
Statamic with Varnish for a high-performance website

Abhishek Tiwari 

Citation: *A. Tiwari*, "Statamic with Varnish for a high-performance website", Abhishek Tiwari, 2014. [doi:10.59350/dqcj5-aa022](https://doi.org/10.59350/dqcj5-aa022)

Published on: June 22, 2014

So you got your [website up and running](#) using Statamic. For better performance you have enabled,

- Statamic rendered-HTML caching.
- Cache-Control header for all static assets
- Gzip compression at web server level

Basically, Statamic rendered-HTML caching stores the entire rendered HTML page and by enabling it you can cut down page-load time by up to 90%~85% on average. You can turn it on by setting `enable` to `true` in `_config/bundles/html_caching/html_caching.yaml`.

But, in some use cases you may want additional performance gain on top of rendered-HTML caching or at least ability to handle high-load conditions. That's where Varnish comes handy. In addition to serving content faster, on a decent server spec (like EC2 micro-instance) Varnish can easily handle 1000-2000 concurrent requests.

Varnish is a web application accelerator also known as a caching HTTP reverse proxy. This is a quick rundown on how to enable Varnish for your Statamic installation. **Please note that** when Statamic content is update approach describe here *does not invalidate Varnish cache automatically*. You will need to invalidate cache manually or [use this varnish cache buster add-on](#).

Setup

Here we are going to setup Varnish as reverse proxy for Apache as web server. Apache is used as web server to serve Statamic content as discussed in [this post](#).

- First install Varnish as root user,

```
apt-get install varnish
```

- Then update `/etc/default/varnish` using following `DAEMON_OPTS`. Here, we are saying that

1. Varnish output will accessible at port 80 on IP address `XYZ.XX.YY.ZZ`
2. Varnish will use `default.vcl` with 256M RAM for in-memory cache
3. Varnish will use 1 worker thread pool, maximum 1000 and minimum 100 threads for this pool

```
DAEMON_OPTS="-a XYZ.XX.YY.ZZ:80 \  
             -T localhost:6082 \  
             -f /etc/varnish/default.vcl \  
             -S /etc/varnish/secret \  
             -s malloc,256m \  
             -p thread_pool_max=1000 \  
             -p thread_pools=1 \  
             -p thread_pool_min=100 \  
             "
```

```
-p thread_pool_add_delay=1 \"
```

- Update `/etc/varnish/default.vcl` with following. First directive enables both Varnish and Apache backend running on port 80 but different IP addresses. In second, we added IP address to to enable purge requests.

```
backend default {
    .host = "127.0.0.1";
    .port = "80";
}

acl purge {
    # Purge requests are only allowed from localhost.
    "localhost";
    "127.0.0.1"/24;
    "XYZ.XX.YY.ZZ"/24;
}
```

- Edit `/etc/apache2/ports.conf` for VirtualHost name and port,

```
NameVirtualHost *:80
Listen 127.0.0.1:80
```

- Update `/etc/varnish/default.vcl` by adding following at the end. Let me explain key sub-routines in following snippet,
 1. In `vcl_fetch` we set `beresp.ttl` depending on file type/extension, this is how long Varnish is gone keep individual page in cache.
 2. In `vcl_recv` we removed cookies from request except for the admin area, added conditional purge based on IP address
 3. In `vcl_deliver` we added X-Cache header with HIT or MISS

```
# Issue purge when there is a cache hit for the purge request.
sub vcl_hit {
    if (req.request == "PURGE") {
        purge;
        error 200 "Purged.";
    }
}

# Set the beresp.ttl
sub vcl_fetch {
    if(req.url ~ "\.(jpg|jpeg|gif|png|ico|css|pdf|js)$") {
        set beresp.ttl = 30d;
    }
    else {
        set beresp.ttl = 2d;
    }
}
```

```
}
}

# Issue a no-op purge when there is a cache miss for the purge request.
sub vcl_miss {
    if (req.request == "PURGE") {
        purge;
        error 200 "Purged.";
    }
}

sub vcl_recv {
    # Remove cookies from request
    if ( !( req.url ~ "^/admin.php/" ) ) {
        unset req.http.Cookie;
    }
    # Verify the ACL for an incoming purge request and handle it.
    if (req.request == "PURGE") {
        if (!client.ip ~ purge) {
            error 405 "Not allowed.";
        }
        return (lookup);
    }
    # Verify the ACL for an incoming ban request and handle it.
    if (req.request == "BAN") {
        if (!client.ip ~ purge) {
            # Not from an allowed IP? Then die with an error.
            error 405 "This IP is not allowed to send PURGE requests.";
        }
        ban("req.http.host == " + req.http.host + " && req.url ~ "+req.url+"$");
        error 200 "Ban added";
    }
    # Only cache responses to clients that support gzip. Most clients
    # do, and the cache holds much more if it stores gzipped responses.
    if (req.http.Accept-Encoding !~ "gzip" || req.http.Accept-Encoding !~ "
        deflate") {
        return (pass);
    }
}

# Add HIT or MISS Header
sub vcl_deliver {
    if (obj.hits > 0) {
        set resp.http.X-Cache = "HIT";
    } else {
        set resp.http.X-Cache = "MISS";
    }
}
```

