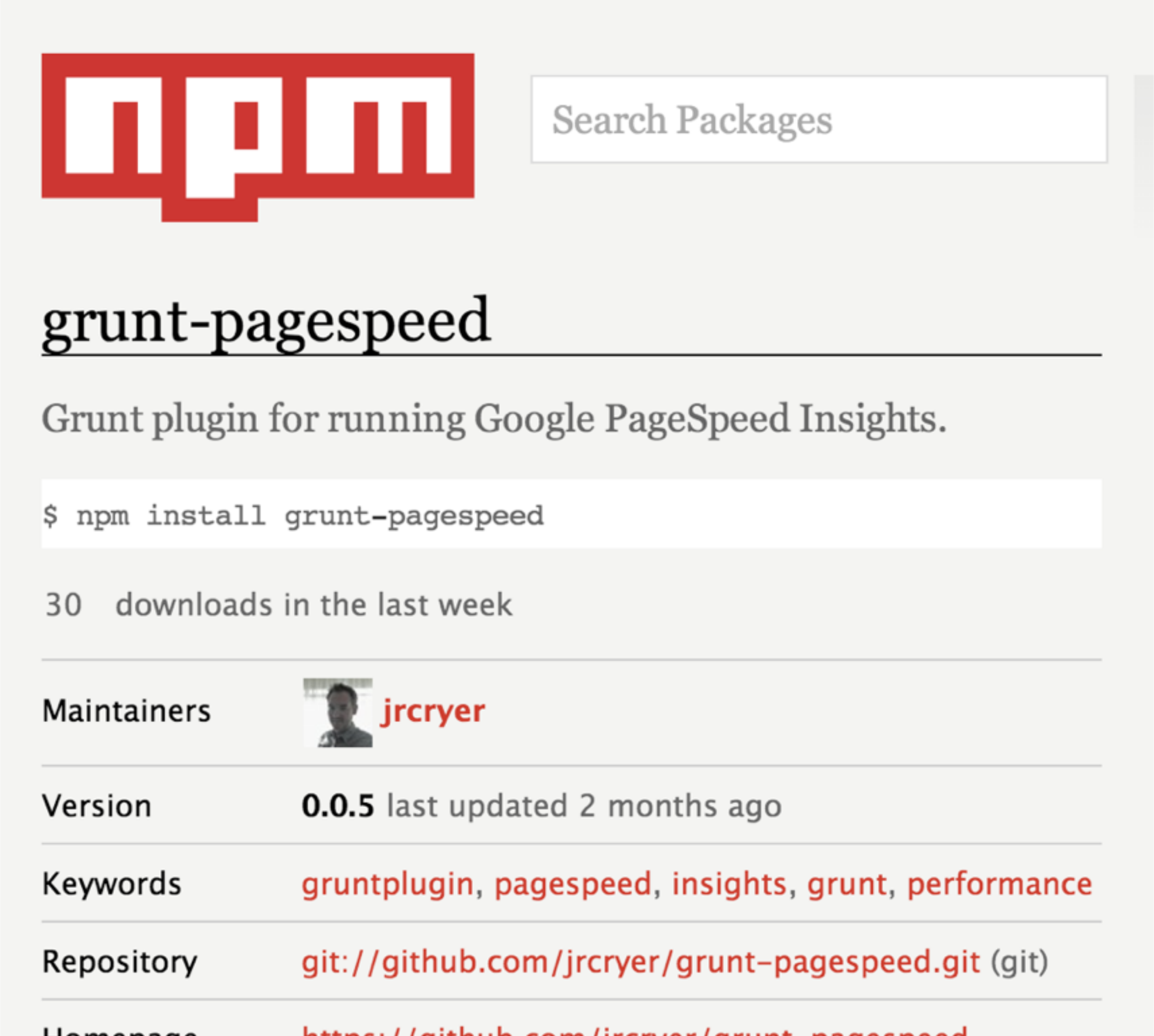
Why use a task runner?

In one word: automation. The less work you have to do when performing repetitive tasks like minification, compilation, unit testing, linting, etc, the easier your job becomes. After you've configured it, a task runner can do most of that mundane work for you—and your team—with basically zero effort.

Why use Grunt?

The Grunt ecosystem is huge and it's growing every day. With literally hundreds of plugins to choose from, you can use Grunt to automate just about anything with a minimum of effort. If someone hasn't already built what you need, authoring and publishing your own Grunt plugin to npm is a breeze.

Available Grunt plugins



grunt-pagespeed

Grunt plugin for running Google PageSpeed Insights.

```
$ npm install grunt-pagespeed
```

30 downloads in the last week

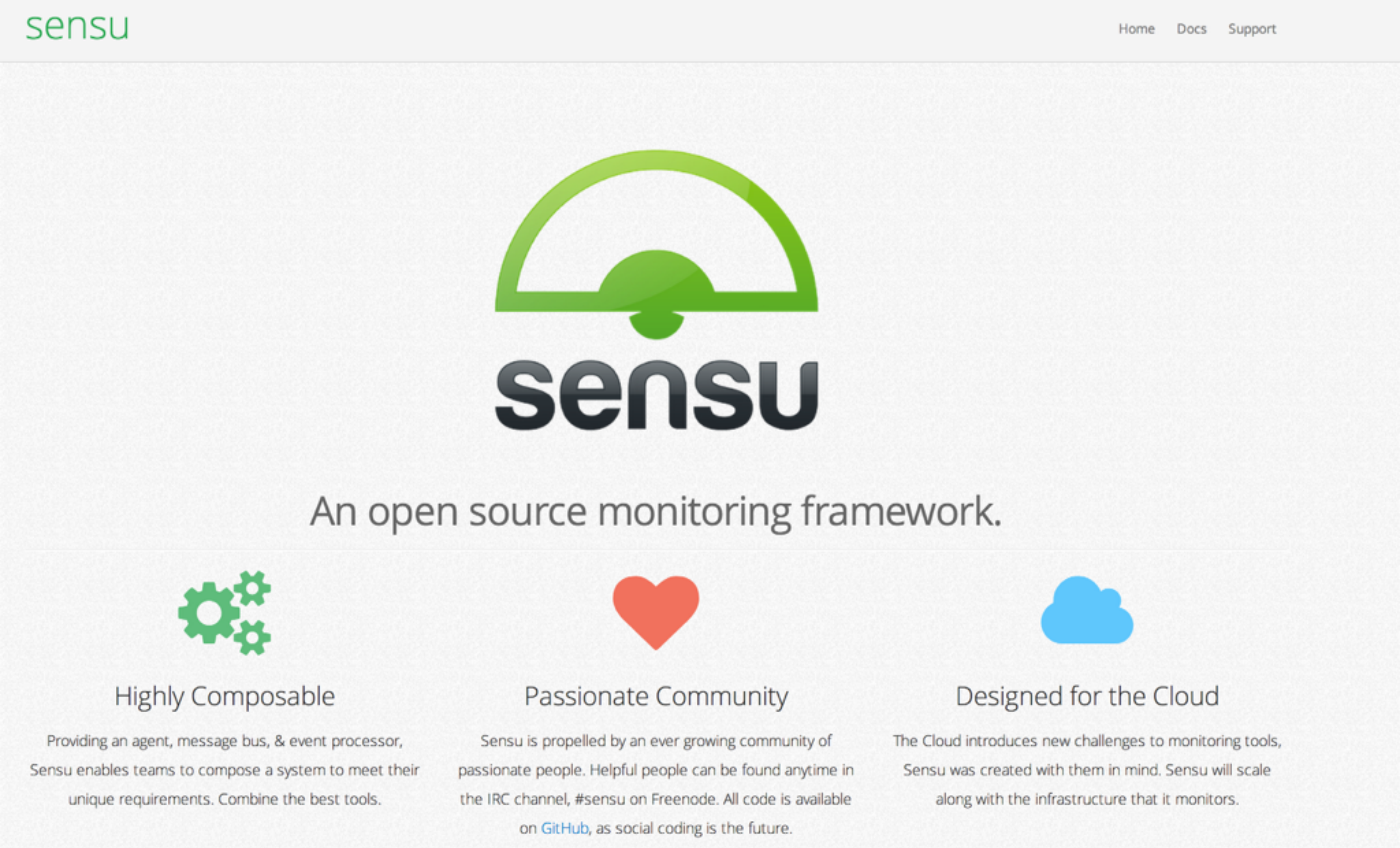
Maintainers	jrcryer
Version	**0.0.5** last updated 2 months ago
Keywords	gruntplugin, pagespeed, insights, grunt, performance
Repository	git://github.com/jrcryer/grunt-pagespeed.git (git)
Homepage	https://github.com/jrcryer/grunt-pagespeed

How many people understand exactly how fast their site runs in production?

Track performance in development and production

Instrument everything = code, databases, caches, queues, third party services, and infrastructure.

Chef / Sensu



An open source monitoring framework.

Highly Composable

Providing an agent, message bus, & event processor, Sensu enables teams to compose a system to meet their unique requirements. Combine the best tools.

Passionate Community

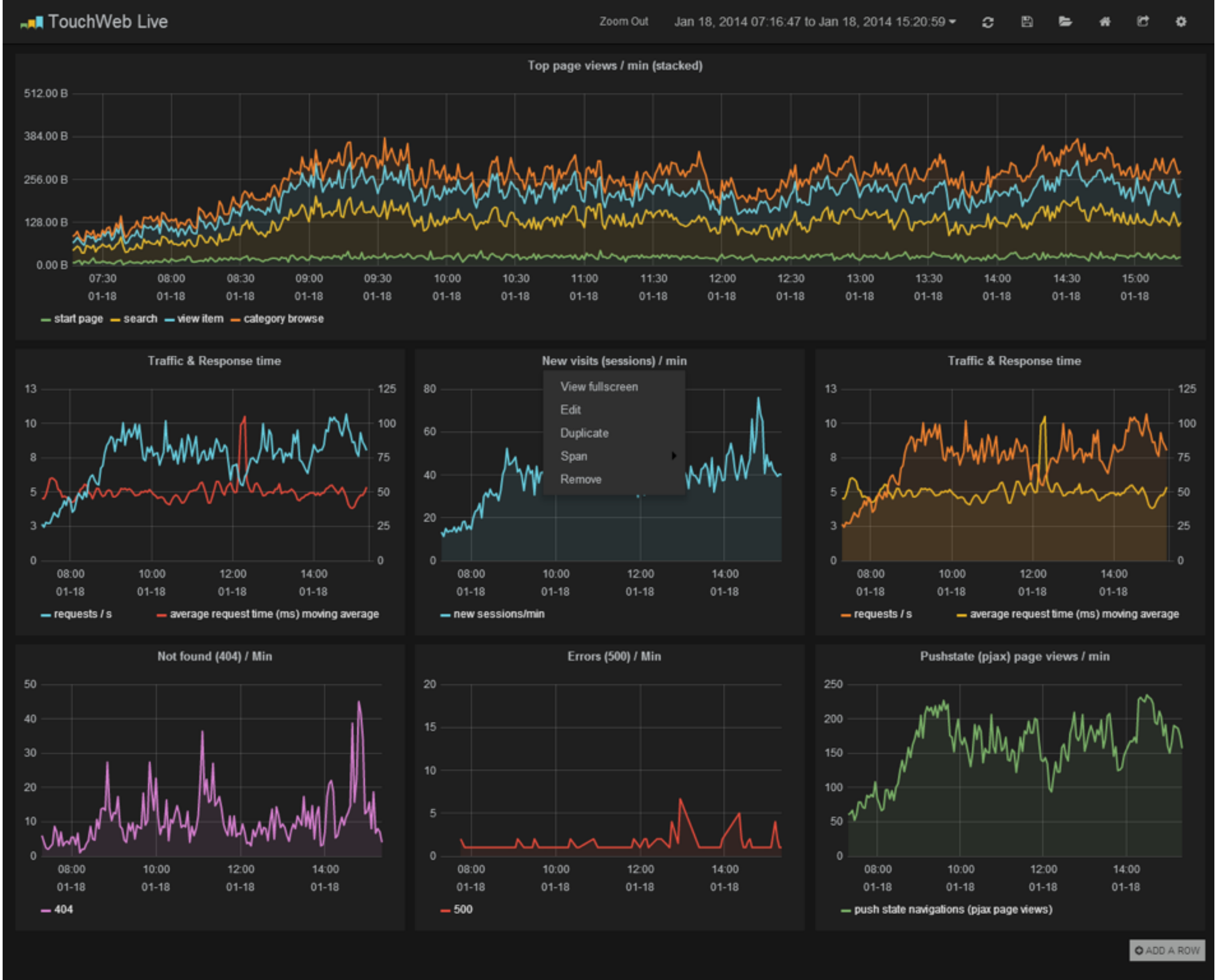
Sensu is propelled by an ever growing community of passionate people. Helpful people can be found anytime in the IRC channel, #sensu on Freenode. All code is available on GitHub, as social coding is the future.

