NGINX

Tuning NGINX for high performance

Nick Shadrin

nick@nginx.com

All links on one page

shadrin.org/talks/

Twitter:

@shadrin
@nginx
@nginxorg

About me

- Nick Shadrin
- Product Manager at NGINX
- Based in San Francisco
- 16 years experience with web tech
- nick@nginx.com || @shadrin || https://shadrin.org

Agenda

- A basic NGINX configuration
- NGINX performance optimizations:
- Operating system-level optimizations
- Networking-level optimizations
- NGINX core optimizations
- Conclusions and questions


NGINX

“… when I started NGINX, I focused on a very specific problem – how to handle more customers per single server.”

- Igor Sysoev, NGINX creator & our founder

About NGINX, Inc.

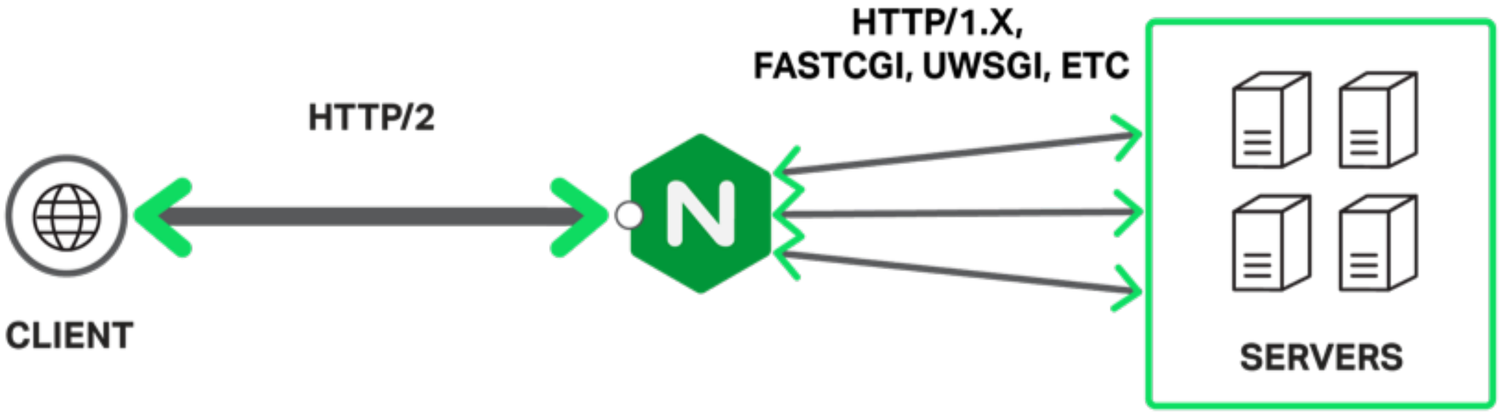
- Company founded in 2011, NGINX Plus started in 2013
- VC-backed by enterprise software industry leaders
- HQ in San Francisco, offices in US and Europe
- 800+ commercial customers
- 120+ employees

