Varnish Cache 101

A technical guide to getting started with Varnish Cache and VCL

prepared by Section

Section

Table of Contents

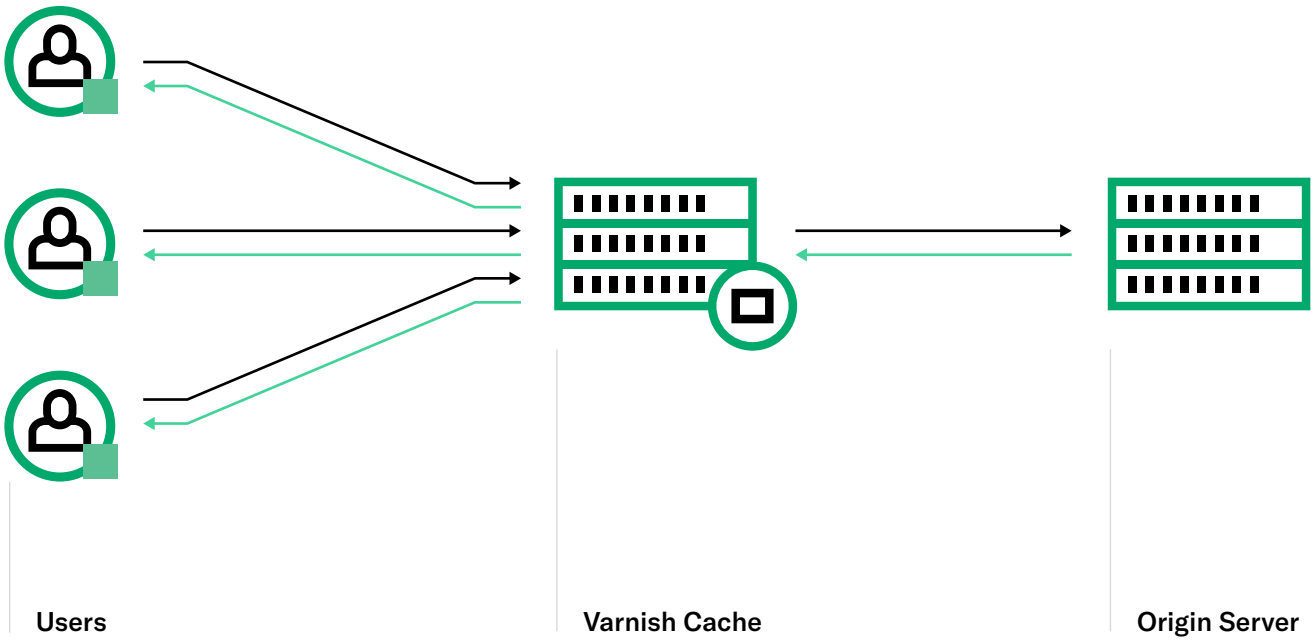
Introduction

Varnish Cache is a reverse proxy for caching HTTP, also sometimes known as an HTTP accelerator. It is most often used to cache content in front of the web server - anything from static images and CSS files to full HTML documents can be cached by Varnish Cache. The key advantages of Varnish Cache are it's speed and flexibility: It can speed up delivery of content by 300-1000x, and because of the flexibility of its domain specific language, Varnish Configuration Language (VCL), it can be configured to act as a load balancer, block IP addresses, and more. It is this combination of speed and configurability that has helped Varnish grow in popularity over older caching reverse proxies like Nginx and Squid.

Varnish Cache is an open-source project first developed by Poul-Henning Kamp in 2005, meaning it can be downloaded and installed by anyone for free. There are also several paid services which provide Varnish Cache as a service or hosted versions of Varnish, including Varnish Software (the commercial arm of Varnish Cache), Fastly (a Content Delivery Network running modified Varnish 2.1), and Section (an Edge Compute Platform offering all versions of unmodified Varnish Cache up to the most current version).

In this paper we will go through the basics of Varnish Cache and what you need to know to get started with VCL. By the end of this paper you should have an understanding of:

- The flow of traffic through Varnish Cache and your web server
  - > Enforcing HTTPs with Varnish Cache
- What type of content you can cache with Varnish Cache
- What is VCL and how each built in subroutine handles HTTP traffic
  - > How to use the built-in and default VCL files
- Methods for caching static objects
  - > Considerations including cookies and your origin configurations
- Methods for caching dynamic content
  - > Caching pages with personalization using hole-punching
- Extending Varnish Cache capabilities with VMODs
- Measuring the success of Varnish Cache

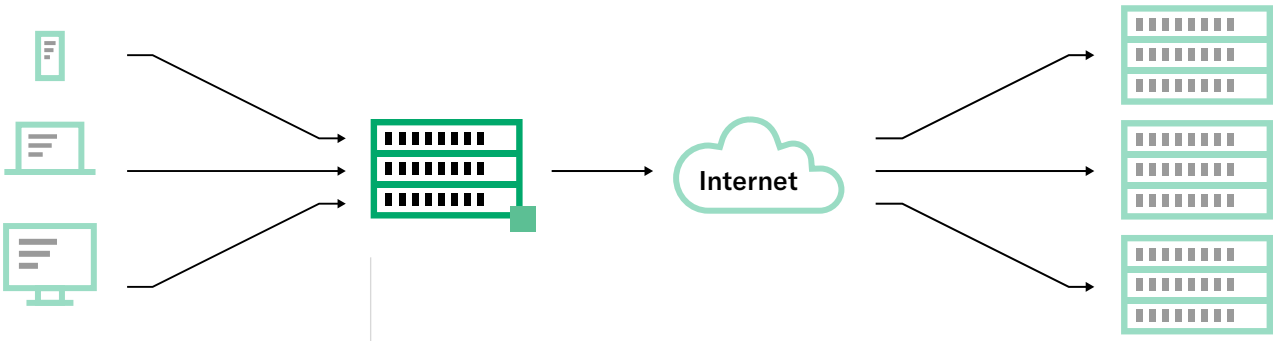


The Basics Of Varnish Cache

Where Varnish Cache Sits

Varnish Cache is deployed as a reverse proxy, a piece of software that sits in front of a web server and intercepts requests made to that server, therefore acting as a proxy to the server a visitor is trying to access. The reverse part comes in because a reverse proxy acts as the server itself, whereas a forward proxy acts on the client or user side to, for example, block access to certain sites within a company network. Reverse proxies have a huge range of uses: They can examine traffic for threats, block bots, and serve cached content directly without traffic needing to go back to the origin server. The Varnish Cache reverse proxy can be configured to do many things but for this paper we are focusing on its main use, caching content. Varnish sits in front of the origin server and any database servers and caches or stores copies of requests which can then be delivered back to visitors extremely quickly.

