Custom Caching with NGINX:
How I used NPR Digital Services Team's botcache project as a template to protect Drupal websites from bot traffic

Jessica D. Dooley
2020
• The service: 80 Drupal websites
• The problem: bot traffic

How do I protect Drupal websites from traffic spikes that slow performance or consume all system resources?

Two self-hosted tools for web log analysis:
GoAccess: Aggregate analysis of all traffic in logs
https://goaccess.io
Matomo: In-depth, per-domain traffic analysis
https://matomo.org

80 websites
758k requests /day
21k visitors /day
The solution:
A 2014 post by Rick Ennis and the NPR Digital Services team:

https://digitalservices.npr.org/post/hardening-drupal-against-badly-behaved-bots
I replicate the NPR Digital Services team's botcache project in a sandbox environment to answer the following questions.

**Project Questions:**

- Can I replicate this project?
- Will this project scale down?
- Will it fit my organization's needs?
- Are the essential features free?
- How can I measure impact?
- What else do I not know to ask?

https://digitalservices.npr.org/post/hardening-drupal-against-badly-behaved-bots
Goal: determine whether NPR’s botcache NGINX configuration can solve our problem with aggressive drive-by bot traffic.

Sandbox environment:
- Apache VM with Drupal websites
- NGINX VM with reverse proxy & cache
- NGINX VM with botcache

Project guidelines:
- No additional costs
- No impact to customers
- No changes to websites
- Eliminate bot traffic impact and improve website performance

https://digitalservices.npr.org/post/hardening-drupal-against-badly-behaved-bots
Why NGINX?

"The web server should be able to scale nonlinearly with the growing number of simultaneous connections and requests per second."
- Andrew Alexeev, nginx co-creator

- NGINX serves static web content, is a content cache and software load balancer
- NGINX was open sourced in 2004
- NGINX now serves 46% of the top 1k busiest websites
- NGINX Community Edition contains all core features
- Commercial products are available
- NGINX acquired by F5 in 2019

https://w3techs.com/technologies/comparison/ws-apache,ws-nginx
Replicating the NPR botcache project:

“We use a pair of NGINX servers as a reverse proxy cache in front of Drupal. Not only do they help decrease database load by serving frequently used pages, they can also serve pages when the database is down. That helps us maintain uptime for anonymous users even during maintenance windows.”

"Hardening Drupal," Rick Ennis, NPR
Two approaches to keeping cached content fresh:

**Microcaching:**

Microcaching is an effective method for accelerating the delivery of dynamic, non-personalized content by caching it for very short periods of time.

Owen Garrett of F5, "The Benefits of Microcaching with NGINX"
https://www.nginx.com/blog/benefits-of-microcaching-nginx/

**Cache invalidation:**

- NGINX Plus implements PURGE method
- Drupal's cache tags for invalidation
- Third-party implementations available

Original quote credited to Phil Karlton.
{ step 1: sandboxing }

How should I deal with Drupal's cache control headers?

- Ignore them
- Don't modify them
- Pass them to the client

Clients ➔ Requests ➔ Reverse proxy ➔ TLS termination ➔ Cache ➔ Drupal websites

Cache-Control: max-age=300, public
Vary: Cookie, Accept-Encoding
ETag: W/"1604249203"
Expires: Sun, 19 Nov 1978 05:00:00 GMT
Last-Modified: Sun, 01 Nov 2020 16:46:43 GMT
X-Drupal-Dynamic-Cache: MISS
X-Drupal-Cache: HIT
X-Cache-Status: STALE
How does caching work in NGINX?