Web Scale

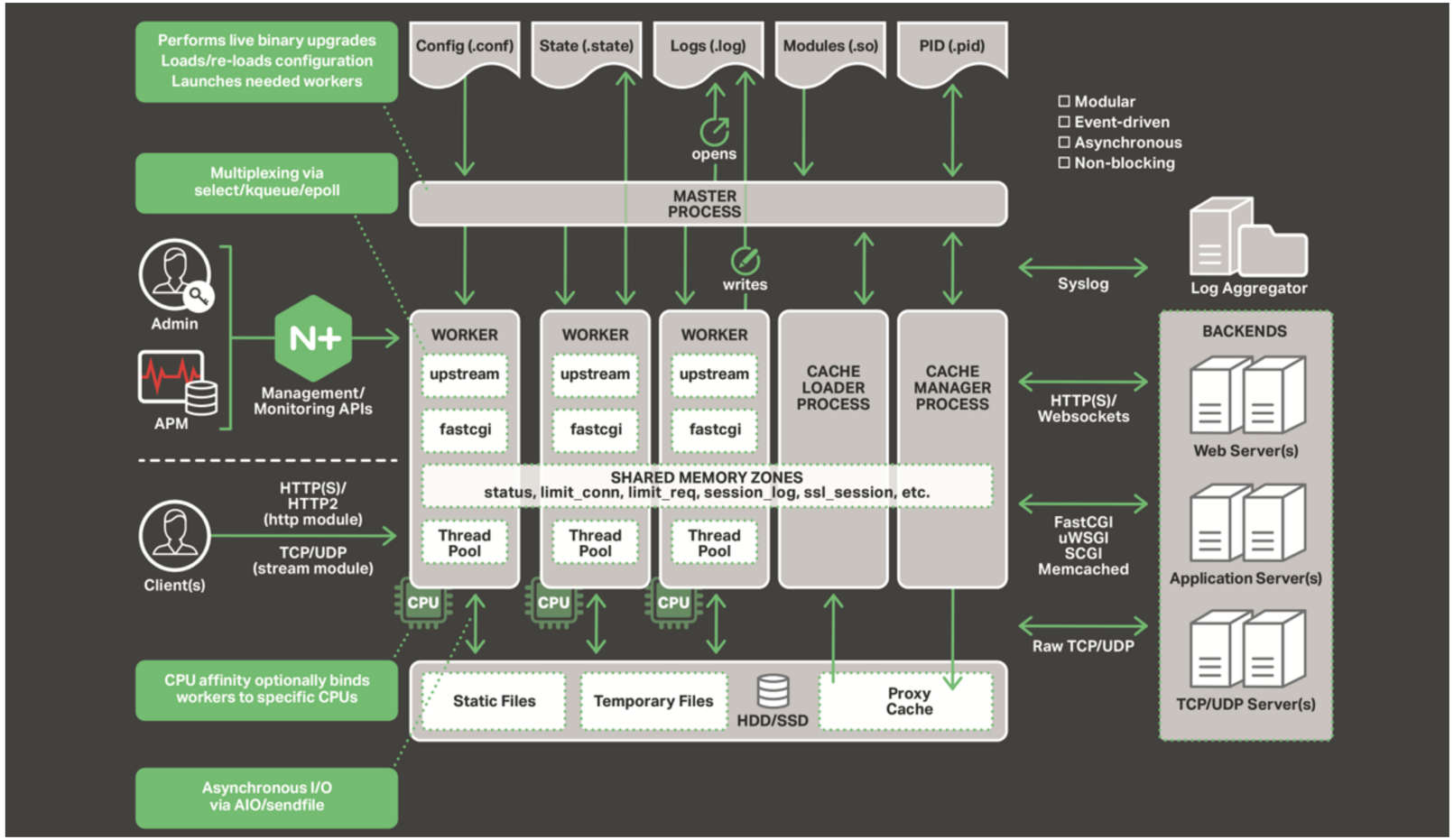
Architecture approach

- Design for scaling
- Segment microservices out
- Use caching and microcaching

Basic NGINX placement

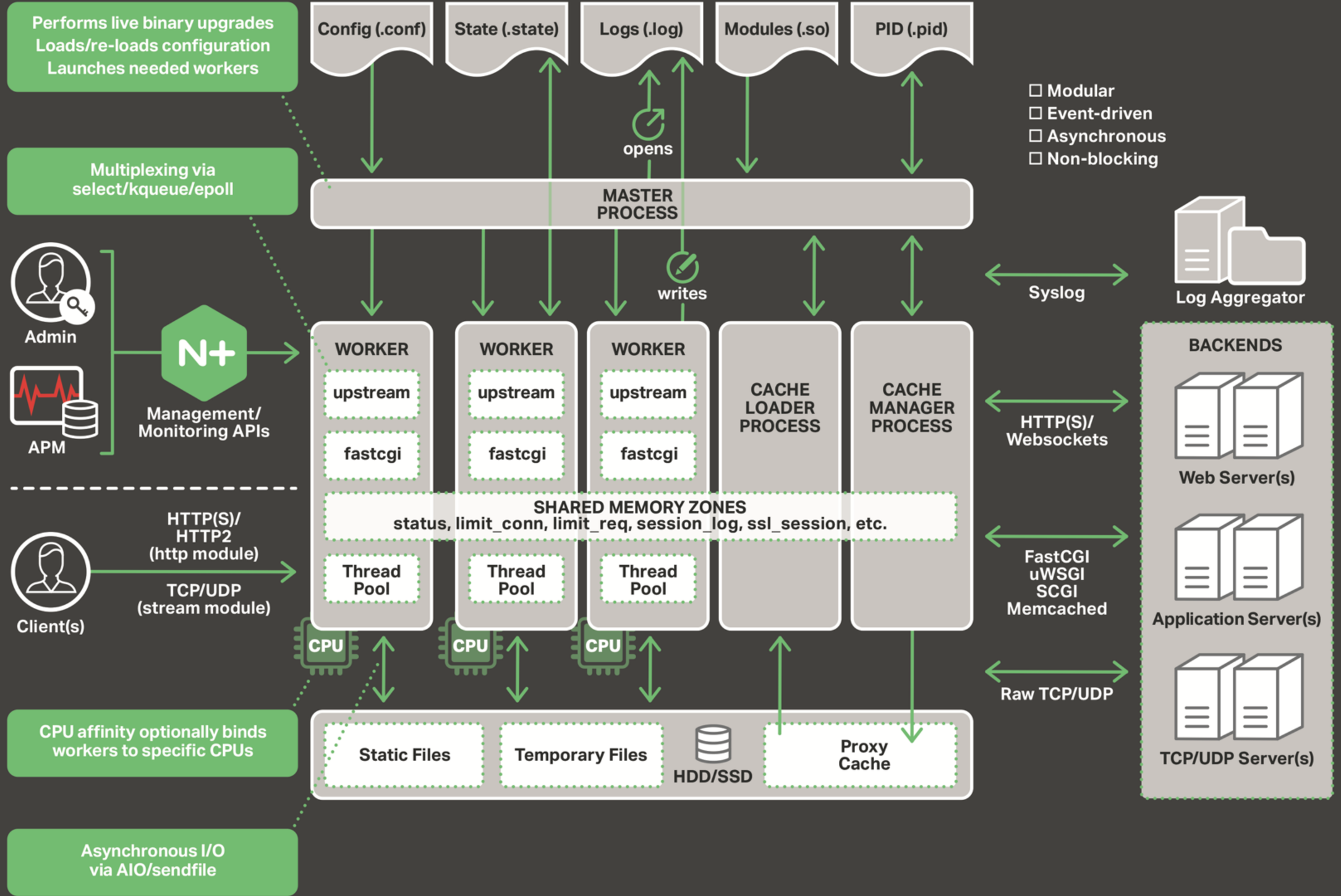


Inside NGINX



https://www.nginx.com/blog/inside-nginx-how-we-designed-for-performance-scale/

http://www.aosabook.org/en/nginx.html


Performs live binary upgrades
Loads/re-loads configuration
Launches needed workers
Config (.conf)
State (.state)
Logs (.log)
Modules (.so)
PID (.pid)
opens
Modular
Event-driven
Asynchronous
Non-blocking
Multiplexing via
select/kqueue/epoll
MASTER
PROCESS
writes
Syslog
Log Aggregator
Admin
APM
N+
Management/
Monitoring APIs
WORKER
WORKER
WORKER
upstream
fastcgi
CACHE
LOADER
PROCESS
CACHE
MANAGER
PROCESS
BACKENDS
HTTP(S)/
Websockets
Web Server(s)
SHARED MEMORY ZONES
status, limit_conn, limit_req, session_log, ssl_session, etc.
Thread
Pool
HTTP(S)/
HTTP2
(http module)
TCP/UDP
(stream module)
Client(s)
FastCGI
uWSGI
SCGI
Memcached
Application Server(s)
CPU
Raw TCP/UDP
TCP/UDP Server(s)
CPU affinity optionally binds
workers to specific CPUs
Static Files
Temporary Files
HDD/SSD
Proxy
Cache
Asynchronous I/O
via AIO/sendfile