Designed for the Cloud

The Cloud introduces new challenges to monitoring tools, Sensu was created with them in mind. Sensu will scale along with the infrastructure that it monitors.

http://sensuapp.org/

Statsd + Graphite + Grafana



TouchWeb Live

Top page views / min (stacked)

Traffic & Response time

New visits (sessions) / min

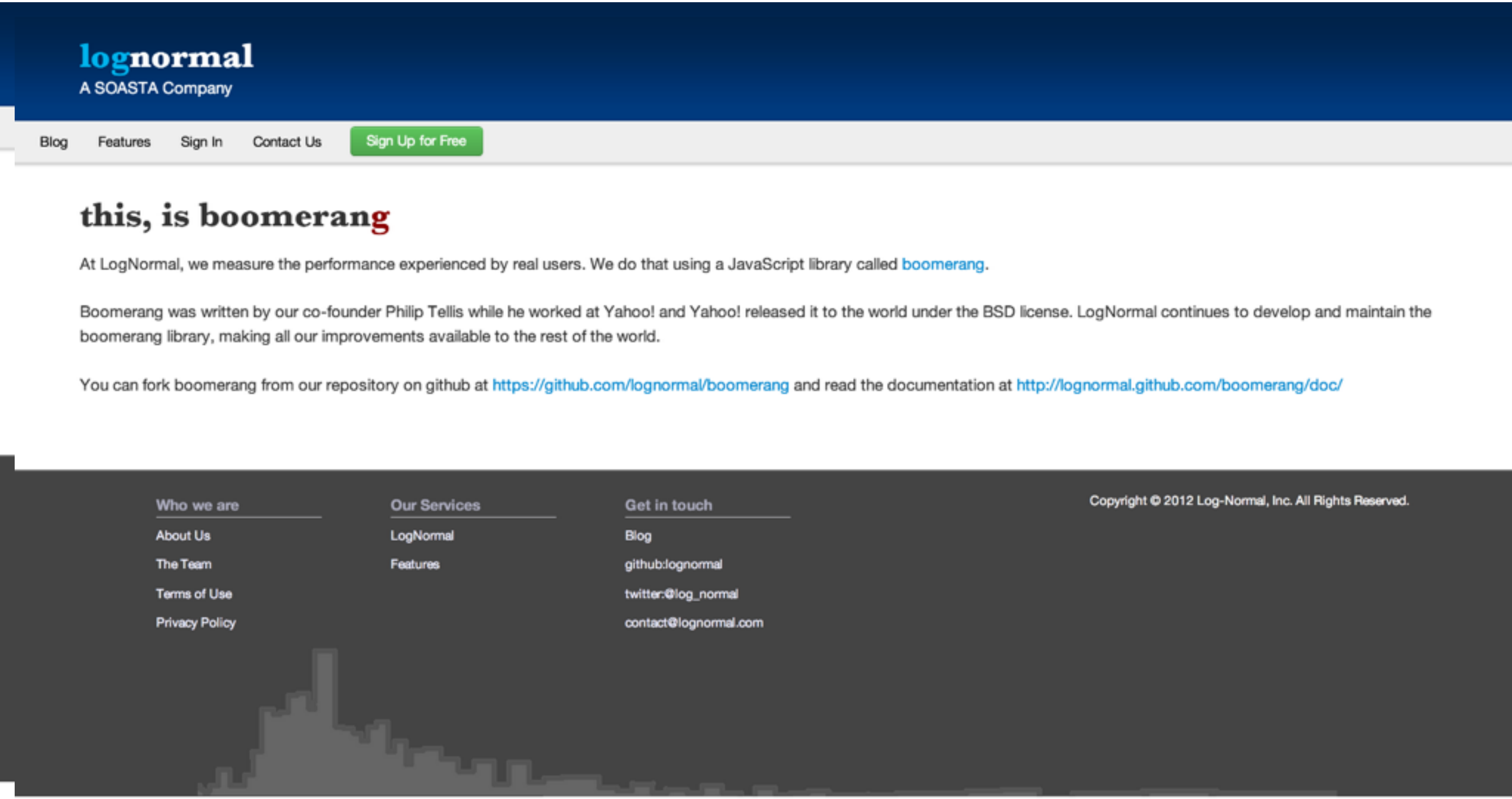
Traffic & Response time

Not found (404) / Min

Errors (500) / Min

Pushstate (pjax) page views / min

Episodes / Boomerang

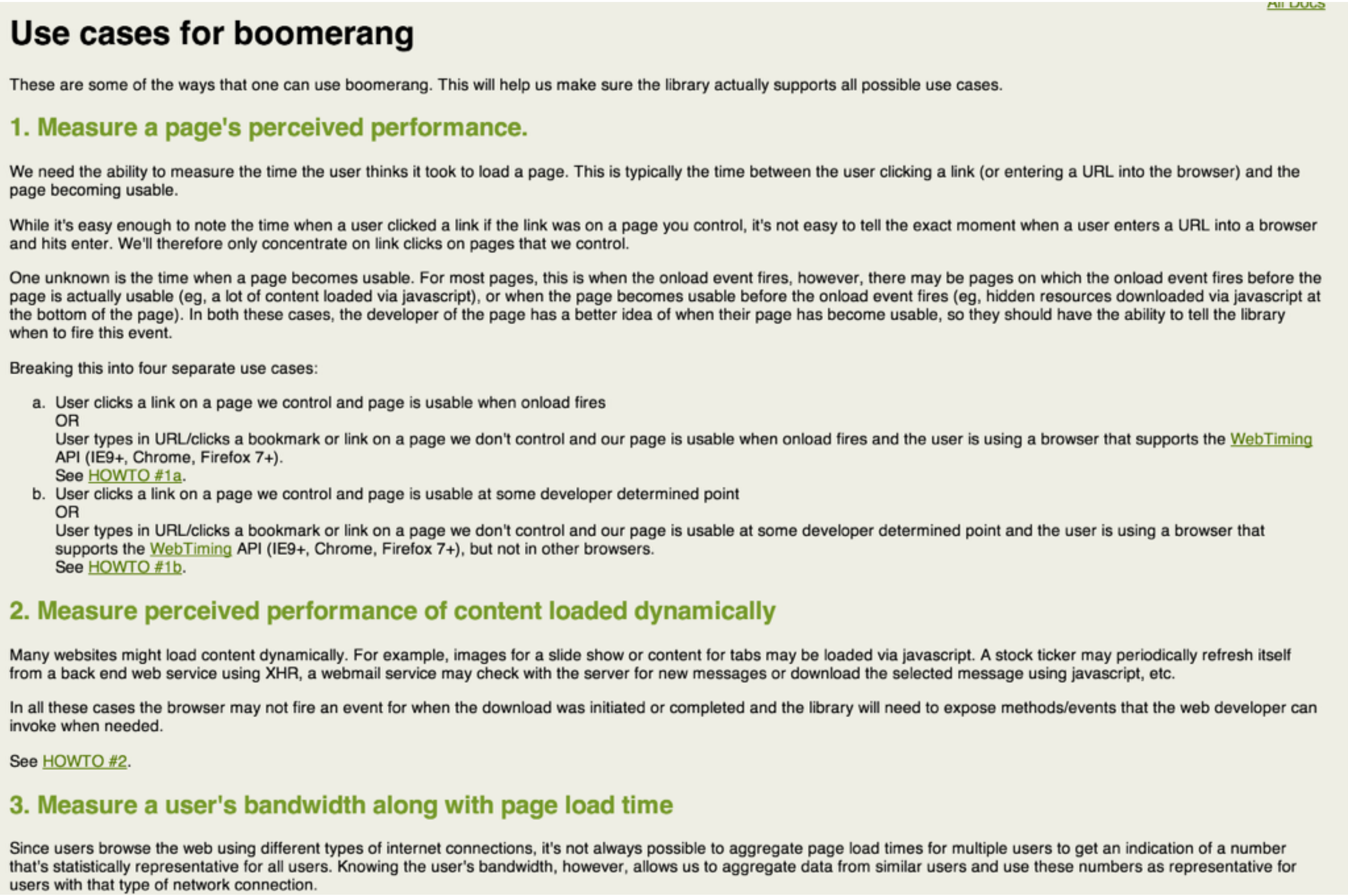


this, is boomerang

At LogNormal, we measure the performance experienced by real users. We do that using a JavaScript library called boomerang.

Boomerang was written by our co-founder Philip Tellis while he worked at Yahoo! and Yahoo! released it to the world under the BSD license. LogNormal continues to develop and maintain the boomerang library, making all our improvements available to the rest of the world.

You can fork boomerang from our repository on github at https://github.com/lognormal/boomerang and read the documentation at http://lognormal.github.com/boomerang/doc/



All Docs

Use cases for boomerang

These are some of the ways that one can use boomerang. This will help us make sure the library actually supports all possible use cases.

1. Measure a page's perceived performance.

We need the ability to measure the time the user thinks it took to load a page. This is typically the time between the user clicking a link (or entering a URL into the browser) and the page becoming usable.

While it's easy enough to note the time when a user clicked a link if the link was on a page you control, it's not easy to tell the exact moment when a user enters a URL into a browser and hits enter. We'll therefore only concentrate on link clicks on pages that we control.