- Append following to `/etc/apache2/httpd.conf`. This basically unsets the Statamic cook-

ies from response except for the admin area. This is required otherwise Varnish will not cache pages with cookie and serve them directly from Statamic backend.

```
<LocationMatch "^(?!/admin)/[^/]+">
Header unset Set-Cookie
</LocationMatch>
```

- Now restart Varnish and Apache.

```
service varnish stop && service varnish start
service apache2 restart
```

- To purge cache associated with a particular URL (let say `abhishek-tiwari.com/about`) run following from a white-listed IP address,

```
curl -X PURGE abhishek-tiwari.com/about
```

Results

Using [Blitz.io](#), I generated a rush of 29,259 requests in 60 seconds on [this query URL](#) using a load of 1-1000 users.

- This rush generated 29,259 successful hits in 60 seconds and a total 196.30 MB data was transferred.
- The average hit rate of 487.65/second translates to about 42,132,960 hits/day. The max hit rate was: 966 hits per second.
- The average response time was 8 ms. The fastest response time was: 7 ms.

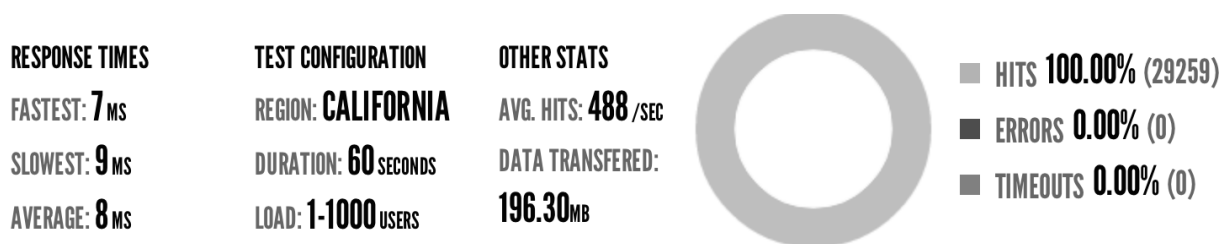


Figure 1: Blitz.io test results for Varnish powered Statamic, 29259 successful hits in 60 seconds

This is a massive improvement on top of HTML caching in terms of performance as well as concurrency capable of delivering 42,132,960 hits/day.

Update I managed to run another Blitz.io test with load of 1-3000 users with 43,041 successful hits in 30 seconds. The average hit rate of 1,435/second which means about 123,958,080 hits/day. To achieve

this I changed the `thread_pool_max` to 3000 and it worked like charm.

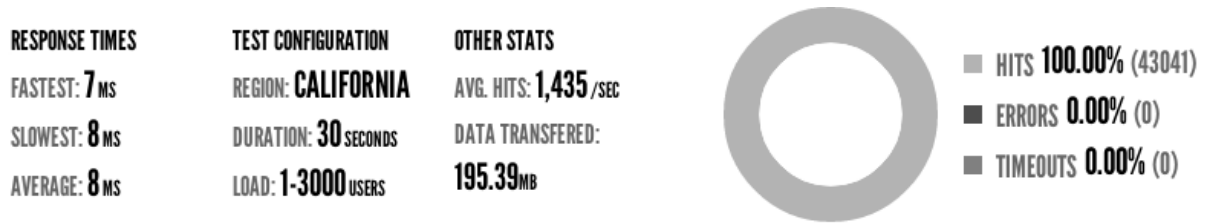


Figure 2: Blitz.io test results for Varnish powered Statamic, 43,041 successful hits in 30 seconds