OS tuning

- **`net.core.somaxconn`**
- **`net.core.netdev_max_backlog`**
- **`net.ipv4.ip_local_port_range`**
- **`sys.fs.file_max`**
- **`/etc/security/limits.conf`**, nofile setting

See https://www.nginx.com/blog/tuning-nginx/

Overcoming ephemeral port exhaustion

- Increase local port range
- Split traffic across multiple IPs
- NGINX since 1.11.2 uses IP_BIND_ADDRESS_NO_PORT socket option when available

https://www.nginx.com/blog/overcoming-ephemeral-port-exhaustion-nginx-plus/

Minimal NGINX configuration

```
events {}

http {

  server {
    listen 80;
    location / {
      proxy_pass http://backend;
    }
  }

  upstream backend {
    server backend1.example.com:8080;
    server backend2.example.com:8080;
  }
}
```

NGINX Performance features

NGINX Core features

- Use correct number of **`worker_processes`**
  - auto
  - # of available CPU cores
- Increase **`worker_connections`**
- Increase **`worker_rlimit_nofile`**

NGINX Core Features (cont'd)

- Turn off accept_mutex:
  `accept_mutex off;`
- Turn on Sendfile
  `sendfile on;`
- Use thread pools if I/O needs offloading:
  `aio threads;`

https://www.nginx.com/blog/thread-pools-boost-performance-9x/

Changes with nginx 1.11.3 26 Jul 2016

*) Change: now the "accept_mutex" directive is turned off by default.

[skip]

http://nginx.org/en/CHANGES

NGINX Core Features (cont'd)

- Turn off accept_mutex:
  `accept_mutex off;`
- Turn on Sendfile
  `sendfile on;`
- Use thread pools if I/O needs offloading:
  `aio threads;`

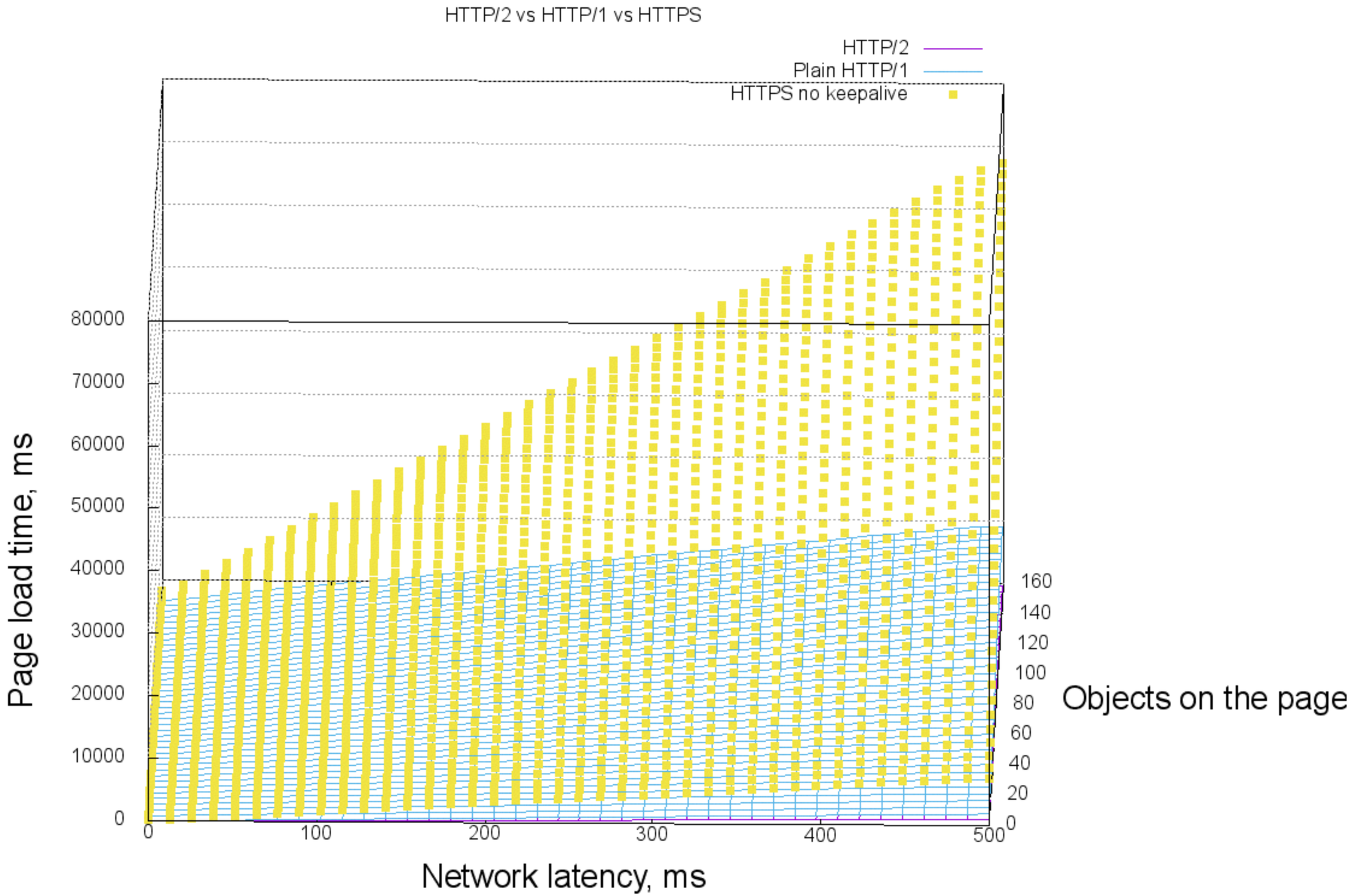
https://www.nginx.com/blog/thread-pools-boost-performance-9x/

HTTP Keep alive

- Keepalive connections allow to reuse the same TCP connection for multiple HTTP requests.
- For HTTP/1.1, no need to define anything, it's enabled by default on the frontend.
- Keepalives provide major performance benefit when used over SSL/TLS connections.

