FREE eBook

# LEARNING varnish

Free unaffiliated eBook created from **Stack Overflow contributors.**

#varnish

# Table of Contents

# About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: varnish

It is an unofficial and free varnish ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official varnish.



Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# Chapter 1: Getting started with varnish

## Remarks

This section provides an overview of what varnish is, and why a developer might want to use it.

It should also mention any large subjects within varnish, and link out to the related topics. Since the Documentation for varnish is new, you may need to create initial versions of those related topics.

## Versions

| Version | Release Date |
|---|---|
| 5.1.2 | 2017-04-07 |
| 5.1.1 | 2017-03-16 |
| 5.0 | 2016-09-15 |
| 4.1.5 | 2016-02-09 |
| 4.0.4 | 2016-11-30 |
| 3.0.7 | 2015-03-23 |

## Examples

### Installation or Setup

The following are instructions to setup latest version of Varnish on various Linux distros.

## CentOS 7

```
curl -s https://packagecloud.io/install/repositories/varnishcache/varnish5/script.rpm.sh |
sudo bash
```

## Ubuntu

```
apt-get install apt-transport-https
curl https://repo.varnish-cache.org/GPG-key.txt | apt-key add -
echo "deb https://repo.varnish-cache.org/ubuntu/ trusty varnish-4.1" \
  >> /etc/apt/sources.list.d/varnish-cache.list
apt-get update
```

```
apt-get install varnish
```

# Debian

```
apt-get install apt-transport-https
curl https://repo.varnish-cache.org/GPG-key.txt | apt-key add -
echo "deb https://repo.varnish-cache.org/debian/ jessie varnish-4.1"\
  >> /etc/apt/sources.list.d/varnish-cache.list
apt-get update
apt-get install varnish
```

## Varnish VCL

Varnish controls and manipulates HTTP requests using Varnish Configuration Language (VCL). The following snippet of VCL removes cookie from incoming requests to /images subdirectory:

```
sub vcl_recv {
    if (req.url ~ "^/images") {
        unset req.http.cookie;
    }
}
```

Read Getting started with varnish online: https://riptutorial.com/varnish/topic/4705/getting-started-with-varnish

# Chapter 2: Building vmods

## Introduction

You don't necessarily have to compile vmods if the binaries for them are already available for your platform. Both CentOS 6 and 7 can leverage COPR builds by Ingvar in order to install a collection of extra modules by Varnish Software: https://copr.fedorainfracloud.org/coprs/ingvar/varnish51/

## Examples

### Compile and install a vmod

Installation of a vmod requires an installed version of Varnish Cache, including the development files. Requirements can be found in the Varnish documentation.

Source code is built with autotools:

```
sudo apt-get install libvarnishapi-dev || sudo yum install varnish-libs-devel
./bootstrap   # If running from git.
./configure
make
make check   # optional
sudo make install
```

Read Building vmods online: https://riptutorial.com/varnish/topic/9669/building-vmods

# Chapter 3: Built-in VCL

## Introduction

The built-in VCL contains procedures that are included and run *last* by Varnish.

They can complement user defined VCL by providing logic that is appropriate for the majority of the sites. For example, skips cache for POST requests, and/or in presence of cookie or authorization headers.

If some of the built-in logic is not needed, a user can add `return()` call from a procedure where built-in VCL logic is not desirable.