Forward Proxy

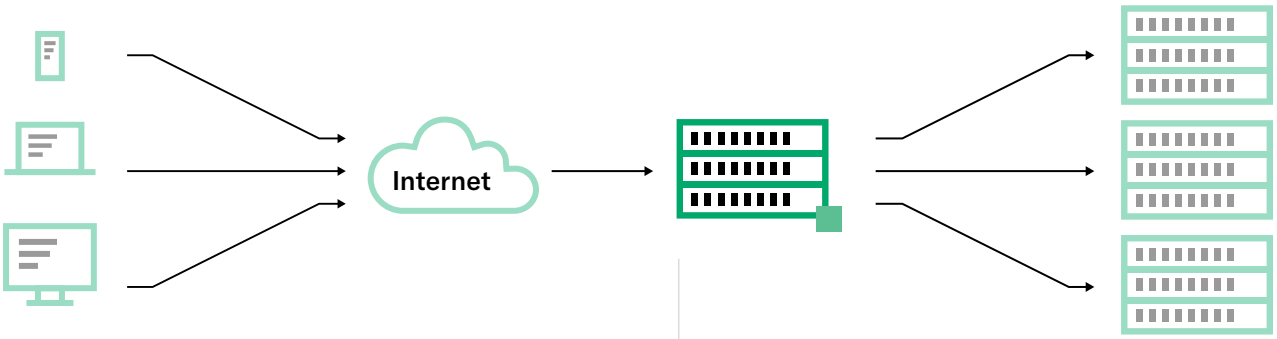


User Devices

Forward Proxy
Restricts certain websites and monitors traffic before leaving a network

Website Origin Servers
Once passed through forward proxy, individual requests go directly to origin servers

Reverse Proxy



User Devices

Reverse Proxy
Acts on behalf of the website origin server. Monitors traffic before arriving at website origin server

Website Origin Servers
All requests go through reverse proxy which makes a single request back to origin server

The Varnish Cache Flow

When a visitor attempts to visit your website or application by going to your IP address, they will be redirected to first go through your Varnish Cache instance. Varnish Cache will immediately serve them any content that is stored in its cache, and then Varnish Cache will make requests back to your origin server for any content that is not cached.

The amount of content that is served from cache depends on how you have configured your Varnish Cache instance (i.e. if you have set it to only cache images) in addition to how "warm" the cache is. When you first cache content and every time the set cache time expires the cache will be "cold" and the next time a visitor comes to your website, it will need to fetch content from the origin before delivering it to that visitor. Varnish Cache fetches the content and, if it is cacheable, will then store it for the next visitor who can be served directly from cache.

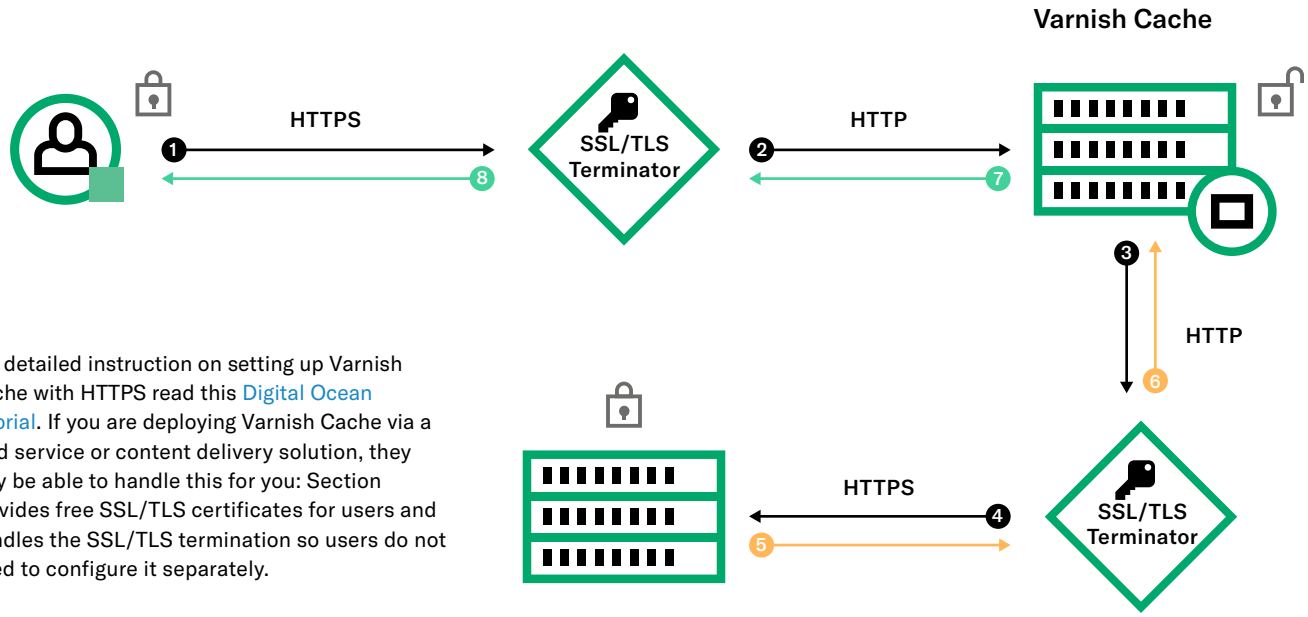
Content that is labeled as uncacheable will never be stored in Varnish Cache and will always be fetched from the origin. Content might be labeled as uncacheable if it has a max-age set at 0, if it has cookies attached, or because you don't want it to be cached. By using VCL you can override headers set at your server that say content should not be cached, and even cache pages around content that is personalized. This means you should, in theory, be able to cache the majority of your requests and achieve a fast webpage for nearly all visitors. The next sections on VCL go more into detail on how to do this.

Varnish Cache And HTTPS

One hurdle of Varnish Cache is that it is designed to accelerate HTTP, not the secure HTTPS protocol. As more and more websites are moving all of their pages to HTTPS for better protection against attacks, this has become something many Varnish Cache users have to work around. To enforce HTTPS with Varnish Cache you will need to put an SSL/TLS terminator in front of Varnish Cache to convert HTTPS to HTTP.