HTTP Keepalive: benchmark

- HTTPS with NO keepalive (worst setup)
- Plain HTTP
- HTTP/2 with SSL


HTTP/2 vs HTTP/1 vs HTTPS
HTTP/2
Plain HTTP/1
HTTPS no keepalive
Page load time, ms
80000
70000
60000
50000
40000
30000
20000
10000
0
0
100
200
300
400
500
Network latency, ms
160
140
120
100
80
60
40
20
0
Objects on the page

HTTP Keepalive

- Keepalive on the Frontend:

```
keepalive_requests 100;
keepalive_timeout 75s;
```

HTTP Keepalive on the backend

Keepalive on the Backend:

```
server {
  location / {
    proxy_pass http://backend;
    proxy_http_version 1.1;
    proxy_set_header Connection "";
  }
}
...
upstream backend {
  server example.com;
  keepalive 32;
}
```

HTTP Caching

- Microcaching with NGINX: https://www.nginx.com/blog/benefits-of-microcaching-nginx/

- Cache placement strategies: https://www.nginx.com/blog/cache-placement-strategies-nginx-plus/

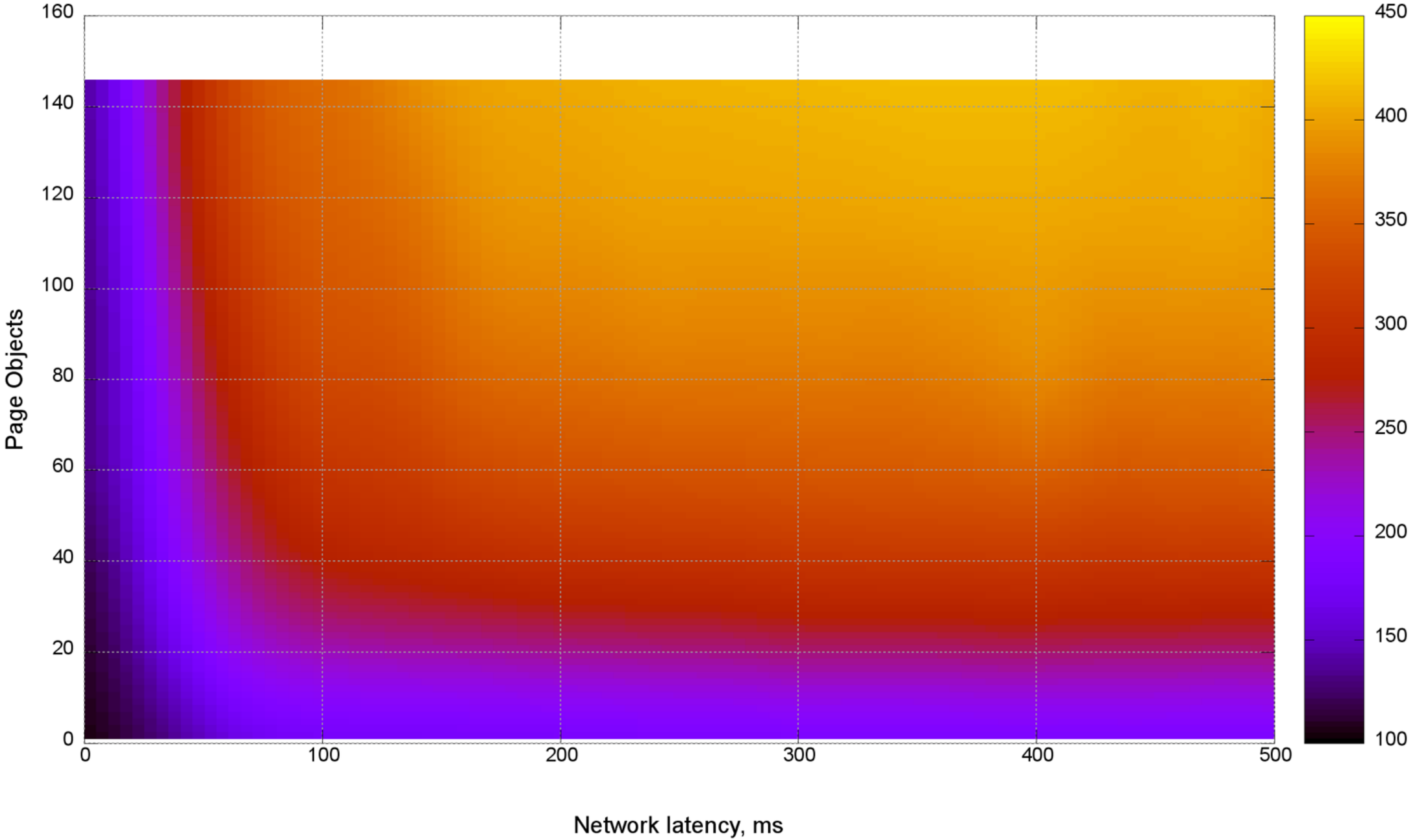
HTTP/2

- Introduced in 2015 as a standard
- Based on Google's SPDY
- Includes major changes compared to HTTP/1:
  - Binary headers with HPACK
  - Multiple streams
  - Prioritization
  - Server Push