## Examples

### Varnish 3.0

```
/*-
 *
 * The default VCL code.
 *
 * NB! You do NOT need to copy & paste all of these functions into your
 * own vcl code, if you do not provide a definition of one of these
 * functions, the compiler will automatically fall back to the default
 * code from this file.
 *
 */

sub vcl_recv {
    if (req.restarts == 0) {
        if (req.http.x-forwarded-for) {
            set req.http.X-Forwarded-For =
                req.http.X-Forwarded-For + ", " + client.ip;
        } else {
            set req.http.X-Forwarded-For = client.ip;
        }
    }
    if (req.request != "GET" &&
      req.request != "HEAD" &&
      req.request != "PUT" &&
      req.request != "POST" &&
      req.request != "TRACE" &&
      req.request != "OPTIONS" &&
      req.request != "DELETE") {
        /* Non-RFC2616 or CONNECT which is weird. */
        return (pipe);
    }
    if (req.request != "GET" && req.request != "HEAD") {
        /* We only deal with GET and HEAD by default */
        return (pass);
    }
    if (req.http.Authorization || req.http.Cookie) {
        /* Not cacheable by default */
```

```
        return (pass);
    }
    return (lookup);
}

sub vcl_pipe {
    # Note that only the first request to the backend will have
    # X-Forwarded-For set.  If you use X-Forwarded-For and want to
    # have it set for all requests, make sure to have:
    # set bereq.http.connection = "close";
    # here.  It is not set by default as it might break some broken web
    # applications, like IIS with NTLM authentication.
    return (pipe);
}

sub vcl_pass {
    return (pass);
}

sub vcl_hash {
    hash_data(req.url);
    if (req.http.host) {
        hash_data(req.http.host);
    } else {
        hash_data(server.ip);
    }
    return (hash);
}

sub vcl_hit {
    return (deliver);
}

sub vcl_miss {
    return (fetch);
}

sub vcl_fetch {
    if (beresp.ttl <= 0s ||
        beresp.http.Set-Cookie ||
        beresp.http.Vary == "*") {
                /*
                 * Mark as "Hit-For-Pass" for the next 2 minutes
                 */
                set beresp.ttl = 120 s;
                return (hit_for_pass);
    }
    return (deliver);
}

sub vcl_deliver {
    return (deliver);
}

sub vcl_error {
    set obj.http.Content-Type = "text/html; charset=utf-8";
    set obj.http.Retry-After = "5";
    synthetic {"
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

```
<html>
  <head>
    <title>"} + obj.status + " " + obj.response + {"</title>
  </head>
  <body>
    <h1>Error "} + obj.status + " " + obj.response + {"</h1>
    <p>"} + obj.response + {"</p>
    <h3>Guru Meditation:</h3>
    <p>XID: "} + req.xid + {"</p>
    <hr>
    <p>Varnish cache server</p>
  </body>
</html>
"};
    return (deliver);
}

sub vcl_init {
        return (ok);
}

sub vcl_fini {
        return (ok);
}
```

## Varnish 4.0

```
/*
 * The built-in (previously called default) VCL code.
 *
 * NB! You do NOT need to copy & paste all of these functions into your
 * own vcl code, if you do not provide a definition of one of these
 * functions, the compiler will automatically fall back to the default
 * code from this file.
 *
 * This code will be prefixed with a backend declaration built from the
 * -b argument.
 */

vcl 4.0;

######################################################################
# Client side

sub vcl_recv {
    if (req.method == "PRI") {
    /* We do not support SPDY or HTTP/2.0 */
    return (synth(405));
    }
    if (req.method != "GET" &&
      req.method != "HEAD" &&
      req.method != "PUT" &&
      req.method != "POST" &&
      req.method != "TRACE" &&
      req.method != "OPTIONS" &&
      req.method != "DELETE") {
        /* Non-RFC2616 or CONNECT which is weird. */
        return (pipe);
    }
```

```
    if (req.method != "GET" && req.method != "HEAD") {
        /* We only deal with GET and HEAD by default */
        return (pass);
    }
    if (req.http.Authorization || req.http.Cookie) {
        /* Not cacheable by default */
        return (pass);
    }
    return (hash);
}

sub vcl_pipe {
    # By default Connection: close is set on all piped requests, to stop
    # connection reuse from sending future requests directly to the
    # (potentially) wrong backend. If you do want this to happen, you can undo
    # it here.
    # unset bereq.http.connection;
    return (pipe);
}

sub vcl_pass {
    return (fetch);
}

sub vcl_hash {
    hash_data(req.url);
    if (req.http.host) {
        hash_data(req.http.host);
    } else {
        hash_data(server.ip);
    }
    return (lookup);
}

sub vcl_purge {
    return (synth(200, "Purged"));
}

sub vcl_hit {
    if (obj.ttl >= 0s) {
        // A pure unadultered hit, deliver it
        return (deliver);
    }
    if (obj.ttl + obj.grace > 0s) {
        // Object is in grace, deliver it
        // Automatically triggers a background fetch
        return (deliver);
    }
    // fetch & deliver once we get the result
    return (fetch);
}