Ignore some of Drupal's cache control headers:
proxy_ignore_headers Cache-Control Expires Vary;

Select cache matches:
proxy_cache_key $scheme$request_method$host$request_uri;

Determine when to serve cached content:
proxy_cache_methods GET HEAD;  # which methods will be cached
proxy_cache_revalidate on;     # if upstream indicates content unchanged, mark updated
proxy_cache_lock on;          # only one request for stale content is sent upstream
proxy_cache_lock_age 5s;      # grace period before additional requests are sent
proxy_cache_min_uses 1;       # add content to cache after this number of requests
proxy_cache_background_update on;  # serve STALE content while updating in background
proxy_cache_use_stale error timeout updating http_500 http_502 http_503 http_504;

https://docs.nginx.com/nginx/admin-guide/content-cache/content-caching/#specifying-which-requests-to-cache
Cache bypass:

Don't cache authenticated sessions:

```nginx
proxy_cache_methods GET HEAD;  # POST method is not cached
proxy_pass_header Set-Cookie;   # make sure Drupal's auth cookie is passed unaltered
proxy_no_cache $no_cache;      # when $no_cache is set, never cache
proxy_cache_bypass $no_cache;  # when $no_cache is set, bypass cache matches

map $http_cookie $no_cache {
    default 0;
    ~cookievalue 1;
}
```

# map the value of $http_cookie to a $no_cache variable
# by default, the variable is set to off
# if a Drupal auth cookie is found, the variable is on
{ step 1: sandboxing }

NPR's botcache mechanism:

Our NGINX config also includes another element. We have it examine the request's User-Agent and siphon bots and spiders off to a separate NGINX server that we call botcache.

"Hardening Drupal," Rick Ennis, NPR
Replicating NPR's botcache mechanism:

```nginx
map $http_user_agent $botcheck {
    default 0;
    "~*bot" 1;
    "~*spider" 1;
    ...
}

map $botcheck $upstream {
    0 10.0.0.1:80;    # Drupal
    1 10.0.0.2:80;    # Botcache
}

proxy_pass https://$upstream;
```

How does this differ from using Apache mod_rewrite to exclude traffic by matching on User-Agents? Diverting traffic matching a list of bot User-Agents to a cache ensures that ALL requests are served, including search engine spiders, crawlers, and less friendly visitors, without putting load on the upstream server.
Questions answered:

- Can I replicate this project? Yes.
- Are the essential features free? Yes.

Questions addressed next:

- Will this project scale down?
- Will it fit my organization's needs?

Scale: NPR has 3x more websites, 10x more content, ??x more concurrent visitors.

Needs: My logs show 7% of requests are from bot User-Agents.

I need to serve cached pages based on behavior, not identity.
NPR's botcache design:

**Cache clears:** There are none. Pages live on in botcache forever which is fitting since nothing is ever deleted from Drupal.

**Filesystem:** Botcache runs on a regular hard disk so it can have somewhat infinite space. We're currently using around 480G+ for our cache.

**Logic:** Requests for cached pages come strictly from frontcache1/2. If the request is in the cache, it's served up. If it's not, the request is given a 502 and goes no further. In short bots no longer generate Drupal load.

"Hardening Drupal," Rick Ennis, NPR

What if I can configure all the benefits of botcache into my primary cache, limiting traffic based on rate of requests/time instead of a list of User-Agents?
Limit based on $limit_req_zone rather than $http_user_agent

```nginx
geo $limit {
  default 1;
  10.0.0.1/17 0;
  ...  
}

map $limit $external {
  0 "";
  1 $binary_remote_addr;
}

limit_req_zone $external zone=public:10m rate=5r/s;
limit_req_status 429;
```

A primary cache that is fully pre-populated, serves all requests, and protects the upstream with a request rate limit, while exempting known customer campus IPs from the request rate limit.
How will the cache get populated?

We also had a level0 that would spider entire sites with no limit...
Since a spider (we use "wget" with its -r option) crawls in a not-exactly-predefined order, we saw some rather odd results...
We saw as many as 30% misses on stories we expected to be covered by our level0 crawl. As a result we replaced the level0 crawl with a simple database query to dump all /post, /people and /term pages to a file, which we in turn feed to wget (no recursion).