HTTP/2 benchmark

- NGINX 1.10.0
- Ubuntu 16.04
- Openssl 1.0.2
- Chrome Web browser
- Measuring full page reload

HTTP/2 vs HTTP/1/SSL, percentage performance increase



Some numbers

- 40ms / 50 objects:
  HTTP/1: **510ms**
  HTTP/2: **250ms**

~2 times faster

- 200ms / 100 objects:
  HTTP/1: **4.0s**
  HTTP/2: **1.1s**

~4 times faster

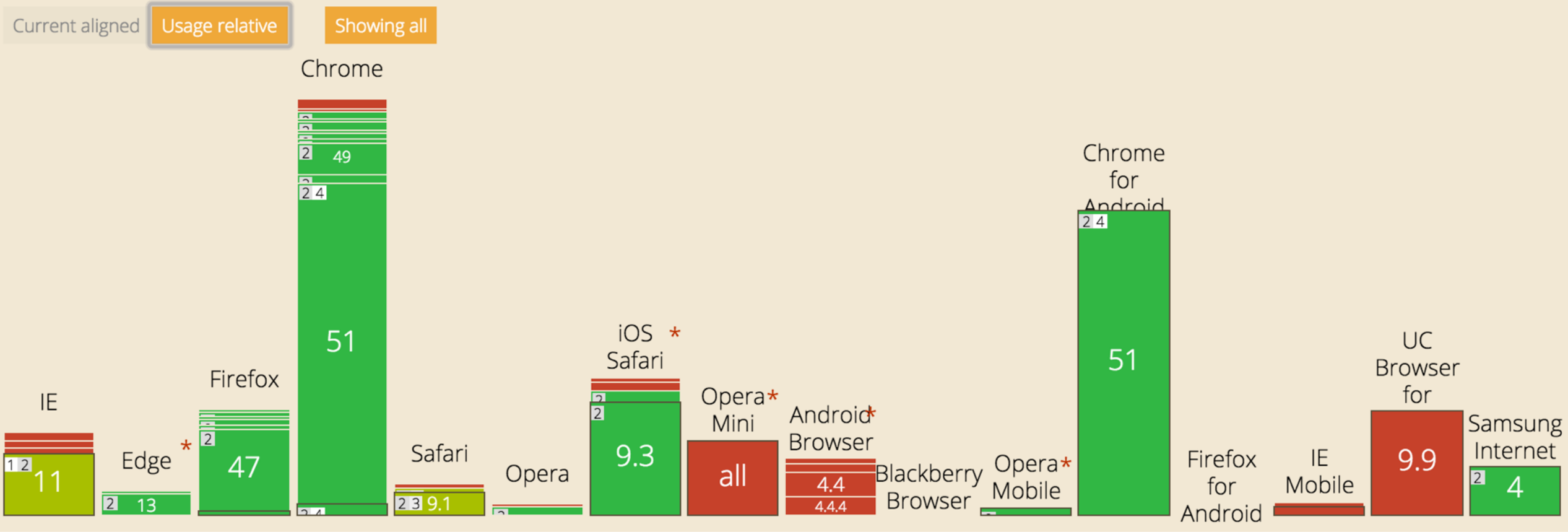
NGINX

@shadrin | nick@nginx.com

HTTP/2 protocol - OTHER

Global 70.13% + 5.95% = 76.08%

Networking protocol for low-latency transport of content over the web. Originally started out from the SPDY protocol, now standardized as HTTP version 2.