One unknown is the time when a page becomes usable. For most pages, this is when the onload event fires, however, there may be pages on which the onload event fires before the page is actually usable (eg, a lot of content loaded via javascript), or when the page becomes usable before the onload event fires (eg, hidden resources downloaded via javascript at the bottom of the page). In both these cases, the developer of the page has a better idea of when their page has become usable, so they should have the ability to tell the library when to fire this event.

Breaking this into four separate use cases:

a. User clicks a link on a page we control and page is usable when onload fires
   OR
   User types in URL/clicks a bookmark or link on a page we don't control and our page is usable when onload fires and the user is using a browser that supports the WebTiming API (IE9+, Chrome, Firefox 7+).
   See HOWTO #1a.
b. User clicks a link on a page we control and page is usable at some developer determined point
   OR
   User types in URL/clicks a bookmark or link on a page we don't control and our page is usable at some developer determined point and the user is using a browser that supports the WebTiming API (IE9+, Chrome, Firefox 7+), but not in other browsers.
   See HOWTO #1b.

2. Measure perceived performance of content loaded dynamically

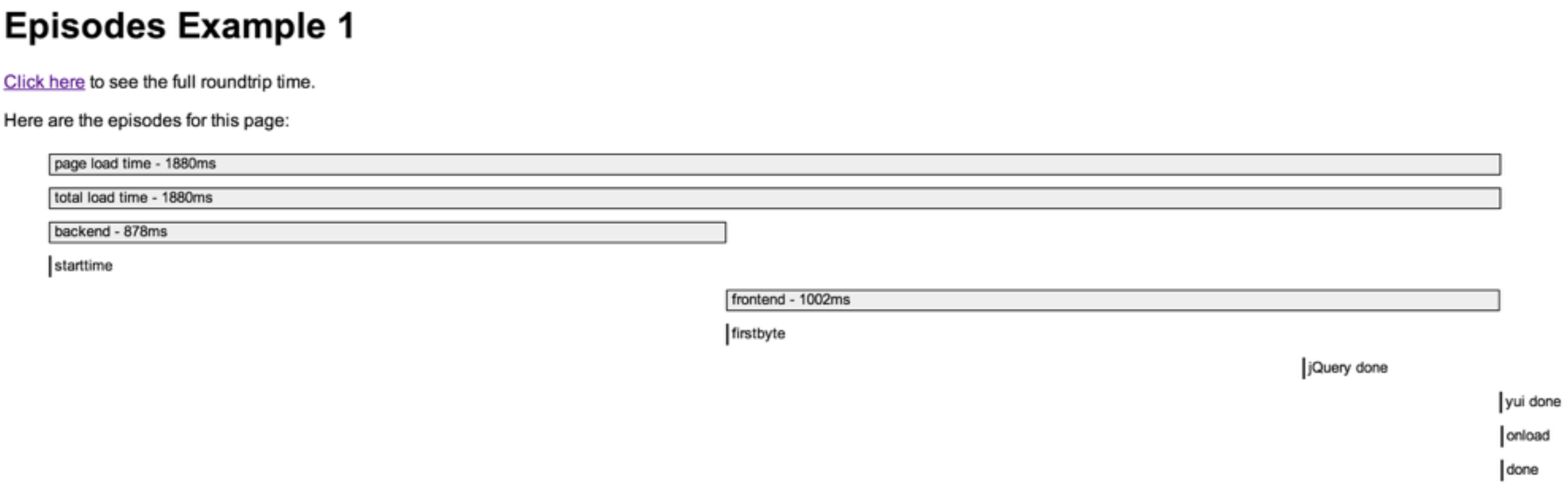
Many websites might load content dynamically. For example, images for a slide show or content for tabs may be loaded via javascript. A stock ticker may periodically refresh itself from a back end web service using XHR, a webmail service may check with the server for new messages or download the selected message using javascript, etc.

In all these cases the browser may not fire an event for when the download was initiated or completed and the library will need to expose methods/events that the web developer can invoke when needed.

See HOWTO #2.

3. Measure a user's bandwidth along with page load time

Since users browse the web using different types of internet connections, it's not always possible to aggregate page load times for multiple users to get an indication of a number that's statistically representative for all users. Knowing the user's bandwidth, however, allows us to aggregate data from similar users and use these numbers as representative for users with that type of network connection.

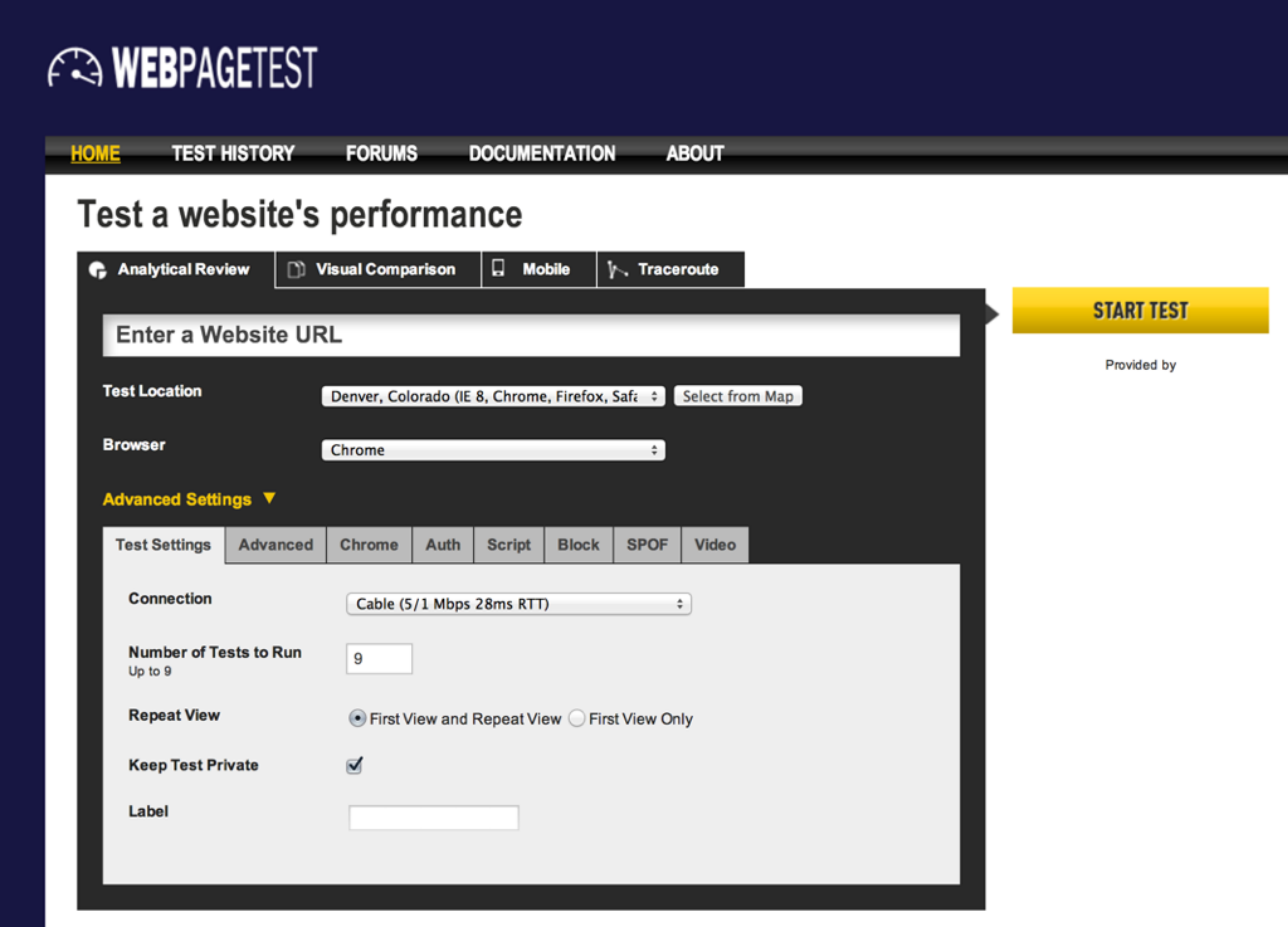


Episodes Example 1

Click here to see the full roundtrip time.

Here are the episodes for this page:

webpagetest.org



Test a website's performance

Enter a Website URL

Test Location: Denver, Colorado (IE 8, Chrome, Firefox, Safa | Select from Map

Browser: Chrome

Advanced Settings ▼

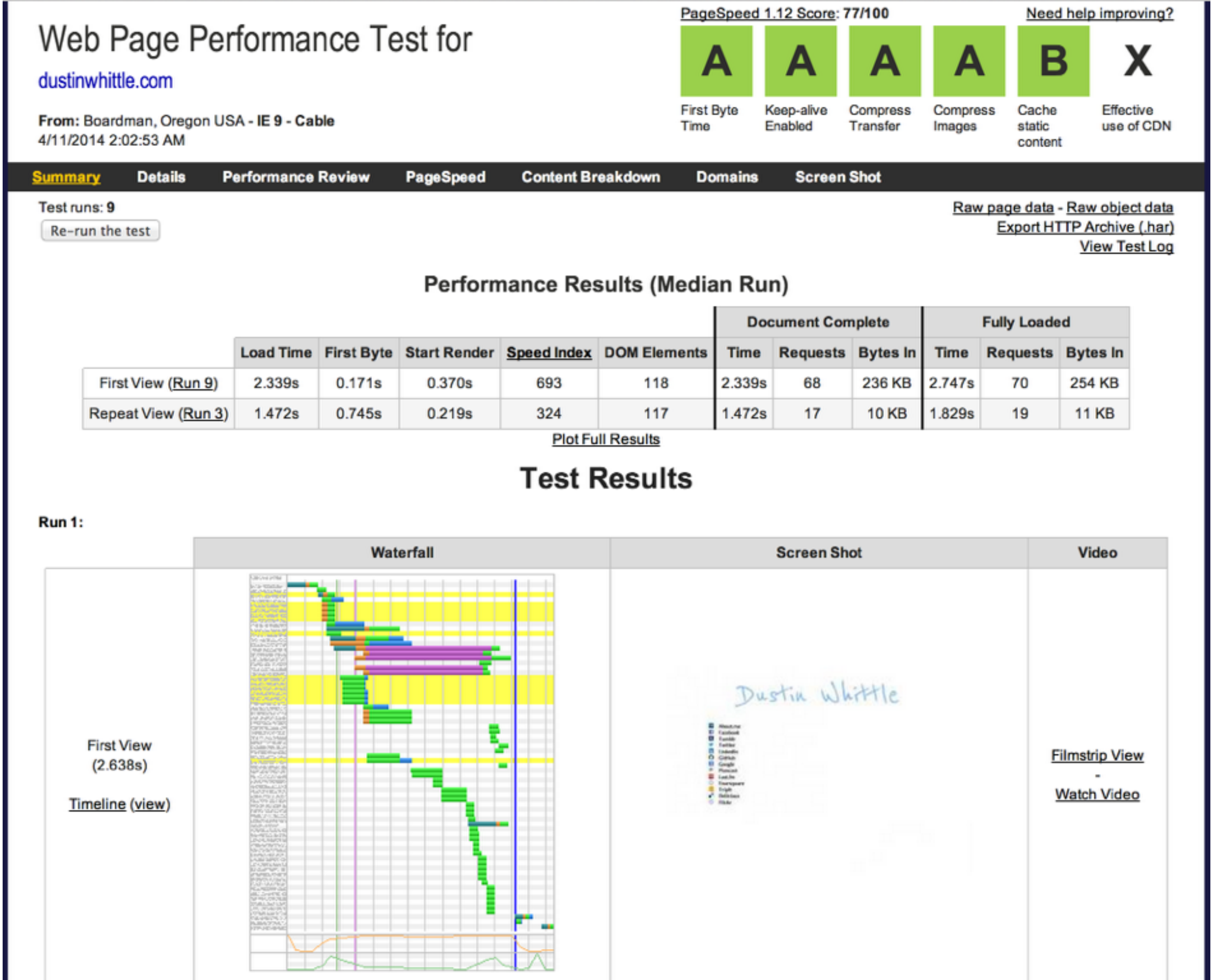
Connection: Cable (5/1 Mbps 28ms RTT)

Number of Tests to Run (Up to 9): 9

Repeat View: ● First View and Repeat View ○ First View Only

Keep Test Private: ☑

Label:



Web Page Performance Test for

dustinwhittle.com

From: Boardman, Oregon USA - **IE 9 - Cable**
4/11/2014 2:02:53 AM

PageSpeed 1.12 Score: 77/100

Need help improving?