sub vcl_miss {
    return (fetch);
}

sub vcl_deliver {
    return (deliver);
}
```

```
/*
 * We can come here "invisibly" with the following errors: 413, 417 & 503
 */
sub vcl_synth {
    set resp.http.Content-Type = "text/html; charset=utf-8";
    set resp.http.Retry-After = "5";
    synthetic( {"<!DOCTYPE html>
<html>
  <head>
    <title>"} + resp.status + " " + resp.reason + {"</title>
  </head>
  <body>
    <h1>Error "} + resp.status + " " + resp.reason + {"</h1>
    <p>"} + resp.reason + {"</p>
    <h3>Guru Meditation:</h3>
    <p>XID: "} + req.xid + {"</p>
    <hr>
    <p>Varnish cache server</p>
  </body>
</html>
"} );
    return (deliver);
}

#######################################################################
# Backend Fetch

sub vcl_backend_fetch {
    return (fetch);
}

sub vcl_backend_response {
    if (beresp.ttl <= 0s ||
      beresp.http.Set-Cookie ||
      beresp.http.Surrogate-control ~ "no-store" ||
      (!beresp.http.Surrogate-Control &&
        beresp.http.Cache-Control ~ "no-cache|no-store|private") ||
      beresp.http.Vary == "*") {
        /*
        * Mark as "Hit-For-Pass" for the next 2 minutes
        */
        set beresp.ttl = 120s;
        set beresp.uncacheable = true;
    }
    return (deliver);
}

sub vcl_backend_error {
    set beresp.http.Content-Type = "text/html; charset=utf-8";
    set beresp.http.Retry-After = "5";
    synthetic( {"<!DOCTYPE html>
<html>
  <head>
    <title>"} + beresp.status + " " + beresp.reason + {"</title>
  </head>
  <body>
    <h1>Error "} + beresp.status + " " + beresp.reason + {"</h1>
    <p>"} + beresp.reason + {"</p>
    <h3>Guru Meditation:</h3>
    <p>XID: "} + bereq.xid + {"</p>
    <hr>
```

```
    <p>Varnish cache server</p>
  </body>
</html>
"} );
    return (deliver);
}

#########################################################################
# Housekeeping

sub vcl_init {
    return (ok);
}

sub vcl_fini {
    return (ok);
}
```

Read Built-in VCL online: https://riptutorial.com/varnish/topic/5001/built-in-vcl

# Chapter 4: Monitoring Varnish

## Introduction

Use `varnishstat` to monitor the numeric metrics of a currently running Varnish instance. It's location will differ based on your installation. Running `varnishstat -1` will output all metrics in a simple `grep`-able format.

Other utilities are available for watching varnish current status and logging: `varnishtop`, `varnishlog` etc.

## Examples

### Client metrics - incoming traffic

Client metrics cover the traffic between the client and the Varnish cache.

- sess_conn - Cumulative number of connections.
- client_req - Cumulative number of client requests.
- sess_dropped - Dropped connections because of a full queue.

Monitor `sess_conn` and `client_req` to keep track of traffic volume - is it increasing or decreasing, is it spiking etc. Sudden changes might indicate problems.

Monitor `sess_dropped` to see if the cache is dropping any sessions. If so you might need to increase `thread_pool_max`.

```
varnishstat -1 | grep "sess_conn\|client_req \|sess_dropped"
MAIN.sess_conn              62449574         3.38 Sessions accepted
MAIN.client_req            184697229         9.99 Good client requests received
MAIN.sess_dropped                  0         0.00 Sessions dropped for thread
```

### Cache performance

Perhaps the most important performance metric is the hitrate.

Varnish routes it's incoming requests like this:

- Hash, a cacheable request. This might be either `hit` or `miss` depending on the state of the cache.
- Hitpass, a not cacheable request.

A hash with a `miss` and a `hitpass` will be fetched from the server backend and delivered. A hash with a `hit` will be delivered directly from the cache.

Metrics to monitor:

- cache_hit - Number of hashes with a hit in the cache.
- cache_miss - Number of hashes with a miss in the cache.
- cache_hitpass - Number of hitpasses as above.