One way to do this is by using Nginx as the SSL/TLS terminator. Nginx is another reverse proxy that is sometimes used to cache content, but Varnish Cache is much faster. Because Nginx allows for HTTPS traffic, you can install Nginx in front of Varnish Cache to perform the HTTPS to HTTP conversion. You should also install Nginx behind Varnish Cache to fetch content from your origin over HTTPS.

In the following graph, the TLS/SSL terminator (such as Nginx) is sitting both in front of Varnish Cache to intercept HTTPS traffic before it gets to Varnish Cache, and behind Varnish Cache so that requests are converted back to HTTPS before going to your origin. As shown by steps 7 and 8, if Varnish Cache already has an item or full page in its cache it will serve the content directly through the first Nginx instance and will not need to request via HTTPS back to the origin.



For detailed instruction on setting up Varnish Cache with HTTPS read this Digital Ocean tutorial. If you are deploying Varnish Cache via a paid service or content delivery solution, they may be able to handle this for you: Section provides free SSL/TLS certificates for users and handles the SSL/TLS termination so users do not need to configure it separately.

What Content To Cache With Varnish Cache

Varnish Cache is powerful because it is so fast, but even more importantly because it has such a wide range of abilities.

Many caching proxies only focus on caching static items like images and CSS files, but due to its flexibility, Varnish Cache can cache static items, the HTML document, and even pages that have personalized elements. When caching first came along in the 1990s it was usually tied to CDNs. CDNs focused on caching images and other content that is the same for all users, often on a separate URL from other content - for example, all images might be fetched from cdn.yoursite.com.

While caching static content solved the challenges of websites in the 1990s - namely that both end-users and the data centers content was served from had much lower bandwidth than those of today - now with huge server farms and fast connections on both sides, to win the speed game you need to do more than cache static items.

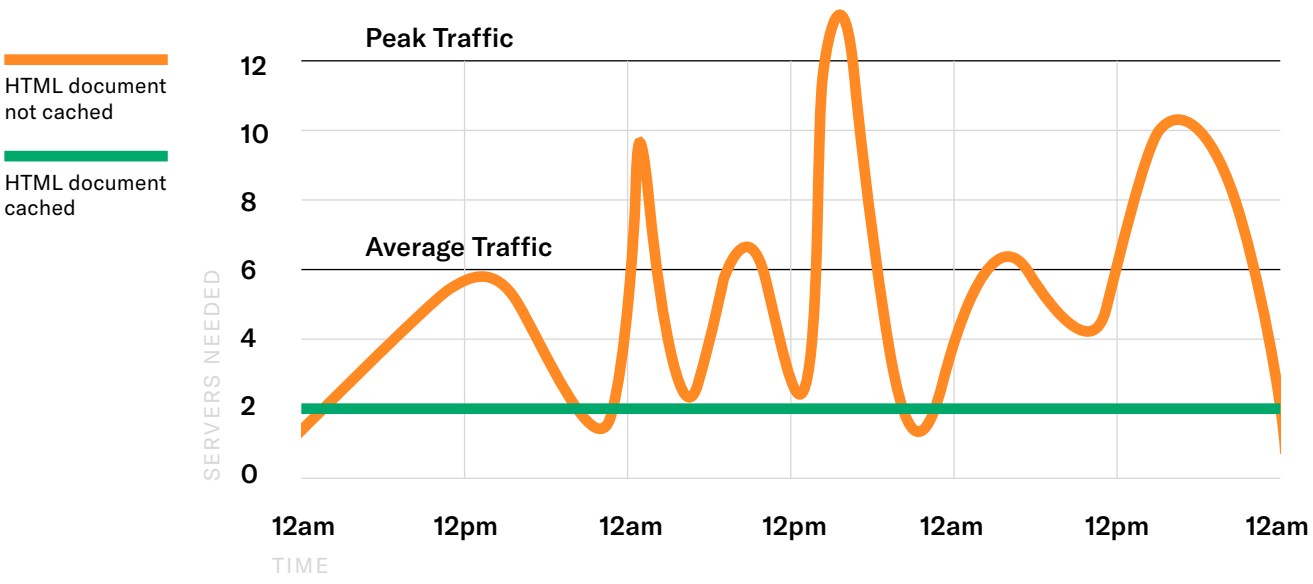
Caching a range of content is where Varnish Cache's flexibility really shines. With Varnish Cache, you can cache what is sometimes called "dynamic content" but usually refers to the HTML document, and cache content around personalized elements using a "hole punching" technique. Examples of content that can be cached with Varnish Cache include:

- Images (PNG, JPG, GIF files)
- CSS stylesheets and fonts
- JavaScript
- Downloadable files
- Full HTML documents

Caching the HTML document is where the real value of Varnish Cache comes in. The HTML document is the first piece of information delivered from the web server to the visitor's browser and includes all the information needed to build a web page, including CSS files, text, and links to images. Before the HTML document is delivered to the browser, the visitor is looking at a blank page with no indication that the page is beginning to load. When the HTML document loads slowly, it delays the time to first byte and start render time, which studies have shown are the most important metrics for both user experience and SEO.

If your web server needs to generate each HTML document individually, you will always need to plan for the peak amount of traffic you expect on your website, as your servers could get overloaded with HTML document requests. Even if images and other linked files are cached, the HTML document will need to be generated for each visitor. This would mean if you have 200 visitors in 1 minute, the web server needs to generate 200 HTML documents. By contrast, if you cache the HTML document you both reduce the time it takes for it to be delivered to the user and reduce the load on your origin servers.

HTML Document Requests



If the HTML document is cached using Varnish Cache and set to live for just one minute, then each minute Varnish Cache will make one request to your back end server for the HTML document, and the other 199 requests will be served directly from Varnish Cache. The speed that Varnish Cache can serve a cached HTML document is extremely fast, often under 200ms, whereas a server generating and serving the HTML document often takes 1 second or longer.

By caching the HTML document the web server only needs to generate 1 HTML document per minute which dramatically reduces the number of servers a website needs to plan for peak traffic. In the below graph, a website's back end needs 12 servers to adequately prepare for peak traffic without the HTML document cached, and only 2 when the HTML document is cached and they can predict exactly how many requests per minute the server will get. This practice saves hosting costs while keeping servers free for critical transactions, and improves user experience by reducing the time to first byte and start render time.