A	A	A	A	B	X
First Byte Time	Keep-alive Enabled	Compress Transfer	Compress Images	Cache static content	Effective use of CDN

Test runs: **9**

Raw page data - Raw object data
Export HTTP Archive (.har)
View Test Log

Performance Results (Median Run)

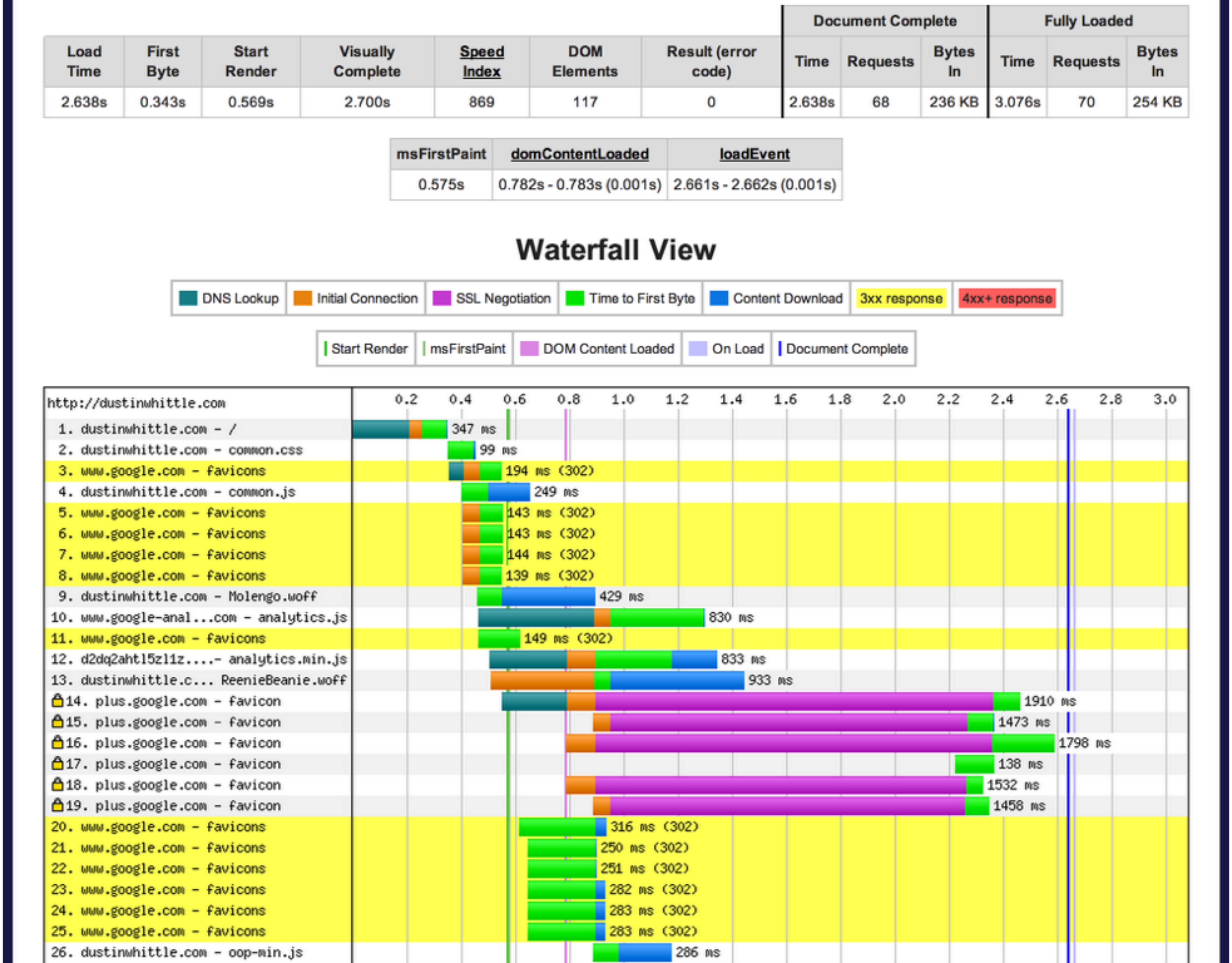
						Document Complete			Fully Loaded		
	Load Time	First Byte	Start Render	Speed Index	DOM Elements	Time	Requests	Bytes In	Time	Requests	Bytes In
First View (Run 9)	2.339s	0.171s	0.370s	693	118	2.339s	68	236 KB	2.747s	70	254 KB
Repeat View (Run 3)	1.472s	0.745s	0.219s	324	117	1.472s	17	10 KB	1.829s	19	11 KB

Plot Full Results

Test Results

Run 1:

	Waterfall	Screen Shot	Video
First View (2.638s) Timeline (view)			Filmstrip View - Watch Video



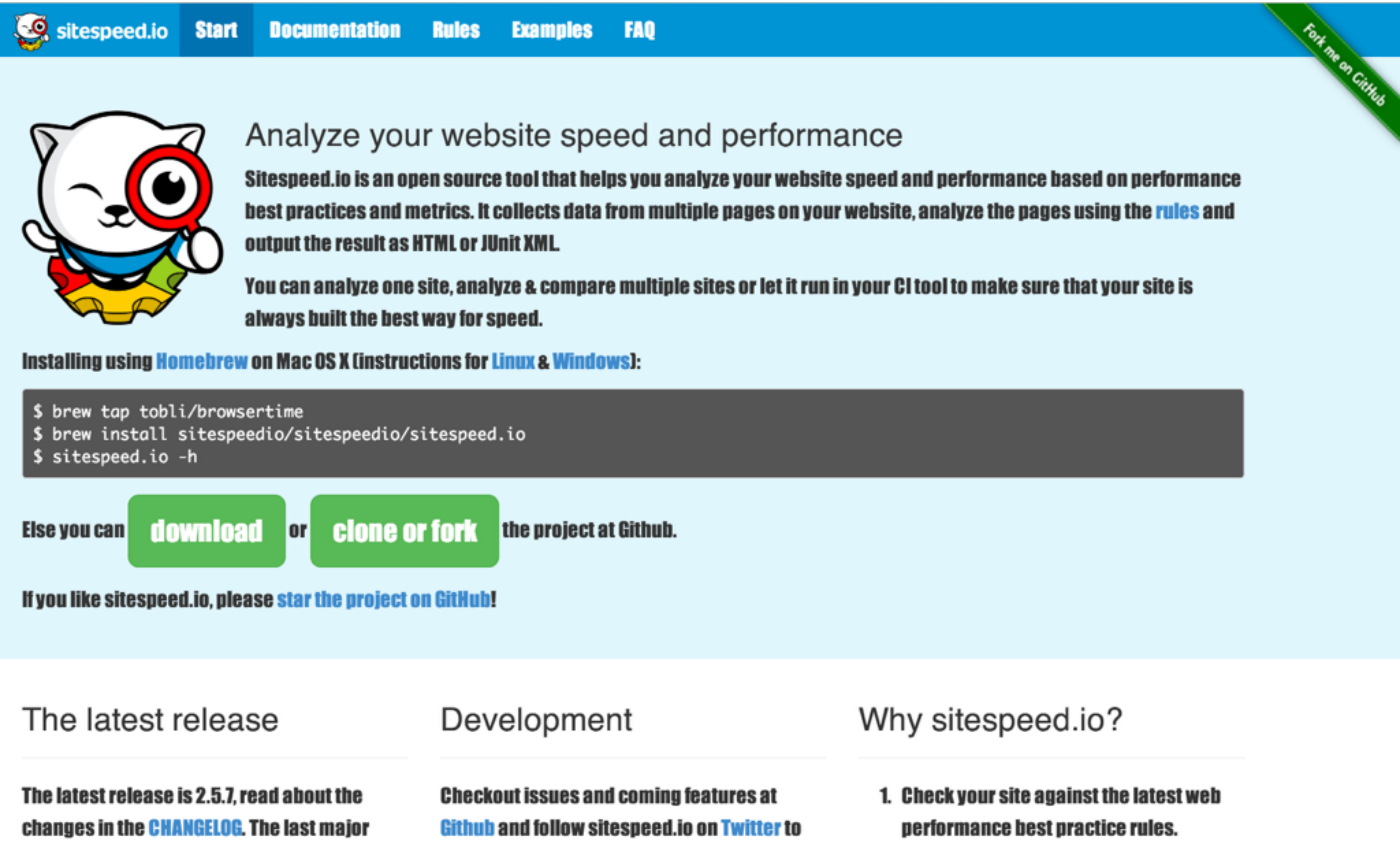
							Document Complete			Fully Loaded		
Load Time	First Byte	Start Render	Visually Complete	Speed Index	DOM Elements	Result (error code)	Time	Requests	Bytes In	Time	Requests	Bytes In
2.638s	0.343s	0.569s	2.700s	869	117	0	2.638s	68	236 KB	3.076s	70	254 KB

msFirstPaint	domContentLoaded	loadEvent
0.575s	0.782s - 0.783s (0.001s)	2.661s - 2.662s (0.001s)

Waterfall View

```
1. dustinwhittle.com - /                    347 ms
2. dustinwhittle.com - common.css           99 ms
3. www.google.com - favicons                194 ms (302)
4. dustinwhittle.com - common.js            249 ms
5. www.google.com - favicons                143 ms (302)
6. www.google.com - favicons                143 ms (302)
7. www.google.com - favicons                144 ms (302)
8. www.google.com - favicons                139 ms (302)
9. dustinwhittle.com - Molengo.woff         429 ms
10. www.google-anal...com - analytics.js    830 ms
11. www.google.com - favicons               149 ms (302)
12. d2dq2aht15z11z....- analytics.min.js    833 ms
13. dustinwhittle.c... ReenieBeanie.woff    933 ms
14. plus.google.com - favicon               1910 ms
15. plus.google.com - favicon               1473 ms
16. plus.google.com - favicon               1798 ms
17. plus.google.com - favicon               138 ms
18. plus.google.com - favicon               1532 ms
19. plus.google.com - favicon               1458 ms
20. www.google.com - favicons               316 ms (302)
21. www.google.com - favicons               250 ms (302)
22. www.google.com - favicons               251 ms (302)
23. www.google.com - favicons               282 ms (302)
24. www.google.com - favicons               283 ms (302)
25. www.google.com - favicons               283 ms (302)
26. dustinwhittle.com - oop-min.js          286 ms
```

SiteSpeed.io



Analyze your website speed and performance

Sitespeed.io is an open source tool that helps you analyze your website speed and performance based on performance best practices and metrics. It collects data from multiple pages on your website, analyze the pages using the rules and output the result as HTML or JUnit XML.