```
varnishstat -1 | grep "cache_hit \|cache_miss \|cache_hitpass"
MAIN.cache_hit              99032838         5.36 Cache hits
MAIN.cache_hitpass                 0         0.00 Cache hits for pass
MAIN.cache_miss             42484195         2.30 Cache misses
```

Calculate the actual hitrate like this:

```
cache_hit / (cache_hit + cache_miss)
```

In this example the hitrate is 0.7 or 70%. You want to keep this as high as possible. 70% is a decent number. You can improve hitrate by increasing memory and customizing your vcl. Also monitor big changes in your hitrate.

## Monitoring cached objects

You monitor the cached objects to see how often they expire and if they are "nuked".

- n_expired - Number of expired objects.
- n_lru_nuked - Last recently used nuked objects. Number of objects nuked (removed) from the cache because of lack of space.

```
varnishstat -1 | grep "n_expired\|n_lru_nuked"
MAIN.n_expired              42220159          .   Number of expired objects
MAIN.n_lru_nuked              264005          .   Number of LRU nuked objects
```

The one to watch here is `n_lru_nuked`, if the rate is increasing (the *rate*, not only the number) your cache is pushing out objects faster and faster because of lack of space. You need to increase the cache size.

The `n_expired` is more up to your application. A longer time to live will decrease this number but on the other hand not renew the objects as often. Also the cache might require more size.

## Monitoring threads

You need to keep track of some threads metrics to watch your Varnish Cache. Is it running out of OS resources or is it functioning well.

- threads - Number of threads in all pools.
- threads_created - Number of created threads.
- threads_failed - Number of times Varnish failed to create a thread.
- threads_limited - Number of times Varnish was forced not to create a thread since it was maxed out.
- thread_queue_len - Current queue length. Number of requests waiting for a thread.
- sess_queued - Number of times there wasn't any threads available so a request had to be

queued.

```
varnishstat -1 | grep "threads\|thread_queue_len\|sess_queued"
MAIN.threads                     100           .   Total number of threads
MAIN.threads_limited               1        0.00 Threads hit max
MAIN.threads_created            3715        0.00 Threads created
MAIN.threads_destroyed          3615        0.00 Threads destroyed
MAIN.threads_failed                0        0.00 Thread creation failed
MAIN.thread_queue_len              0           .   Length of session queue
MAIN.sess_queued                2505        0.00 Sessions queued for thread
```

If `thread_queue_len` isn't 0 it means that Varnish is out of resources and have started to queue requests. This will decrease performance of those requests. You need to investigate why.

Watch also out for `threads_failed`. If this increases it means your server is out of resources somehow. Increasing numbers in `threads_limited` means you might need to increase your servers `thread_pool_max`.

## Monitoring backend metrics

There are a number of metrics describing the communication between Varnish and it's backends.

The most important metrics here might be these:

- backend_busy - Number of http 5xx statuses recieved by a backend. With VCL you can configure Varnish to try another backend if this happens.
- backend_fail - Number of times Varnish couldnt connect to the backend. This can have a number of causes (no TCP-connection, long time to first byte, long time between bytes). If this happens your backend isn't healthy.
- backend_unhealthy - Number of times Varnish couldn't "ping" the backend (it didn't respond with a HTTP 200 response.

```
varnishstat -1 | grep "backend_"
MAIN.backend_conn           86913481         4.70 Backend conn. success
MAIN.backend_unhealthy             0         0.00 Backend conn. not attempted
MAIN.backend_busy                  0         0.00 Backend conn. too many
MAIN.backend_fail                  7         0.00 Backend conn. failures
MAIN.backend_reuse                 0         0.00 Backend conn. reuses
MAIN.backend_toolate               0         0.00 Backend conn. was closed
MAIN.backend_recycle               0         0.00 Backend conn. recycles
MAIN.backend_retry                 0         0.00 Backend conn. retry
MAIN.backend_req            86961073         4.70 Backend requests made
```

Read Monitoring Varnish online: https://riptutorial.com/varnish/topic/9072/monitoring-varnish

# Credits

| S. No | Chapters | Contributors |
|---|---|---|
| 1 | Getting started with varnish | alejdg, Community, Daniel V. |
| 2 | Building vmods | Daniel V. |
| 3 | Built-in VCL | Daniel V., fgsch, Redithion |
| 4 | Monitoring Varnish | Jensd |