Varnish Cache also allows for caching the HTML document even when it includes personalized elements such as cart size and account information. Using Varnish Cache Configuration Language and a technique called "hole punching" websites can configure their pages so that a majority of the content can be served from cache. In the next section we go into VCL and how to use it to cache all types of content.

Understanding Varnish Cache Configuration Language (VCL)

The reason Varnish Cache is so flexible is due to Varnish Configuration Language (VCL), the domain specific language for Varnish Cache. VCL controls how Varnish Cache handles HTTP requests, and can be thought of as a programming language for HTTP just as PHP is used for server side scripting. Understanding how VCL works is vital to getting a good outcome out of Varnish Cache, but this can be a challenge as most developers will not be familiar with VCL until they start working with Varnish Cache. To understand VCL you must have a grasp of what each VCL subroutine achieves as well as how the built-in VCL file interacts with the default VCL.

Varnish Cache gives users two files on installation: default.vcl and builtin.vcl. The default file is blank and only contains comments. This is the file that you will edit to write VCL specific to your web application. The built-in VCL file gets triggered if you have not overridden each routine in the default VCL. If the default VCL is not edited, Varnish Cache will fall through to the built-in VCL logic. The built-in VCL does have some instructions to cache objects, however because its default behavior is to not cache requests with cookies, and to not override headers set from the server, it often will not cache anything for modern web applications. Because of this, when getting started with Varnish Cache users must edit the default VCL to achieve a solid performance result for their application.

Although Varnish Cache is built so that those requests that are not specifically called out in your default VCL go to the built-in VCL, at Section we recommend programming your default VCL so the vast majority of requests are handled there instead of falling back to the built-in logic. This gives you more control over how each request is handled.

If you have been caching static content with a basic system like Amazon's CloudFront, you could actually see a performance decrease if you switch to Varnish Cache without configuring anything. Although Varnish Cache is a faster and more sophisticated tool which will ultimately provide much better performance results, it requires some configuring to cache content for most visitors.

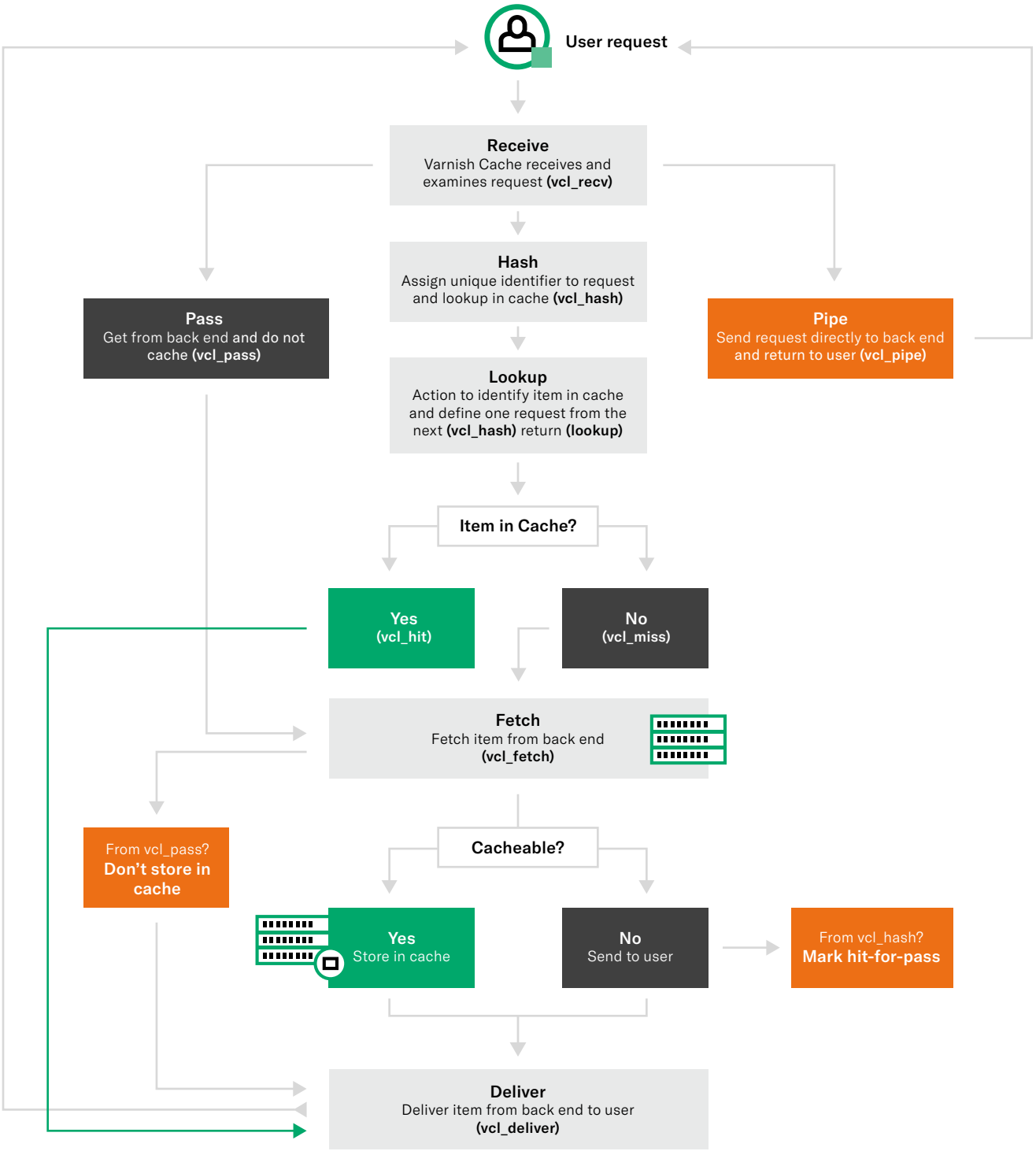
For this reason, we highly recommend reading through the next sections and using other resources like Varnish Software to get an understanding of what VCL will be needed for your application, rather than turning on Varnish Cache with only the built-in VCL. While the built-in VCL is safe in that it will not cache uncacheable content, with an understanding of VCL and your application, you will get a much better result than with the built-in configurations or any other caching solution.