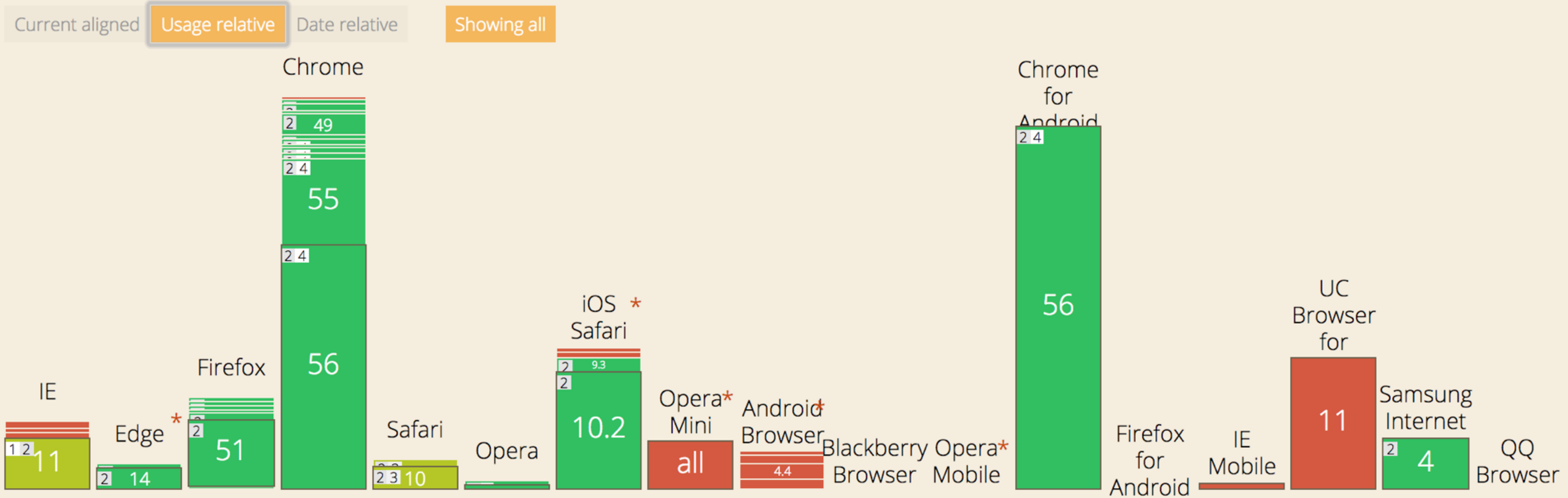


Screenshot: 2016-08-23, caniuse.com

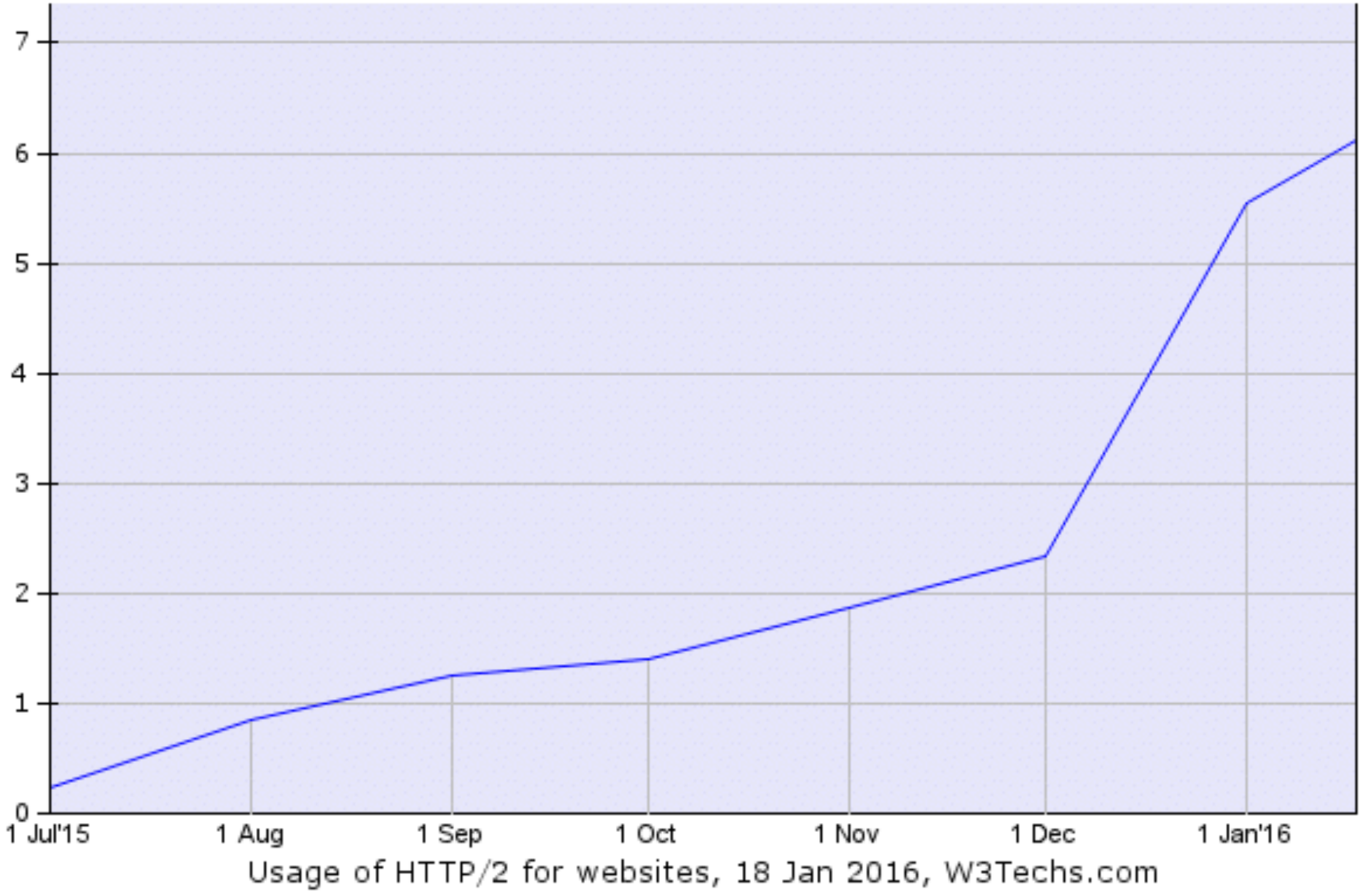
HTTP/2 protocol - OTHER

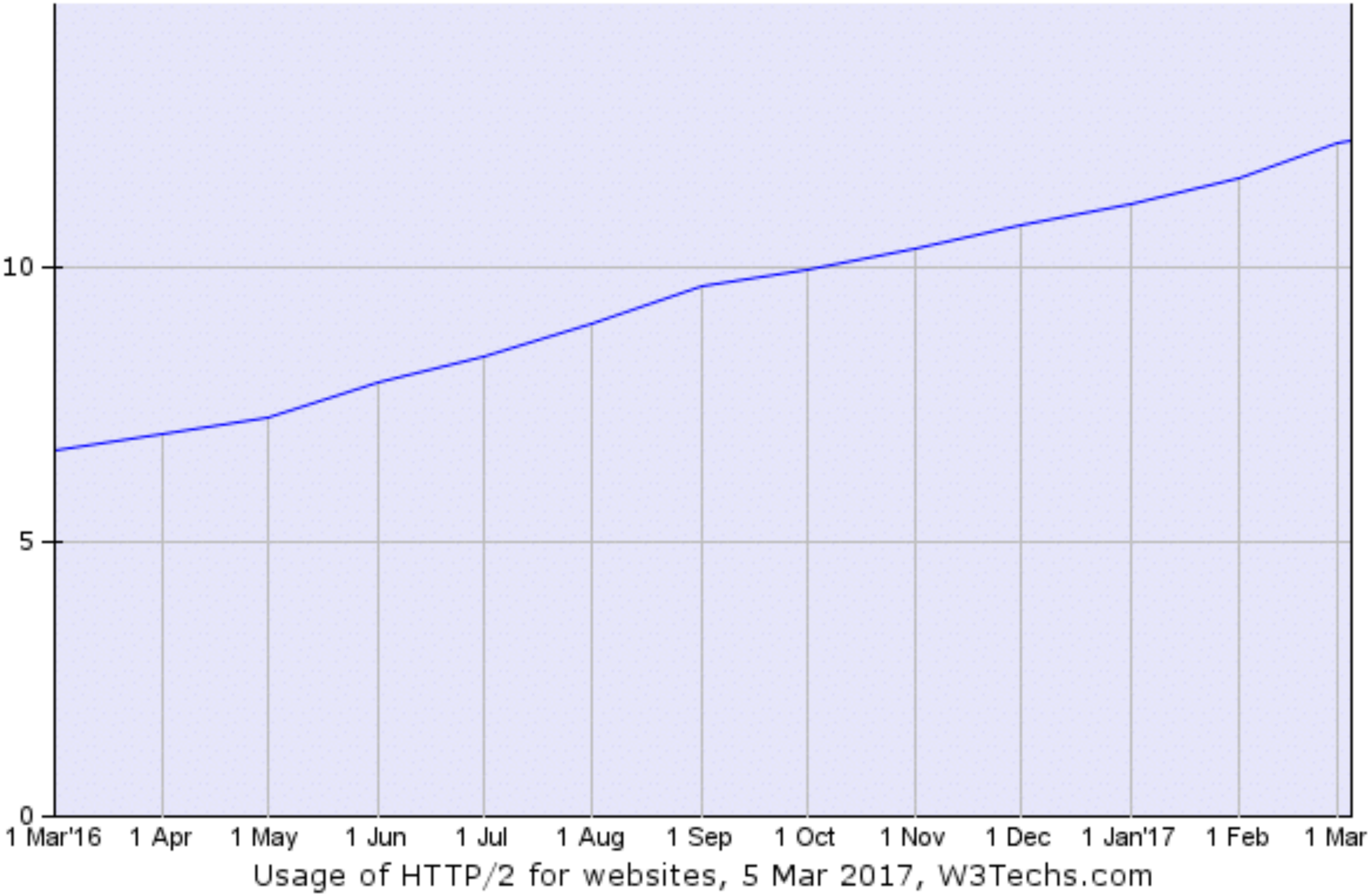
Global 73.91% + 5.65% = 79.56%

Networking protocol for low-latency transport of content over the