"Hardening Drupal," Rick Ennis, NPR

I can pre-populate the cache per website for every customer, and re-populate it on demand, by exporting a database dump of Drupal's pages and feeding it to wget.
Custom cache population per domain, on schedule and on demand:

while read site; do
    # select the database matching the domain from a lookup table
    db=$(awk -v host="$site" -F, '$0~host { print $1 }' /path/to/databaselist.txt)
    wget https://"$site"/
        --domains "$site"                   # only the specified domain will be included
        --page-requisites                  # populate cache with all page dependencies
        -U "MyCrawlerUserAgent"            # exclude your User-Agent from logging or metrics
        --wait=1                           # choose a rate for the crawl
        --base=https://"$site"              # required in order to invoke input-file of paths
        --input-file=/path/to/"$db$date".txt
        -O /dev/null                       # don't store the downloaded site as a file
        --no-verbose
        --append-output=/path/to/"$site"-"$date".log && \
    echo "$fulldate: completed $site crawl" >> /path/to/crawler.log
done < /path/to/site-list.txt
{ step 3: implementation }

Prepare for deployment:

- build final configuration in staging
- automate configuration changes
- write scripts to roll back each step of the config changes
- automate tests to check for success and notify on failure
- build a runbook enumerating changes
- conduct tabletops of the runbook
- plan for time to roll back, if necessary
- implement runbook during a maintenance window

"Leave room for mistakes, or else they'll make room for themselves."

- ME
Automation:

- Drupal web server sends lists of aliases per website as CSV
- Wget crawler references CSV to populate each website's cache
- Script determines in advance how long a crawl will take
- A system service clears a website's NGINX cache when a Drupal authenticated user invokes Drupal's "Rebuild Cache" button
- Website's cache is scheduled for re-population upon cache clear

Tools:

This automation was accomplished using simple classics:
rsync, mysqldump, awk, bc, grep, cron, scripts implemented as system services
Instrumentation:

- Replicate Apache vhost_combined format for Matomo data continuity
- Separate log format tracks cache performance
- NGINX Amplify dashboard makes use of observability metrics
- Add X-Cache-Status header to instrument cache invocation:

```bash
add_header X-Cache-Status $upstream_cache_status;
```

```bash
log_format cache '$host [$time_local] ' 'Remote: $remote_addr ' 'Upstream: $upstream_addr ' 'Cache Status: $upstream_cache_status ' 'Cache-Control: $upstream_http_cache_control ' '"$request" $ssl_protocol $status $body_bytes_sent "$http_referer" "$http_user_agent" ' 'Args: $args ' 'request=$request_time connect="$upstream_connect_time" header="$upstream_header_time" response="$upstream_response_time" last_modified="$upstream_http_last_modified"';
```
NGINX Amplify: SaaS Monitoring
- A Python agent installed locally sends monitoring data to an Amplify dashboard
- Monitors OS, NGINX, PHP-FPM, and MySQL; enhanced by instrumentation in NGINX logs
- Free for 5 agents for community edition

https://amplify.nginx.com/docs/guide-how-nginx-amplify-agent-works.html
Thoughts about Benchmarking:

Don't conflate environment factors with server performance.

**Apache Bench:**

```
ab -t 300 -c 10 "https://domain.tld/path/"
```

Apache Bench is a single threaded application. What if you fed AB to GNU Parallel?

**Apache Bench with GNU Parallel:**

```
cat URLlist.txt | parallel -j4 'ab -t 300 -c 10 {}'
```

I compared the output of Apache Bench sets run in my staging and production environments:

https://www.gnu.org/software/parallel/
This is ApacheBench, Version 2.3

Server Software: Apache
Server Hostname: apache.domain.tld
Document Path: /about

Concurrency Level: 10
Time taken for tests: 300.030 seconds
Complete requests: 9316
Requests per second: 31.05 [#/sec] (mean)
Time per request: 322.059 [ms] (mean)
Time per request: 32.206 [ms] (mean, across all concurrent requests)
Transfer rate: 843.21 [Kbytes/sec] received