VCL Subroutines

To be able to configure Varnish Cache you will need to learn basic Varnish Cache syntax as well as what each subroutine achieves. It's important to note that the built-in Varnish Cache is always running underneath the VCL you write. Unless you write VCL that terminates a state, each request will fall back to the built-in code. For example, you would write VCL saying "if this request matches these parameters, then try to retrieve it from the cache" and all requests that do not match those parameters would fall back to what is written in builtin.vcl.

The basic VCL request flow is illustrated below, where vcl_recv is the first subroutine that is executed after Varnish Cache has examined the basic information on the request such as verifying it is a valid HTTP request. The flow is divided into client side and back end requests, with all back end requests starting with vcl_backend. While the full Varnish Cache flow is more complex, this basic flow will get you started. It is important to understand the purpose of each subroutine in the below chart, however the ones you will likely alter to customize your Varnish Cache configuration are vcl_recv, vcl_hash, vcl_backend_response, and vcl_deliver.

If you already have a basic understanding of these routines you can skip to the next chapter which gives examples of how to configure these sections to cache more content.



Vcl_recv

vcl_recv is used to normalize the request (for example removing www), decide if a request should be looked up in the cache based on its headers, and execute rewrite rules for certain web applications. What happens to a request in vcl_recv will determine where it is sent to next and ultimately if Varnish Cache will try to cache the request or not. The following actions in vcl_recv will terminate vcl_recv and pass the request on to the appropriate next subroutine:

Pass: A request marked pass will not be looked up in the cache but will execute the remainder of the Varnish Cache flow to fetch the object from the back end. A request marked “pass” will not be stored in the cache once it is delivered to the user from the back end, and this response would be used for objects that you do not want to cache such as personalized information.

Pipe: Piped requests go directly to the back end without Varnish Cache looking at their content. These requests go to vcl_pipe and then essentially skip the Varnish Cache flow and are therefore not cached or logged in Varnish Cache. If a request is piped, any other request using the same keep-alive connection will also be piped. Examples of requests that may be marked “pipe” include music streaming.

Hash: Hash is the lookup function of Varnish Cache, as every request going through Varnish Cache is given a unique numbered identifier. Responding “hash” in the vcl_recv section means the request has no immediate flags saying it has not been cached, so Varnish Cache should look it up in the cache in the vcl_hash routine. This action will be used for items that should be cached, for example any URL in an /images/ folder.

Synth: This is used to generate a response from Varnish Cache, such as an error message, or to redirect requests as set in the vcl_synth routine. One use case is redirects at the edge or denying access to certain requests.

The built-in VCL for this routine can be seen here. Section’s general additions to this section are seen in our documentation.

vcl_recv is also where you can block specific IP addresses or ranges of IPs if you are seeing unusual activity come from certain locations. This is achieved by using VCL such as the below example:

```
acl unwanted { "69.60.116.0"/24; "69.90.119.207"; }

sub vcl_recv
{ if (client.ip ~ unwanted)
{ return(synth(403, "Access denied")); }
}
```

Vcl_hash

The vcl_hash subroutine is at the core of how Varnish Cache defines what separates one cached object from the next and looks up each item in the cache. Requests are routed to this whenever a previous subroutine (ie vcl_recv) returns “hash” for a request. This routine is where Varnish gives each object a numerical unique identifier and then looks up if that object is already in the cache. The built-in VCL will add the IP address and URL of the object to the cache hash. In this section you can also add identifiers for user-specific data (i.e. cookies), however it is recommended that you do this with caution so you don’t accidentally share user-specific data.

Vcl_hash only terminates by returning the “lookup” keyword which is not a subroutine but an action which tells Varnish Cache if the specified object is in the cache or not. If an item is in the cache then the request will go to vcl_hit and vcl_deliver. If the item is not in the cache it will be flagged “busy” and added to the cache, then (assuming it is cacheable) go through the vcl_miss and vcl_fetch routines and be added to the cache with the “busy” flag removed. If new visitors request the same object while the vcl_miss routine is being executed they will be placed in a waiting state until the item is cached. These users will be given the “hit-for-pass” response which is usually itself cached for a minute or two.

Once vcl_hash has completed the lookup action it will move the request on to the next subroutine, which will usually be vcl_hit or vcl_miss.

Vcl_hit

This routine is accessed when vcl_hash successfully finds the specified object in the cache, and in most cases it does not need to be modified. Vcl_hit will usually terminate with one of three keywords: deliver, restart, or synth, but it may also terminate with fetch or pass.

Deliver goes to the vcl_deliver routine and sends the cached object to the client browser as long as the Time To Live (TTL) and grace time have not expired. If the object is stale, meaning the object has 0 or negative TTL with only grace or keep alive time remaining, then it will deliver to the client while simultaneously re-fetching an unexpired copy in the background.

The grace time is so that cached items can be used past their expiry if the back end is down or if Varnish Cache is currently fetching an un-expired version and there are subsequent requests which need to be dealt with. The restart keyword restarts the operation and returns the "guru meditation error" if the max restarts has been reached. This is set to avoid continuous looping.

The synth keyword will return the appropriate status code and reason for termination to the client and stop the request.
Fetch will get a fresh copy of the requested object in the background even though a hit has been delivered, and pass will switch to pass mode and move to vcl_pass.

Vcl_miss

Similar to vcl_hit, vcl_miss usually does not need to be modified from the built-in. This subroutine is called when vcl_hash does not find the requested object in the cache, and it then decides if it should fetch said object from the back end server or return vcl_pass if it is uncacheable. Vcl_miss will terminate with fetch, pass, restart, or synth, as described above. Since the object is not yet in the cache it will not return deliver before the object has been fetched from the back end.

Vcl_pass

This routine is called when an item is not cached and is labeled as uncacheable. In this state the request will forward to the back end through vcl_backend_fetch and then deliver to the client without saving in the cache through vcl_deliver. Requests that are not caught in vcl_hit or vcl_miss will usually fall through to this state. Vcl_pass will terminate with fetch, restart, or synth and doesn't usually need to be modified

Vcl_backend_fetch

Vcl_backend_fetch is called before fetching the requested object from the back end. You can alter the request before it goes to the back end in this section, and it will terminate by returning fetch which will fetch the object, or abandon which will send the request to vcl_synth and send a 503 error to the client.

Vcl_backend_response

Vcl_backend_response occurs after vcl_backend_fetch when Varnish Cache has successfully retrieved response headers from the back end. This section is where you can edit headers from the origin to override settings such as "max-age: 0" so that requests are cacheable in Varnish Cache. Many origins are not configured to cache content out of the box , so you will often need to make edits in this section.