web. Originally started out from the SPDY protocol, now standardized as HTTP version 2.



Screenshot: 2017-03-05, caniuse.com

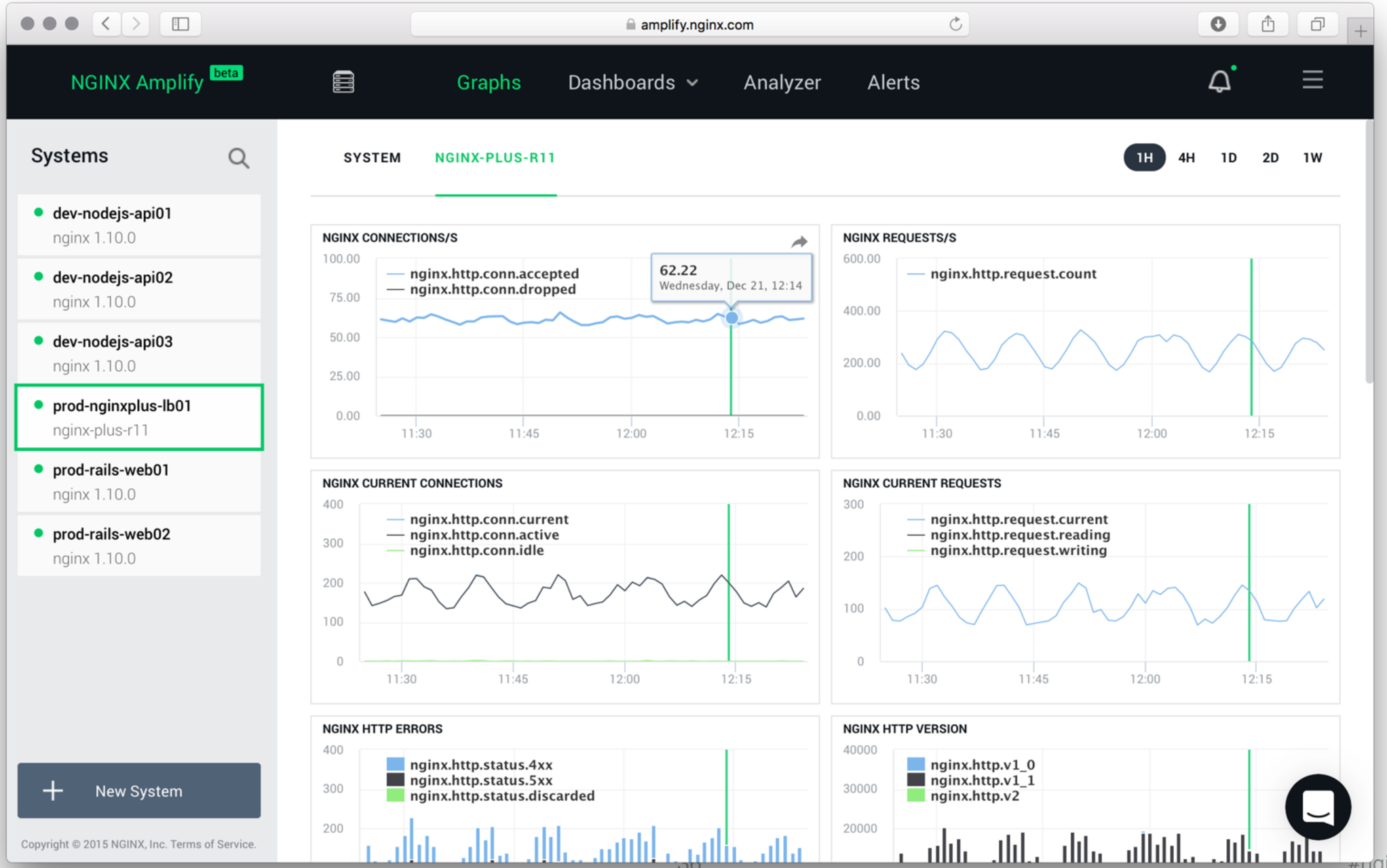

7
6
5
4
3
2
1
0
1 Jul'15
1 Aug
1 Sep
1 Oct
1 Nov
1 Dec
1 Jan'16
Usage of HTTP/2 for websites, 18 Jan 2016, W3Techs.com