You can analyze one site, analyze & compare multiple sites or let it run in your CI tool to make sure that your site is always built the best way for speed.

Installing using Homebrew on Mac OS X (instructions for Linux & Windows):

```
$ brew tap tobli/browsertime
$ brew install sitespeedio/sitespeedio/sitespeed.io
$ sitespeed.io -h
```

Else you can download or clone or fork the project at Github.

If you like sitespeed.io, please star the project on GitHub!

The latest release

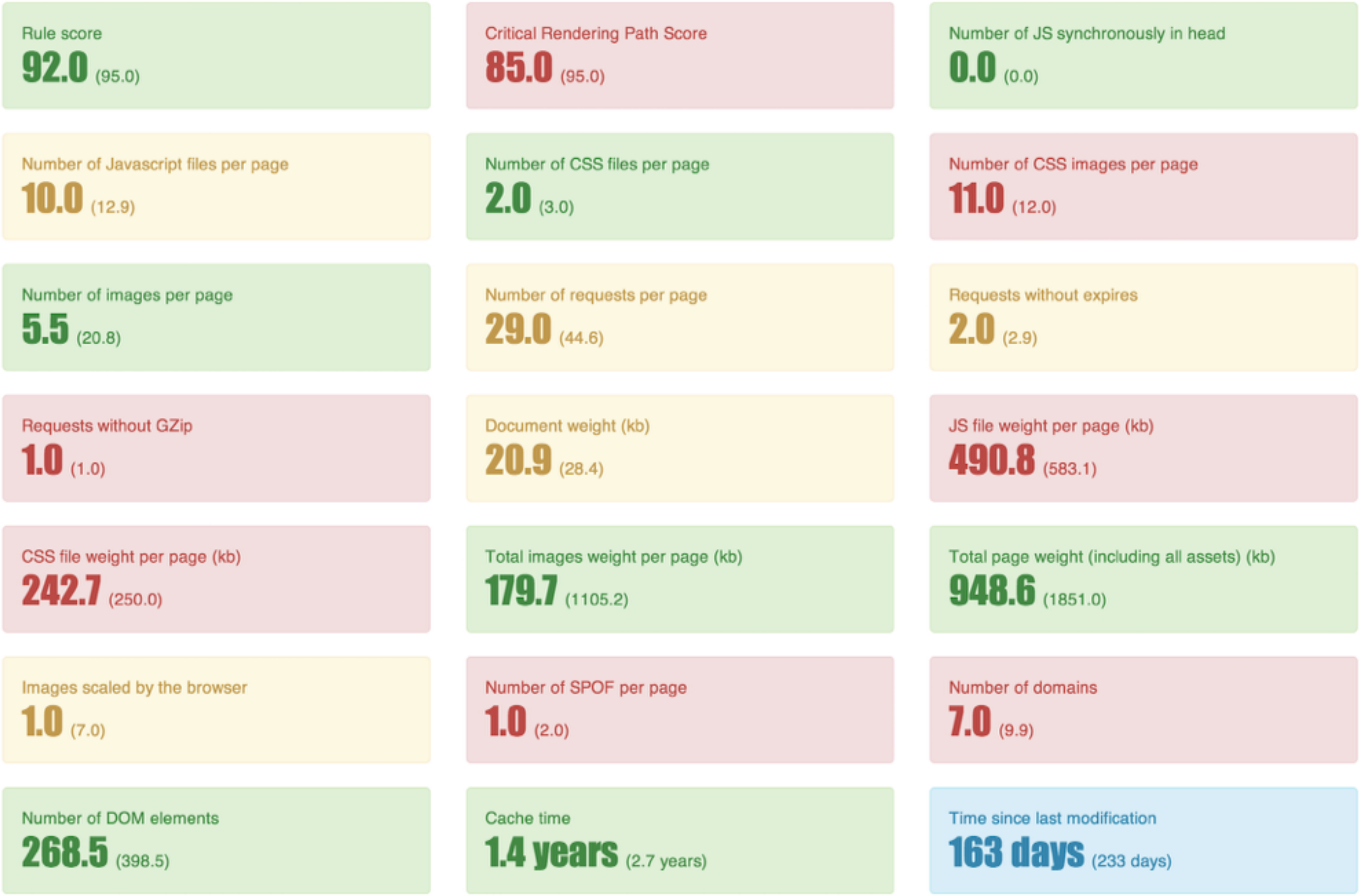
The latest release is 2.5.7, read about the changes in the CHANGELOG. The last major

Development

Checkout issues and coming features at Github and follow sitespeed.io on Twitter to

Why sitespeed.io?

1. Check your site against the latest web performance best practice rules.



Rule score
92.0 (95.0)

Critical Rendering Path Score
85.0 (95.0)

Number of JS synchronously in head
0.0 (0.0)

Number of Javascript files per page
10.0 (12.9)

Number of CSS files per page
2.0 (3.0)

Number of CSS images per page
11.0 (12.0)

Number of images per page
5.5 (20.8)

Number of requests per page
29.0 (44.6)

Requests without expires
2.0 (2.9)

Requests without GZip
1.0 (1.0)

Document weight (kb)
20.9 (28.4)

JS file weight per page (kb)
490.8 (583.1)

CSS file weight per page (kb)
242.7 (250.0)

Total images weight per page (kb)
179.7 (1105.2)

Total page weight (including all assets) (kb)
948.6 (1851.0)

Images scaled by the browser
1.0 (7.0)

Number of SPOF per page
1.0 (2.0)

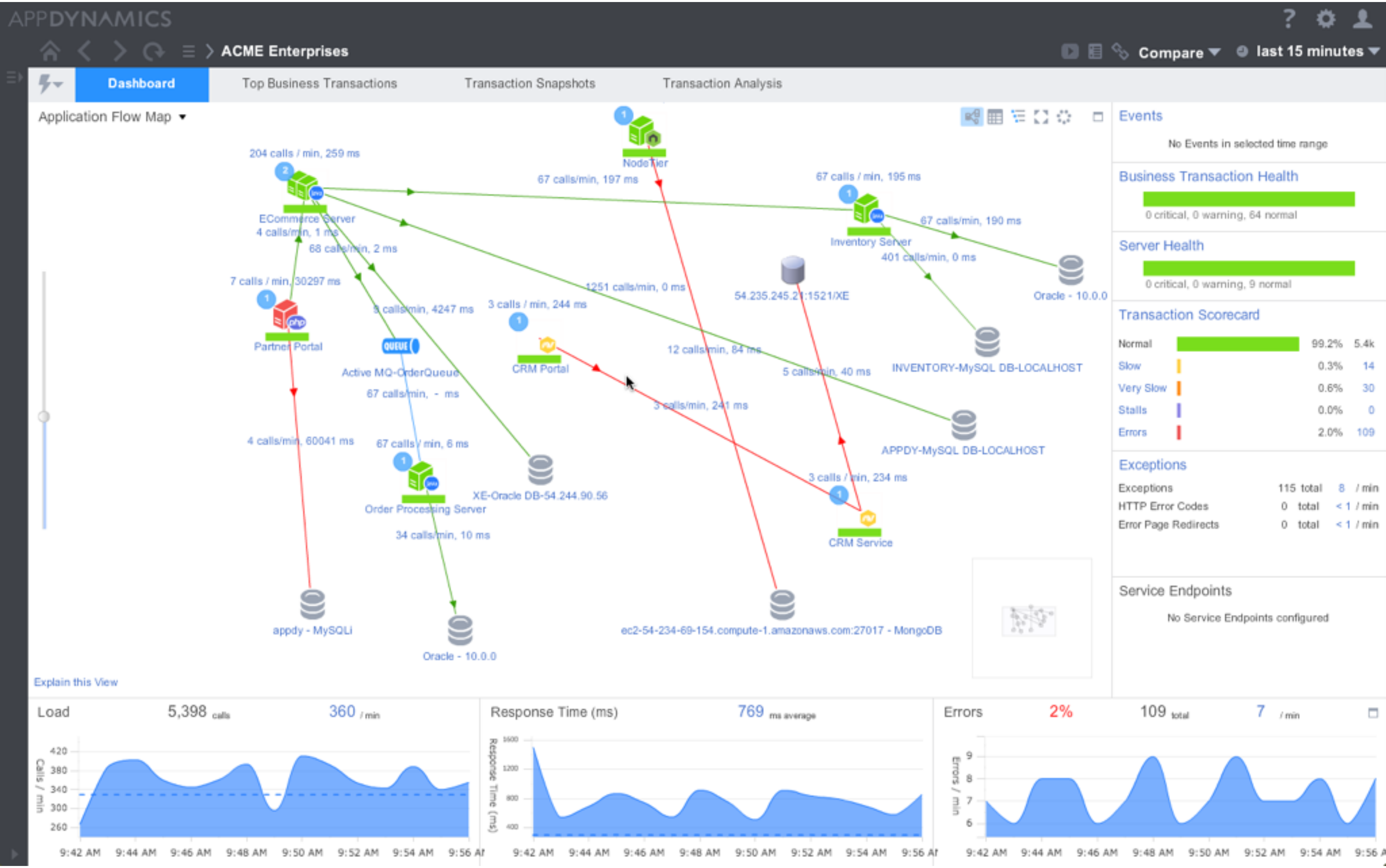
Number of domains
7.0 (9.9)

Number of DOM elements
268.5 (398.5)

Cache time
1.4 years (2.7 years)

Time since last modification
163 days (233 days)