Connection Times (ms)

<table>
<thead>
<tr>
<th>Min</th>
<th>Mean[+/-sd]</th>
<th>Median</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>218</td>
<td>22.4</td>
<td>217</td>
</tr>
<tr>
<td>40</td>
<td>103</td>
<td>16.7</td>
<td>103</td>
</tr>
<tr>
<td>24</td>
<td>85</td>
<td>16.7</td>
<td>83</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>142</strong></td>
<td><strong>321</strong></td>
<td><strong>321</strong></td>
</tr>
</tbody>
</table>

Percentage of the requests served within a certain time (ms)

- 50% 321
- 66% 331
- 75% 337
- 80% 341
- 90% 352
- 95% 362
- 98% 375
- 99% 385
- 100% 539 (longest request)

This is ApacheBench, Version 2.3

Server Software: nginx
Server Hostname: nginx-cache.domain.tld
Document Path: /about

Concurrency Level: 10
Time taken for tests: 263.737 seconds
Complete requests: 50000
Requests per second: 189.58 [#/sec] (mean)
Time per request: 52.747 [ms] (mean)
Time per request: 5.275 [ms] (mean, across all concurrent requests)
Transfer rate: 5648.41 [Kbytes/sec] received

Connection Times (ms)

<table>
<thead>
<tr>
<th>Min</th>
<th>Mean[+/-sd]</th>
<th>Median</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>41</td>
<td>3.2</td>
<td>41</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>1.3</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>1.2</td>
<td>11</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>29</strong></td>
<td><strong>52</strong></td>
<td><strong>53</strong></td>
</tr>
</tbody>
</table>

Percentage of the requests served within a certain time (ms)

- 50% 53
- 66% 54
- 75% 55
- 80% 56
- 90% 57
- 95% 58
- 98% 59
- 99% 60
- 100% 92 (longest request)
ApacheBench - 5 minutes
- 10 concurrency

Apache
9316 requests
31.05 req/sec
32 ms/req
321 median ms/req

NGINX
50000 requests
189.58 req/sec
5 ms/req
53 median ms/req

6.1x faster
This is ApacheBench, Version 2.3

Server Software: Apache
Server Hostname: apache.domain.tld
Document Path: /about
Concurrency Level: 10
Time taken for tests: 300.008 seconds
Complete requests: 12988
Requests per second: 43.29 [#/sec] (mean)
Time per request: 230.988 [ms] (mean)
Time per request: 23.099 [ms] (mean, across all concurrent requests)
Transfer rate: 1401.76 [Kbytes/sec] received

Connection Times (ms)

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>mean[+/-sd]</th>
<th>median</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect:</td>
<td>38</td>
<td>45</td>
<td>27.2</td>
<td>43</td>
</tr>
<tr>
<td>Processing:</td>
<td>53</td>
<td>185</td>
<td>61.2</td>
<td>180</td>
</tr>
<tr>
<td>Waiting:</td>
<td>40</td>
<td>170</td>
<td>60.3</td>
<td>165</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>96</strong></td>
<td><strong>230</strong></td>
<td><strong>67.1</strong></td>
<td><strong>224</strong></td>
</tr>
</tbody>
</table>

Percentage of the requests served within a certain time (ms)

- 50%: 224
- 66%: 252
- 75%: 269
- 80%: 280
- 90%: 311
- 95%: 336
- 98%: 372
- 99%: 404
- 100%: 970 (longest request)

This is ApacheBench, Version 2.3

Server Software: nginx
Server Hostname: nginx-cache.domain.tld
Document Path: /about
Concurrency Level: 10
Time taken for tests: 300.015 seconds
Complete requests: 36868
Requests per second: 122.89 [#/sec] (mean)
Time per request: 81.376 [ms] (mean)
Time per request: 8.138 [ms] (mean, across all concurrent requests)
Transfer rate: 3972.71 [Kbytes/sec] received