10
5
0
1 Mar'16
1 Apr
1 May
1 Jun
1 Jul
1 Aug
1 Sep
1 Oct
1 Nov
1 Dec
1 Jan'17
1 Feb
1 Mar
Usage of HTTP/2 for websites, 5 Mar 2017, W3Techs.com

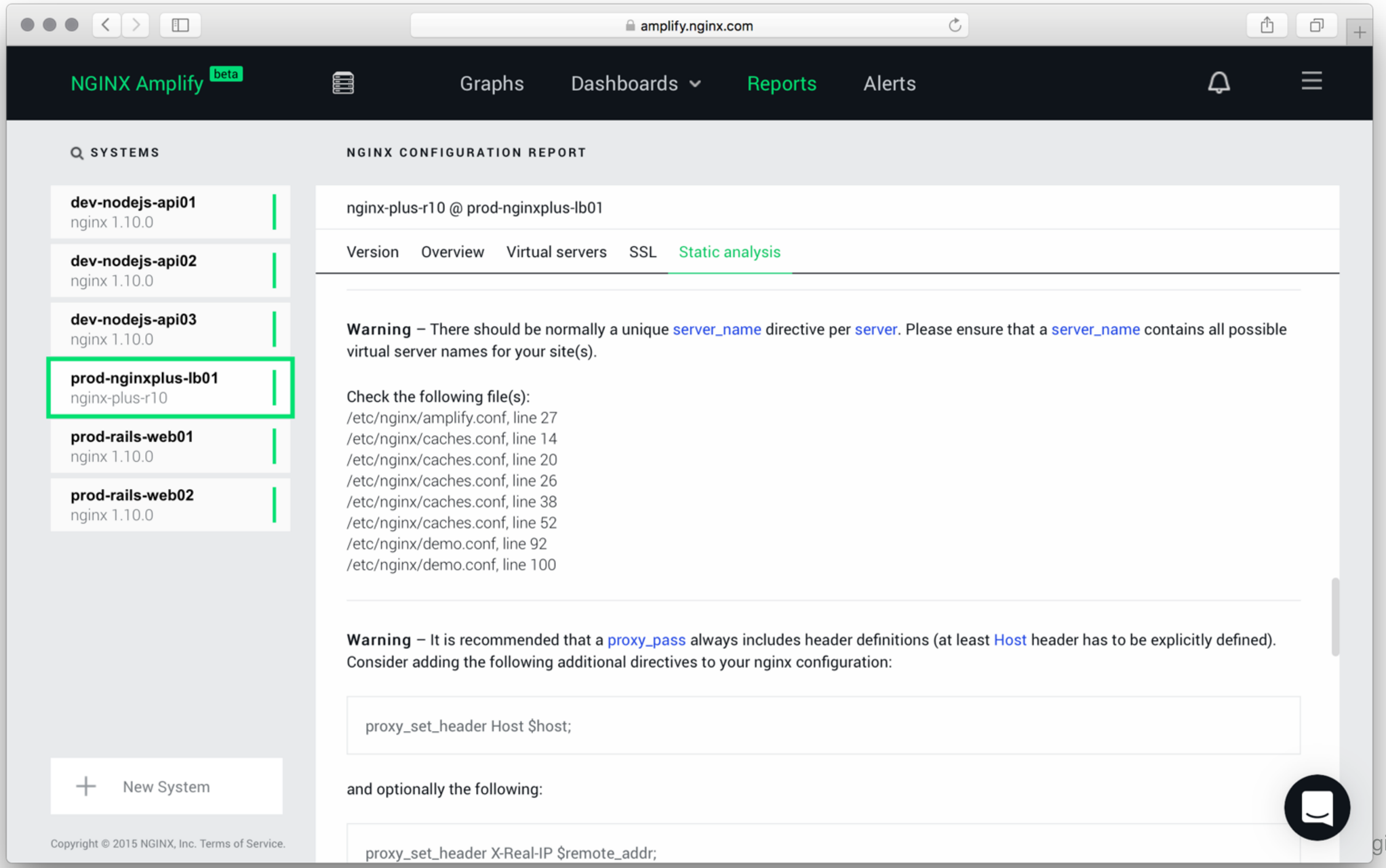
Measure your results

- NGINX provides extensive logs with custom variables. Configure **`log_format`** with:
  `$upstream_response_time`
  `$request_time`
  `$upstream_cache_status`
- NGINX has simple set of metrics with stub_status module. Configure **`stub_status`**
- **NGINX Plus** provides more extensive metrics with Extended Status module
- **NGINX Amplify** is a free monitoring SaaS solution.

Sign up at amplify.nginx.com



Sign up at amplify.nginx.com



Conclusions

- Plan for scalability early
- Tune low level operating system
- Configure Keepalive
- Configure caching
- Enable HTTP/2
- Measure your results

How to Contribute

- hg.nginx.org
- github.com/nginx
- nginx.org/mailman

Thank You

All links in one page:
https://shadrin.org/talks

Twitter: @shadrin

nick@nginx.com