APPDYNAMICS



Events

No Events in selected time range

Business Transaction Health

0 critical, 0 warning, 64 normal

Server Health

0 critical, 0 warning, 9 normal

Transaction Scorecard

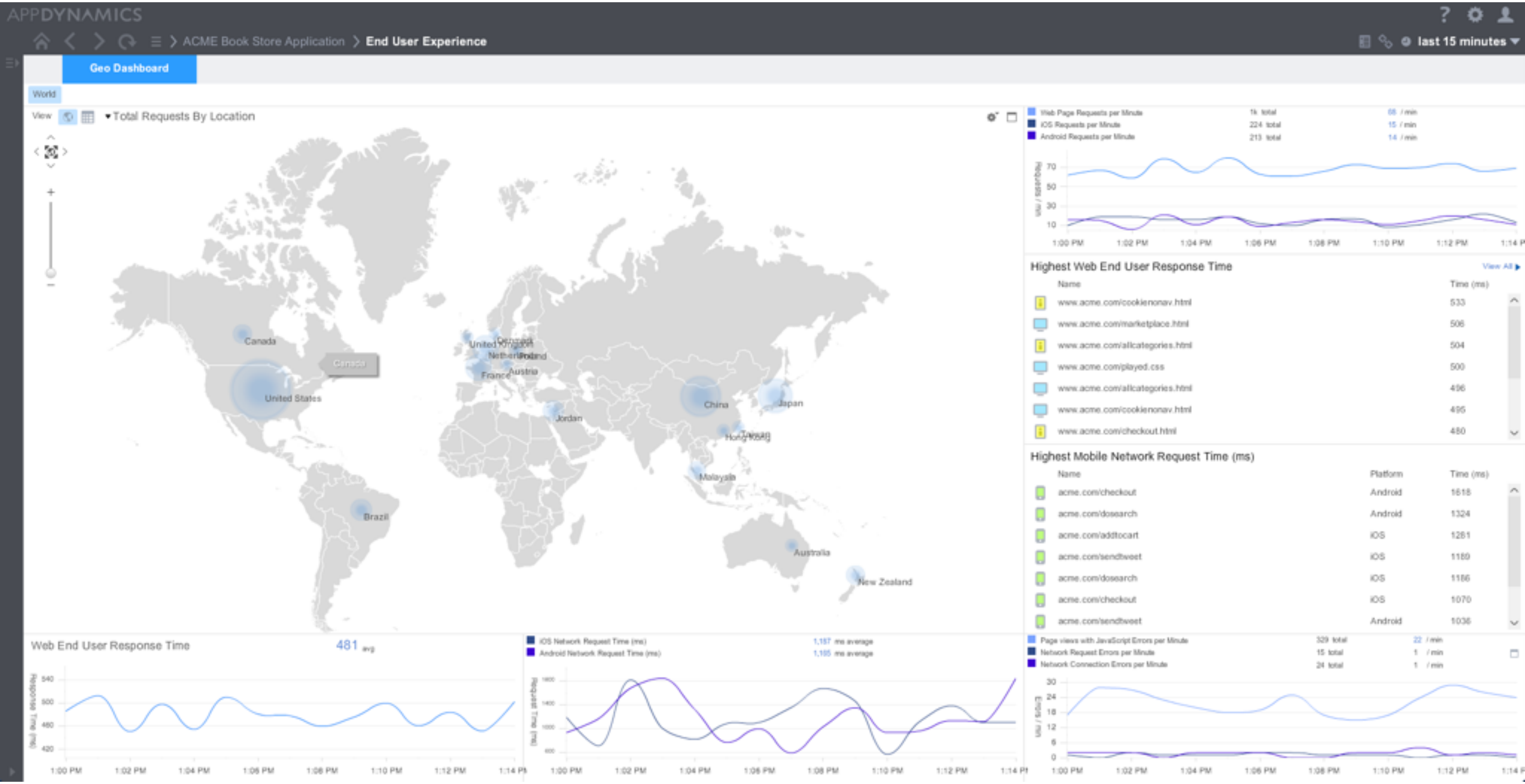
Normal	🟩	99.2%	5.4k
Slow		0.3%	14
Very Slow		0.6%	30
Stalls		0.0%	0
Errors		2.0%	109

Exceptions

Exceptions	115	total	8	/ min
HTTP Error Codes	0	total	< 1	/ min
Error Page Redirects	0	total	< 1	/ min

Service Endpoints

No Service Endpoints configured



🟦 Web Page Requests per Minute	1k total	68 / min
🟦 iOS Requests per Minute	224 total	15 / min
🟪 Android Requests per Minute	213 total	14 / min

Highest Web End User Response Time

View All ▶

	Name	Time (ms)
	www.acme.com/cookienonav.html	533
	www.acme.com/marketplace.html	506
	www.acme.com/allcategories.html	504
	www.acme.com/played.css	500
	www.acme.com/allcategories.html	496
	www.acme.com/cookienonav.html	495
	www.acme.com/checkout.html	480

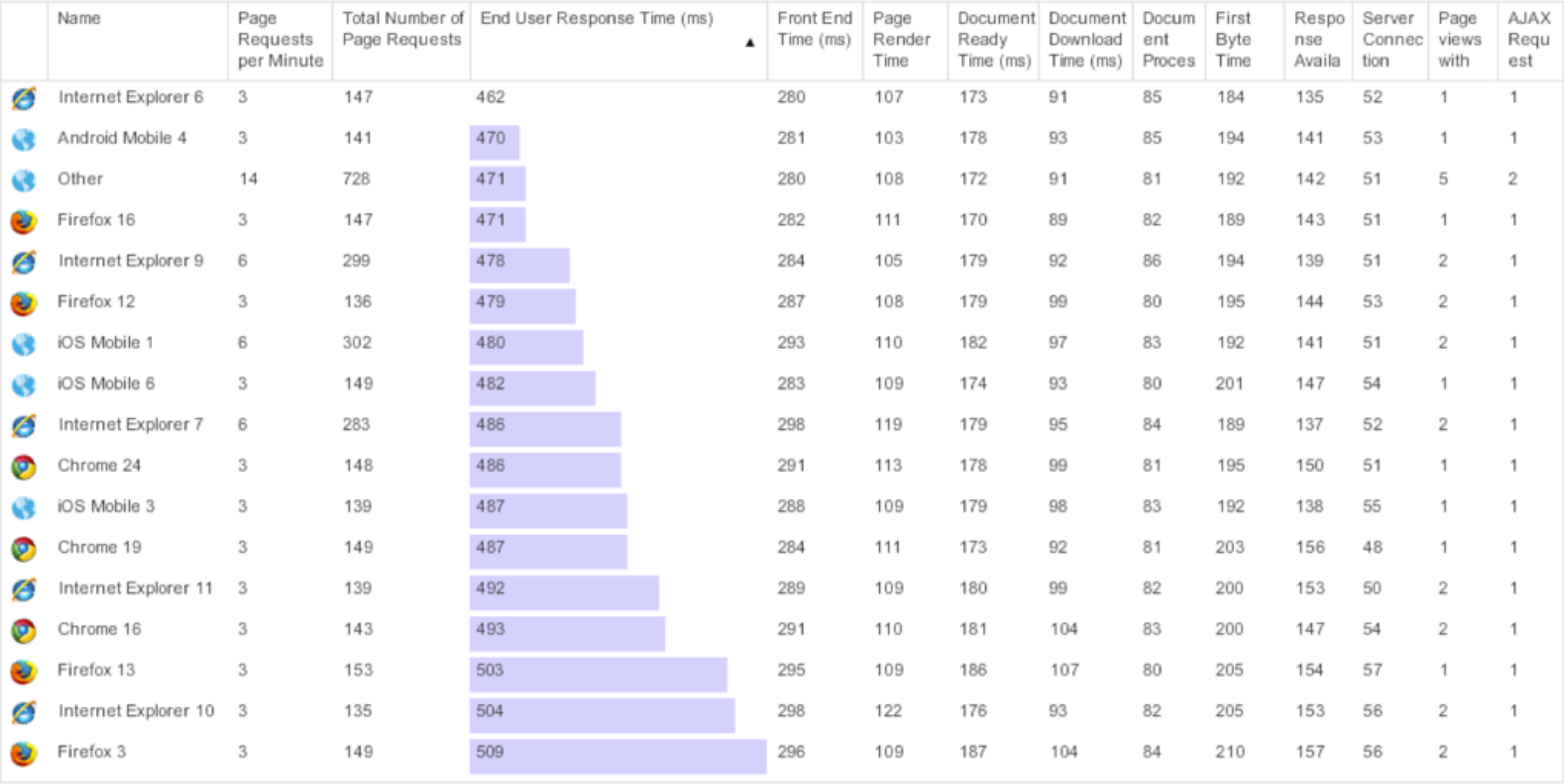
Highest Mobile Network Request Time (ms)

	Name	Platform	Time (ms)
	acme.com/checkout	Android	1618
	acme.com/dosearch	Android	1324
	acme.com/addtocart	iOS	1281
	acme.com/sendtweet	iOS	1189
	acme.com/dosearch	iOS	1186
	acme.com/checkout	iOS	1070
	acme.com/sendtweet	Android	1036

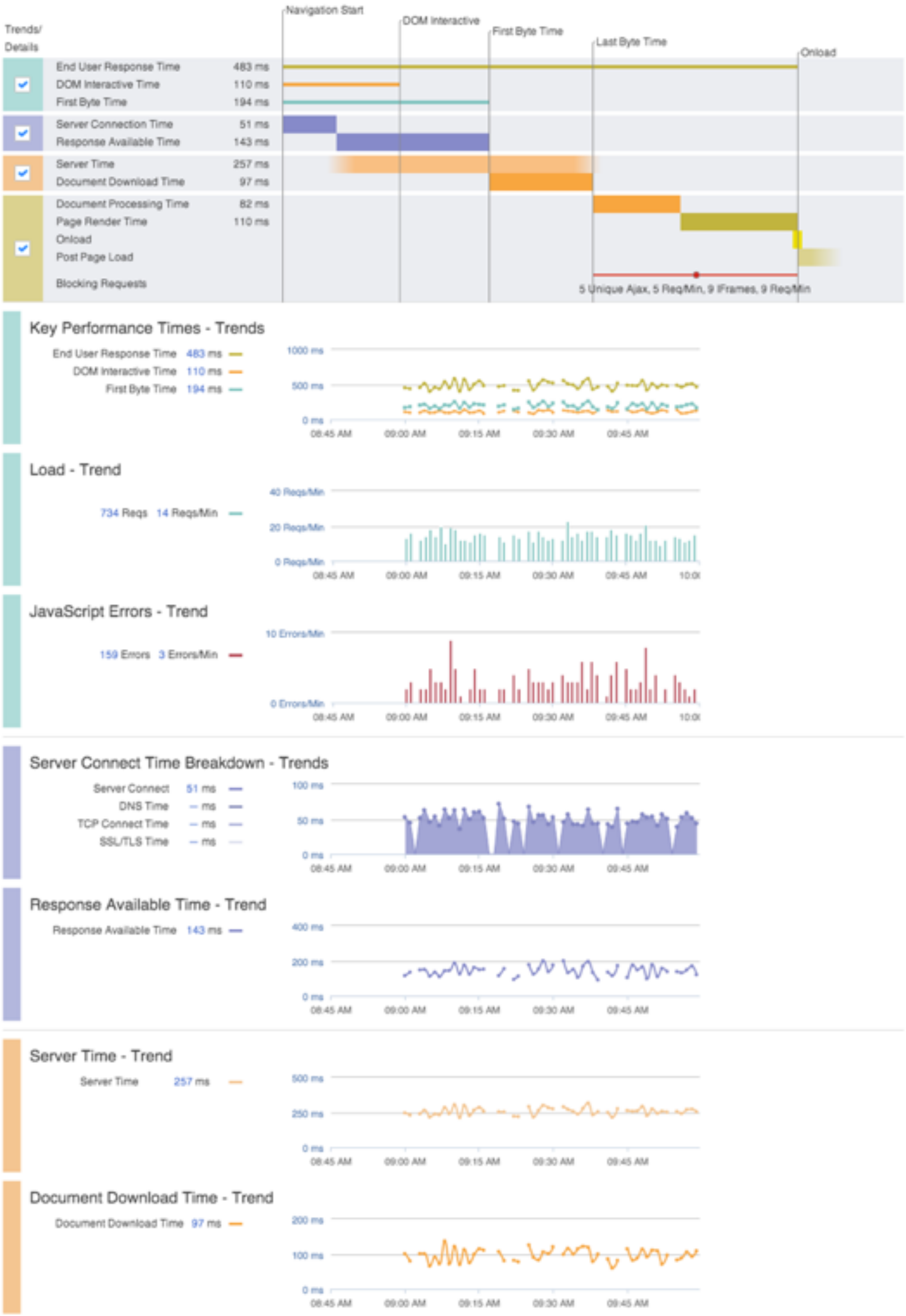
Web End User Response Time 481 avg

🟦 iOS Network Request Time (ms)	1,157	ms average
🟪 Android Network Request Time (ms)	1,105	ms average