This section can terminate with deliver, abandon, or retry. The deliver keyword will see if the response is cacheable (a 304 code) and if so create a cache object with the specified TTL. Abandon will stop the request and send it to vcl_synth with a 503 code, and retry will retry the request until it hits the maximum number of retries, at which point it will forward to vcl_backend_error.

Vcl_deliver

Vcl_deliver is the final subroutine called before a request is delivered to the client and can be used to modify the response headers before they are sent to the client. This routine can terminate with deliver, restart, or synth. Some of the variables that may be modified here using "set" to override or create and "unset" to undo or delete are resp.http for client headers, resp.status for the status code, and obj.hits for the number of cache hits for that object.

Conclusion on subroutines

While it is important to understand all of the above subroutines in Varnish Cache, the ones you most often will be working with to cache content on your website are vcl_recv, vcl_backend_response, vcl_hash, and vcl_deliver. In the next sections we will go over some common scenarios and how to get started with using VCL to cache your website's contents.

VCL Syntax

The syntax used in all VCL subroutines is based on C and Perl. The below show some of the main variables that can be used in each VCL subroutine and what they will achieve. For example, req.http can be used in vcl_revc and vcl_deliver and will access the request headers. For a full list of variables visit the Varnish Cache docs.

Variable	Accessible In	Description
req	vcl_recv, vcl_deliver	HTTP request data structure
Key	**Type**	**Description**
http	HEADER	Object with request headers
method	STRING	The request type (e.g. GET)
url	STRING	The request URL
resp	vcl_deliver, vcl_synth	HTTP response data structure
http	HEADER	Object with response headers
status	INT	The response status (eg 200)
bereq	backend_response, backend_error	Back end request data structure
http	HEADER	Object with back end request headers
method	STRING	Back end response type
url	STRING	The back end request URL
beresp	backend_fetch, backend_error, backend response	Back end request data structure
http	HEADER	Object with back end response headers
status	INT	The response status
ttl	DURATION	Object's remaining time to live', seconds
uncacheable	BOOL	Makes object uncacheable
do_esi	BOOL	ESI-process the object after fetching Set to true to parse object for ESI

Below are the logical operators in VCL:

Operator	Description
=	The assignment operator
==	The comparison operator
~	The match operator. Can either be used with regular expressions or ACLs
!	The negation operator
&&	The logical AND operator
||	The logical OR operator

Caching For Your Application In VCL

When utilizing Varnish Cache and VCL, it is important to remember that every application is unique and will have different caching needs. The recommendations in this section are examples of caching configurations which Section comes across often, however they will need adjusting for your website. Websites looking to cache dynamic content or use hole-punching for pages with personalization will often need to make changes to their source code in addition to VCL to prepare pages for this type of caching. If you want help with your VCL, you can always reach out to Section at contact@section.io or pose a question in the Section community forum.

Caching Statistics

Caching static objects is the first step to speeding up your website with Varnish Cache. To do this you will need to look at the Varnish Cache built-in VCL for a few routines and see where it will impact your application and not cache content. When editing VCL, you will make changes where necessary to allow for caching of more content, and then allow requests to fall through to the built in VCL.

In the below examples we are using Varnish 4.0 VCL. More recent updates are based on Varnish 4.0 and are mostly patches and additions to functionality. However, Varnish 3.x and below are quite different to 4.0 so you will need to look at Varnish Cache's instructions on using 3.0. First let's look at vcl_recv.

vcl_recv built-in VCL

```
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);
}
```

vcl_recv built-in VCL pseudo code

```
sub vcl_recv {
        if the request method is not equal to
        GET, HEAD, PUT, POST, or DELETE then
                then pipe the request

        if the request method is not equal to
        GET or HEAD then
                then pass on the request

        if there is an Authorization or
        Cookie header
                then pass on the request

        if none of the above
                then give the request to vcl_hash
}
```

From this we can tell that Varnish Cache will only cache GET and HEAD requests and pipe all other requests to your origin. We can also see that if Varnish Cache detects cookies or authorization tokens it will not cache anything by default. This will include static items such as images that are rarely impacted by such personalization tokens and therefore should be able to be cached.

Now let's look at vcl_backend_response:

vcl_backend_response built-in VCL

```
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);
}
```

vcl_backend_response built-in VCL pseudo code

```
sub vcl_backend_response {
        if any of the following is true:
        back end response TTL is less than 0,
        back end response has a set-cookie header,
        back end response cache-control header
            contains "no-cache" or "no-store"
            or "private"
        back end response "vary header is equal to
            "*"
        then set the response as uncacheable
        for 120 seconds

        then give the response to vcl_deliver
            which either:
            puts the response in cache or
            does not put the response in cache
}
```

This section is looking at the response your back end is sending to Varnish Cache, and by default will not override back end time to live (TTL) and cache control settings but will mark them as "hit-for-pass" and then ultimately pass them. If Varnish Cache finds a back end that is configured to allow caching, it will by default cache items for 2 minutes or 120 seconds.

By first examining these two subroutines in relation to your application you will immediately see some areas where VCL can be edited for a higher cache hit rate for static objects. For example, most modern websites do set cookies immediately, and many back ends set a cache-control max-age = 0 which tells Varnish Cache the item is uncacheable.

Here are some ways you can use VCL to get around cookies and back end server settings using VCL:

Cookies and UTMs: Cookies set by your origin server (which are different from those set by Google Analytics or other JavaScript tracking mechanisms) are utilized to track unique user sessions and behavior. You need to be careful when removing cookies from certain requests, as you never want to accidentally share user session cookies or user-specific response bodies. However, when caching static objects such as images which are not unique to each user, using VCL to remove cookies and then allow for caching is imperative as without managing cookies almost nothing will be cached by the built in VCL.

```
sub vcl_recv {
        if (req.url ~ "/assets") {
                unset req.http.Cookie;
        }
}
```

Use this line of code in the vcl_recv section to tell VCL to remove cookies from static assets stored in an "assets" folder. Storing static files in a specific folder rather than caching them by file type (JPG, PNG, etc) is a more secure way to implement static caching.