Connection Times (ms)

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>mean[+/-sd]</th>
<th>median</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect:</td>
<td>38</td>
<td>46</td>
<td>4.6</td>
<td>46</td>
</tr>
<tr>
<td>Processing:</td>
<td>24</td>
<td>35</td>
<td>5.0</td>
<td>34</td>
</tr>
<tr>
<td>Waiting:</td>
<td>23</td>
<td>30</td>
<td>3.2</td>
<td>29</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>63</strong></td>
<td><strong>81</strong></td>
<td><strong>6.8</strong></td>
<td><strong>81</strong></td>
</tr>
</tbody>
</table>

Percentage of the requests served within a certain time (ms)

- 50%: 81
- 66%: 84
- 75%: 85
- 80%: 86
- 90%: 89
- 95%: 92
- 98%: 95
- 99%: 98
- 100%: 165 (longest request)
ApacheBench

- 5 minutes
- 10 concurrency

Apache
12988 requests
43.29 req/sec
23 ms/req
224 median ms/req

NGINX
36868 requests
122.89 req/sec
8 ms/req
81 median ms/req

Production
2.8x faster
Questions answered:

- Can I replicate this project? Yes.
- Are the essential features free? Yes.
- Will this project scale down? Yes.
- Will it fit my organization's needs? Yes.
- How can I measure impact? ...
- What else do I not know to ask? ...
Documentation is essential. Staging is critical!

https://nginx.org/en/docs/
https://docs.nginx.com/nginx/
https://www.nginx.com/resources/wiki/community/get_involved/
What does your organization need? Solve for that.

Instrument everything.

(Somebody will ask, and that somebody will be you.)

Document everything (for the same reason).

Plan for failure, and have a rollback plan. Automate it. Test it.

What does success look like?
Since the implementation...

- No more incidents caused by traffic
- Less spent on server resources
- Reduced administrative demand
- Maintenance impact on availability reduced
- Request response time drops to 0.36 seconds
- Cache performance metrics:
  
<table>
<thead>
<tr>
<th>Metric</th>
<th>Percentage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIT</td>
<td>50%</td>
<td># Cached content serves the request</td>
</tr>
<tr>
<td>STALE</td>
<td>21%</td>
<td># Cached content is out of date; fresh content loaded into cache</td>
</tr>
<tr>
<td>MISS</td>
<td>15%</td>
<td># Content requested was not present in cache</td>
</tr>
<tr>
<td>-</td>
<td>11%</td>
<td># N/A; redirect, non-cached HTTP method, limit_req_rate, etc</td>
</tr>
<tr>
<td>BYPASS</td>
<td>2%</td>
<td># authenticated user</td>
</tr>
<tr>
<td>UPDATING</td>
<td>&gt;1%</td>
<td># Cache is actively downloading a new copy of content</td>
</tr>
</tbody>
</table>
Resources:

**The Architecture of Open Source Applications Vol II, Chapter 14: nginx**
by Andrew Alexeev (a co-founder of Nginx Inc)
https://aosabook.org/en/nginx.html

**Hardening Drupal Against Badly Behaved Bots**
by Rick Ennis; published Aug 27, 2014
https://digitalservices.npr.org/post/hardening-drupal-against-badly-behaved-bots

**NGINX vs Apache: Our View of a Decade Old Question**
by Owen Garrett (F5); published Oct 9, 2015
https://www.nginx.com/blog/nginx-vs-apache-our-view/

**W3Techs: Web Technologies Surveys**
https://w3techs.com/technologies/comparison/ws-apache,ws-nginx

**GNU Parallel**
https://www.gnu.org/software/parallel/

**Simultaneously Benchmark Many URLs with ApacheBench and GNU Parallel**
by Simon Holywell; published June 25, 2015
Thank you!

@JessicaDDooley

jessica@marchpane.net