🟦 Page views with JavaScript Errors per Minute	329 total	22 / min
🟦 Network Request Errors per Minute	15 total	1 / min
🟪 Network Connection Errors per Minute	24 total	1 / min



	Name	Page Requests per Minute	Total Number of Page Requests	End User Response Time (ms) ▲	Front End Time (ms)	Page Render Time	Document Ready Time (ms)	Document Download Time (ms)	Docum ent Proces	First Byte Time	Respo nse Availa	Server Connec tion	Page views with	AJAX Requ est
	Internet Explorer 6	3	147	462	280	107	173	91	85	184	135	52	1	1
	Android Mobile 4	3	141	470	281	103	178	93	85	194	141	53	1	1
	Other	14	728	471	280	108	172	91	81	192	142	51	5	2
	Firefox 16	3	147	471	282	111	170	89	82	189	143	51	1	1
	Internet Explorer 9	6	299	478	284	105	179	92	86	194	139	51	2	1
	Firefox 12	3	136	479	287	108	179	99	80	195	144	53	2	1
	iOS Mobile 1	6	302	480	293	110	182	97	83	192	141	51	2	1
	iOS Mobile 6	3	149	482	283	109	174	93	80	201	147	54	1	1
	Internet Explorer 7	6	283	486	298	119	179	95	84	189	137	52	2	1
	Chrome 24	3	148	486	291	113	178	99	81	195	150	51	1	1
	iOS Mobile 3	3	139	487	288	109	179	98	83	192	138	55	1	1
	Chrome 19	3	149	487	284	111	173	92	81	203	156	48	1	1
	Internet Explorer 11	3	139	492	289	109	180	99	82	200	153	50	2	1
	Chrome 16	3	143	493	291	110	181	104	83	200	147	54	2	1
	Firefox 13	3	153	503	295	109	186	107	80	205	154	57	1	1
	Internet Explorer 10	3	135	504	298	122	176	93	82	205	153	56	2	1
	Firefox 3	3	149	509	296	109	187	104	84	210	157	56	2	1



Trends/
Details

☑	End User Response Time	483 ms
	DOM Interactive Time	110 ms
	First Byte Time	194 ms
☑	Server Connection Time	51 ms
	Response Available Time	143 ms
☑	Server Time	257 ms
	Document Download Time	97 ms
☑	Document Processing Time	82 ms
	Page Render Time	110 ms
	Onload	
	Post Page Load	
	Blocking Requests	

5 Unique Ajax, 5 Req/Min, 9 IFrames, 9 Req/Min

Key Performance Times - Trends

End User Response Time 483 ms
DOM Interactive Time 110 ms
First Byte Time 194 ms

Load - Trend

734 Reqs 14 Reqs/Min

JavaScript Errors - Trend

159 Errors 3 Errors/Min

Server Connect Time Breakdown - Trends

Server Connect 51 ms
DNS Time – ms
TCP Connect Time – ms
SSL/TLS Time – ms

Response Available Time - Trend

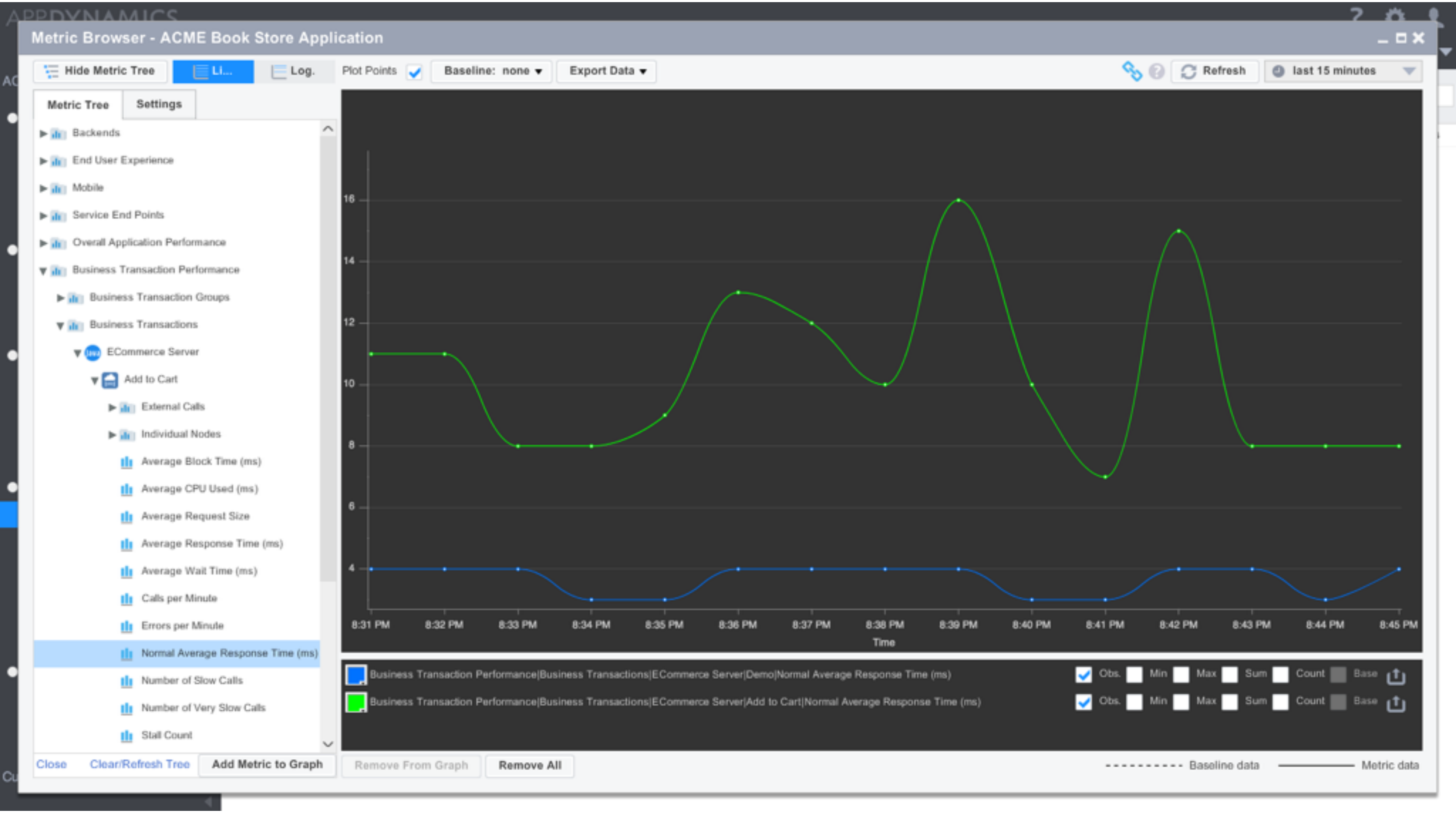
Response Available Time 143 ms

Server Time - Trend

Server Time 257 ms

Document Download Time - Trend

Document Download Time 97 ms



Metric Browser - ACME Book Store Application

Plot Points ☑

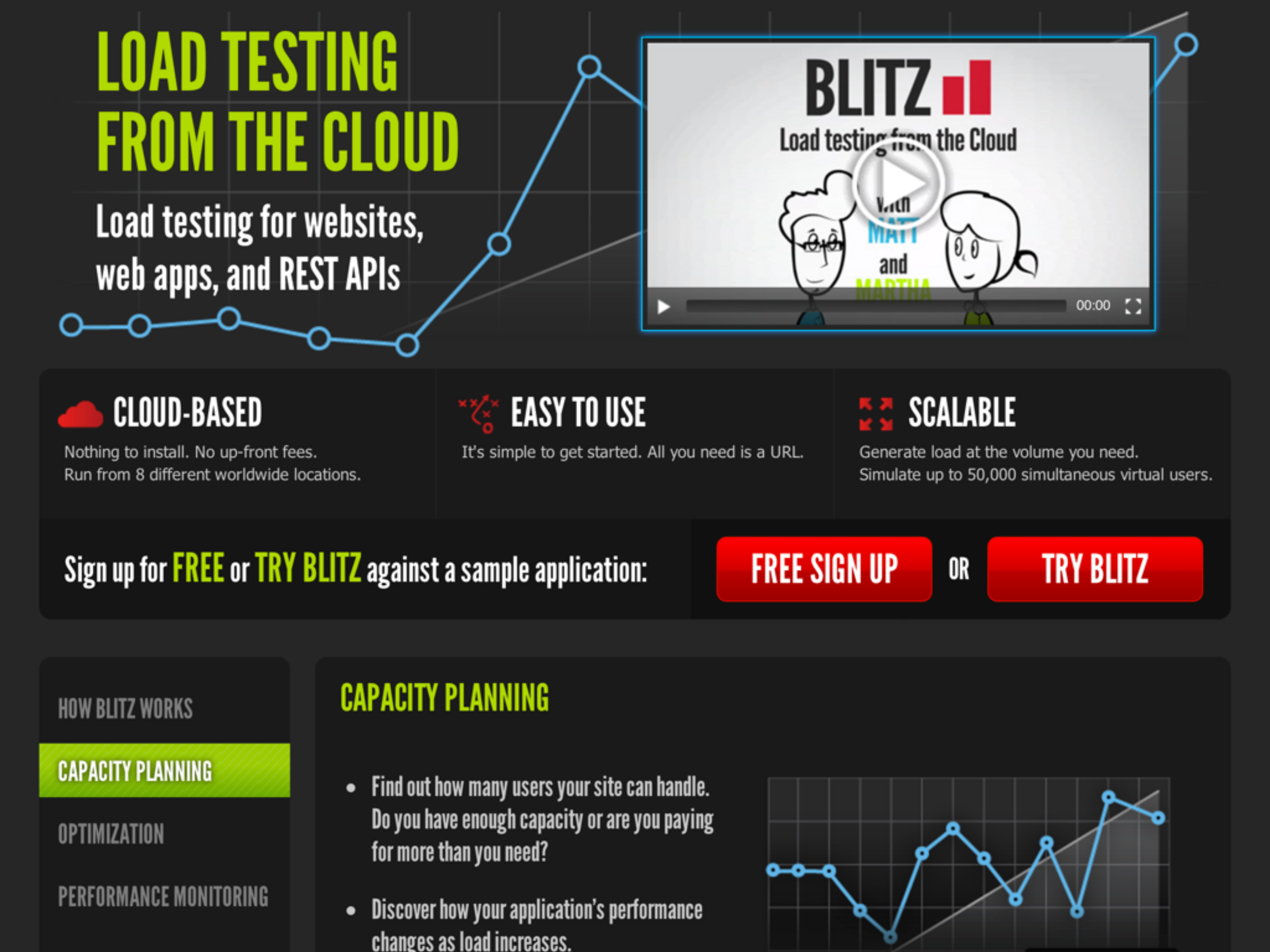
Baseline: none ▾

last 15 minutes

🟦 Business Transaction Performance|Business Transactions|ECommerce Server|Demo|Normal Average Response Time (ms) ☑ Obs. □ Min □ Max □ Sum □ Count ▨ Base

🟩 Business Transaction Performance|Business Transactions|ECommerce Server|Add to Cart|Normal Average Response Time (ms) ☑ Obs. □ Min □ Max □ Sum □ Count ▨ Base

Load testing services from the cloud



LOAD TESTING FROM THE CLOUD

Load testing for websites, web apps, and REST APIs

CLOUD-BASED

Nothing to install. No up-front fees.
Run from 8 different worldwide locations.