You may also want to remove the query strings that are added to track in Google Analytics, Marketo, MailChimp, and other marketing programs. By default Varnish Cache will see these as unique URLs and not cache them even though they are serving the same content. To remove these use this code. If you use tracking other than those detailed below you can simply add them to the list.

```
sub vcl_recv {
        # Strip browser side tracking script to improve cache hit rate
        if  (req.url ~ "[?&](utm_source|utm_medium|utm_campaign|glicd|cx|ie|cof
                siteurl | mc_cid|mc_eid)="    {
                set req.url = regsuball(req.url, "(?:(\?)?|&)(?:utm_source|utm_medium|utm_campaign|
                gclid|cx|ie|cof|siteurl|mc_cid|mc_eid)=[^&]+",    "\1");
                set req.url = regsuball(req.url, "(?:(\?)&|\?$)", :\1");
        }
}
```

You can also remove the cookies set by Google Analytics and other programs in VCL. Varnish Software has a primer on how to do this.

Cache-control and Time to Live: Web servers often set headers which say "Cache-Control: Max-age=0," and TTLs of 0. The TTL will tell Varnish not to cache an object, and the cache-control will tell the client side not to cache an object. When set up correctly, browser and server caches should work together so that, when possible, content is served from a browser cache. This reduces the requests to both your web server and your VCL server. With VCL you can set the browser to cache content for longer to increase the likelihood that content will be served directly from the user's computer.

Using VCL in the vcl_recv and vcl_backend_response routines you can override these headers without having to touch your origin settings. The below example will cache all items stored in the /assets/ folder in Varnish Cache for 3 days and in the browser for 2 days. You will often want to make the browser cache shorter than the Varnish Cache, as your cache clear will not reach the browser side.

This VCL will also only cache responses with a status code under 400 so error pages do not get cached.

```
sub_vcl_recv:
        if  (req.url ~ ""/assets")  {
                return (hash);
}
sub_backend_response
if (beresp.status < 400) {
                if (bereq.url ~"/assets") {
                unset beresp.http.set-cookie;
                        set beresp.ttl = 259200s;
                        #The number of seconds to cache inside Varnish
                        set beresp.http.Cache-Control = "max-age=172800s";
                        #The number of seconds to cache in browser
                }
}
```

You can see more of Section's recommended VCL in GitHub and will get a default_vcl file of your own including quick configurations for static caching if you sign up for a free Section trial.

Caching Dynamic Content

Caching dynamic content, which includes full HTML documents, AJAX requests and personalized elements, is more tricky than caching basic static objects but also gives a much bigger payoff in the end. The reason many websites stay away from caching dynamic content is that it is perceived to be risky and many solutions do not offer adequate testing before going to production. This can lead to web pages being unavailable if the HTML document is improperly cached, or personalized elements shared between users. Luckily, with well configured VCL and a knowledge of how to test and measure your Varnish configuration you can successfully cache dynamic content and see an increase in page load time and reduction in server load.

Especially with dynamic content, it is important to understand that the code examples given below will not work for every application. You will need to adjust these examples to work for your specific users and setup, but they should serve as a good starting point for learning the different methods to cache dynamic content using Varnish Cache.

Caching The Full HTML Document

Depending on your website configuration, caching the full HTML document can be as simple as removing any "if" statements from the VCL that would restrict caching only to certain files. For example, you could add the below to the backend_response routine which would cache all files for one hour:

```
sub vcl_backend_response {
Set beresp.ttl = 3600s;
}
```

This simple line of code assumes that your server is not setting cookies, users do not have tracking codes attached to them, and there are no unique elements on the page, which is why it almost always used in conjunction with more complex VCL or back end configurations.

Another option if your server is configured for HTML caching is below:

```
sub vcl_recv {
    if (req.method != "GET" && req.method != "HEAD" && req.method != "PURGE") {
        return (pass);
    }
```

Going into all of the configurations you will need to consider when caching the HTML document for your specific application is beyond the scope of this guide, but please reach out anytime to contact@section.io if you have specific questions about your VCL. In the next sections we discuss how to cache for anonymous users and use AJAX calls and Edge Side Includes (ESI) to cache your page around personalized content.

Methods For Caching Around Personalization

Caching for Anonymous Users: One way to cache the HTML document even when it has personalized elements is to implement anonymous caching. The next session discusses using AJAX calls and ESI to cache around personalized content, but some websites will hard-code personalized information into the HTML document, and without making changes to the source code this will render the page uncacheable. However, personalized information only comes into play once a user takes an action to move out of an anonymous state and into a user-specific state. This action might be logging in or adding an item to the cart.

The difficulty comes in when the server sets a user-session cookie at the very beginning of a session, even if the user is browsing the site anonymously at first. If a user does not have a session cookie set, they will be considered to be in an anonymous user caching mode. Anonymous User Caching can examine every request from the origin and prevent the session cookie being set on the user's browser. As these responses are generic they can be stored in cache and served to other users. This is done by stripping set-cookies on origin response resources that are requested by anonymous users.

Once a user performs an action such as add to cart or login, the cache detects this and from then on all HTML document requests for that user pass through to the origin servers without being cached. At this time the set-cookie header from the origin response is allowed to be sent to the browser. Once the session cookie is set, subsequent requests from this user will bypass the cache, and the user is no longer considered to be anonymous. Although you will not see as high a cache hit rate improvement using this strategy, it will improve your HTML document cache hit rate without as much configuration as AJAX or ESI require.

An example in VCL is below. To see the pseudo code and logic behind this method read this article.

vcl_recv built-in VCL

```
sub vcl_recv {
        if (req.method != "GET" && req.method  != "HEAD" && req.method != "PURGE") {
        return (pass);
        }
        // Rest of vcl_recv code
    }
sub vcl_backend_response {
        if (bereq.method != "GET" && bereq.method != "HEAD" && bereq.method != "PURGE") {
                set beresp.uncacheable = true;
                set beresp.uncacheable = 120s;
                return (deliver);
        }
        if (beresp.http.Content-Type ~ "text/html" && beresp.status < 400) {
                if (bereq.http.Cookie !~ "ASP.NET_SessionId" && bereq.url !~
                        "(/[Cc]art|/[Cc]heckout|/[Aa]dmin|/[Ll]ogin|[/Aa]ccount)") {
                        unset beresp.http.Set-Cookie;
                        set beresp.ttl = 6h;
                        set beresp.grace = 12h;
                } else {
                        set beresp.uncacheable = true;
                        set beresp.ttl = 120s;
                }
                return (deliver);
        }
    }

sub vcl_hash {
        if (req.http.Cookie !~ "ASP.NET_SessionId" && req.url !~
        "(/[Cc]art|/[Cc]heckout|/[Ll]ogin|/[Aa]ccount)") {
                hash_data("no_unique_cookie");
        } else {
                hash_data("has_unique_cookie");
        }
    }
```

Using this code, when a new user navigates to your website they would not have a session cookie and would be served responses from the cache that have the set-cookie header stripped. As soon as the user performs an action such as POST or go to a URL that is not cached such as /cart, the cache will allow the origin to set a cookie. Once the session cookie is set future requests will be marked as uncacheable. While static assets will still be served from cache, HTML documents containing personalization will not be.