EASY TO USE

It's simple to get started. All you need is a URL.

SCALABLE

Generate load at the volume you need.
Simulate up to 50,000 simultaneous virtual users.

Sign up for FREE or TRY BLITZ against a sample application:

FREE SIGN UP OR TRY BLITZ

HOW BLITZ WORKS

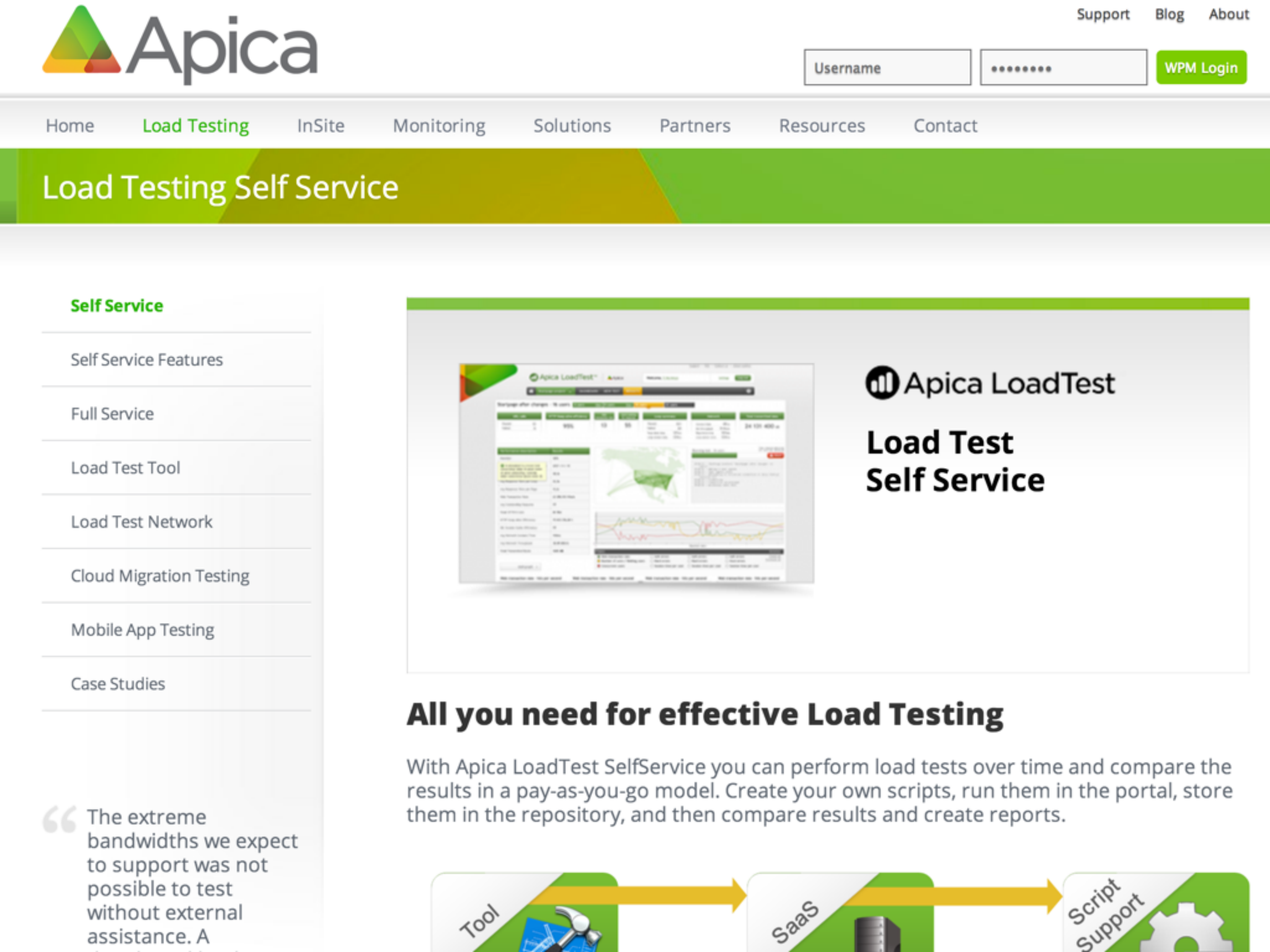
CAPACITY PLANNING

OPTIMIZATION

PERFORMANCE MONITORING

CAPACITY PLANNING

- Find out how many users your site can handle. Do you have enough capacity or are you paying for more than you need?
- Discover how your application's performance changes as load increases.



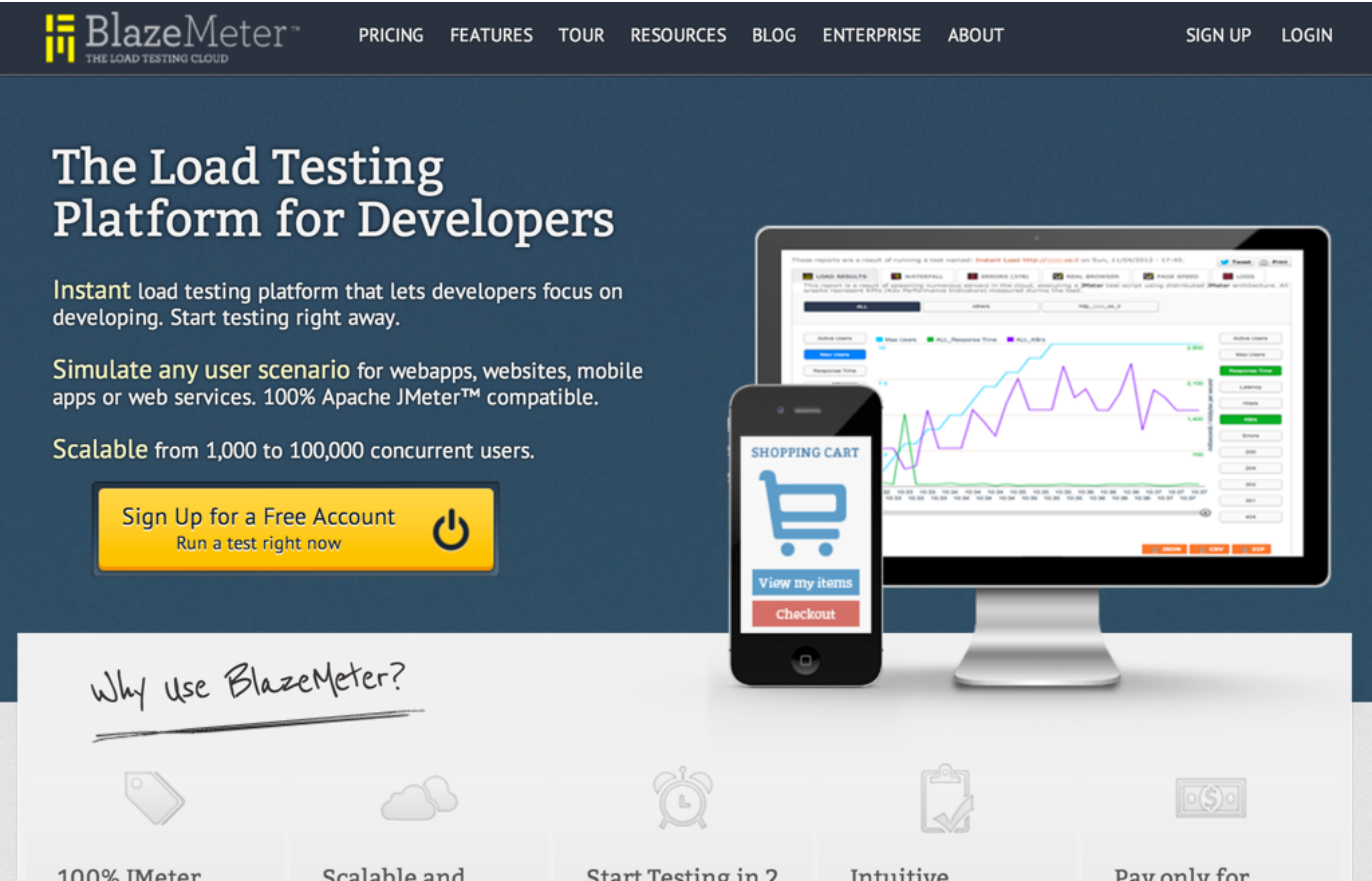
Load Testing Self Service

Load Test
Self Service

All you need for effective Load Testing

With Apica LoadTest SelfService you can perform load tests over time and compare the results in a pay-as-you-go model. Create your own scripts, run them in the portal, store them in the repository, and then compare results and create reports.

> “ The extreme bandwidths we expect to support was not possible to test without external assistance. A



The Load Testing Platform for Developers

Instant load testing platform that lets developers focus on developing. Start testing right away.

Simulate any user scenario for webapps, websites, mobile apps or web services. 100% Apache JMeter™ compatible.

Scalable from 1,000 to 100,000 concurrent users.

Why use BlazeMeter?

100% JMeter

Scalable and

Start Testing in 2

Intuitive

Pay only for

Test for failures

- NetFlix Simian Army + Chaos Monkey

- What happens if you lose a caching layer?
- What happens if dependencies slow down?

Best Practices

- Capacity plan and load test the server-side
- Optimize and performance test the client-side
- Understand your starting point
- Instrument everything
- Measure the difference of every change
- Automate performance testing in your build and deployment process
- Understand how failures impact performance

Integrate automated performance testing into continuous integration for server-side and client-side

Understand the performance implications of every deployment and package upgrade

Monitor end user experience from end to end in production

Questions?

Find these slides on SpeakerDeck

https://speakerdeck.com/dustinwhittle