AJAX Calls vs Edge Side Includes (ESI): Although the above example will work for some users, to get a better cache hit rate on your HTML document you will want to make some adjustments that allow a page to be cached around personalization. In most cases websites will not be able to cache the HTML document without some adjustments to their VCL and/or source code to account for personalized elements.

The two main ways to cache around personalized elements or "hole-punch" pages are with AJAX calls and Edge Side Includes (ESI). While these two methods achieve the same thing they go about it in different ways, and you will need to decide which is most appropriate for your website configuration and team's abilities.

AJAX Calls: AJAX stands for Asynchronous JavaScript And XML and uses both a browser built-in XMLHttpRequest object to get data from the web server and JavaScript and HTML DOM to display that content to the user. Despite the name "AJAX" these calls can also transport data as plain text or JSON instead of XML.

AJAX calls use a JavaScript snippet to load dynamic content. As a basic example you could configure a page counter that changed each time a page is reloaded by programming a snippet that is loaded after the main content:

```
<script>
$.getJSON (
, function(data) {console.log(data); $('#p').html(data.pagecount);});});
</script>
```

Because this is called following the rest of the page, you can cache the entire HTML document of the page without worrying that the dynamic content will be broken. AJAX calls are beneficial for several reasons. Unlike the Edge Side Includes we discuss below, AJAX do not depend on an ESI processor to run, so you could build and test an AJAX call locally without having to run Varnish Cache locally. Since JavaScript and AJAX are common in development it can also be easier to get started with AJAX than with ESI. AJAX is also a good solution for handling failures because you can program custom error messages related to, for example, a user's account or cart information.

Edge Side Includes: Edge Side Includes tells Varnish Cache to cache a page but fill in the blank dynamic content by fetching that content from the origin server. This is done by adding an ESI line such as `<esi:include src='/pagecount-esi'/>` for page count.

By using ESI you can cache the full HTML document with the dynamic content and do not need to use JavaScript. A key point of ESI is that Varnish Cache will start to send the HTML document immediately even while it is fetching the dynamic content from the origin—this will keep your total HTML document load time around the same but dramatically improve the time to first byte which is an important metric for both search engine ranking and perception of load time.

With ESI you need an ESI processor such as Varnish Cache. Some benefits of ESI are that it allows you to program specifically within the Varnish Cache layer and does not require the use of JavaScript. This can be beneficial if your users' devices/browsers do not accept JavaScript. ESI is also a good solution for API calls which typically do not execute JavaScript. One downside of ESI is that it can be difficult to test properly if you do not have a specific testing solution like Section's Developer PoP installed.

HTML Streaming (caching only the <head>): Another more advanced solution for caching dynamic content if your website has too many personalized elements or other restrictions which do not allow for AJAX or ESI calls is to use a feature we have at Section called "HTML Streaming." HTML Streaming is available from Section and Instart Logic and allows for caching of the HTML document <head> section only so that the personalized content contained in the <body> is not cached but the user still experiences a fast time to first byte and start render time. This solution utilizes both Varnish Cache and LUA. We recommend only advanced VCL users attempt a solution like this but you can see a code example here. To use HTML Streaming through Section and have a Section engineer set it up for you please contact us.

Extending Varnish Cache Capabilities

One of the useful features of Varnish Cache is that in addition to using VCL to write specific features for your website, there are Varnish Modules available which will extend the capabilities of Varnish Cache without you needing to write many lines of VCL. VMODs are extensions which allow for additional programming as they are free of some restrictions found in VCL, such as not making external calls natively.

Varnish Cache keeps a full list of VMODs, but below are the ones we typically use at Section to add to Varnish Cache capabilities.

• Varnish UUID Module/ import uuid: Allows you to generate unique random IDs or hashes based on values that you provide. Can be used to build features that require unique IDs or hashing of strings.

• Varnish GeoIP Lookup Module/ import geoip: This allows you to look up a visitor's GeoIP address and include instructions that take their location into account. This is especially important for sites with global visitors as it can send them to the correct site (www.mysite.co.uk vs www.mysite.com). You will need a GeoIP library installed to lookup the GeoIP: at Section we are able to gather city-level information.

• Header VMOD/ import header: This allows you to manipulate requests more than in the default VCL, making it easy to set and handle cookies and group and consolidate headers. This module is now included in a Varnish Module Collection that has been put together - other features in the collection include variable support, advanced cache invalidates and more.

Options To Install Varnish Cache

When installing Varnish Cache you have a few options: Install the open-source version yourself, use one of Varnish Software's paid products which includes support and additional capabilities built on top of Varnish, or deploy Varnish Cache within a content delivery solution such as Fastly (running modified Varnish 2.1) or an Edge Compute Platform such as Section (running all versions of Varnish Cache up to the latest release). Below we go through the pros and cons to each solution so you can decide which option will be right for your web application.

Open Source Varnish Cache

Varnish Cache is an excellent open source project with a healthy community around it which is why many websites choose to install the open source version themselves. This has several benefits, the first and, for some, most important factor being that it is free to download and use. There also may be reasons your organization would prefer to deploy Varnish Cache on-premise rather than in a cloud-based solution. In addition, by using the open-source version you'll always have the option to self-update to the newest Varnish Cache version.

By using the open source Varnish Cache you can always rely on the Varnish Cache documentation, Varnish Software documentation, and Varnish community. You will also know exactly what VCL you are using and don't have to worry about modifications which will make it more difficult to use these open source resources.

However, despite being free this option does come with associated costs. These include the costs to host your own Varnish Cache server and costs around using significant developer resource to set up and maintain the Varnish Cache server. In addition, because Varnish Cache out-of-the-box does not include user-friendly monitoring and metrics tools, you will likely need to spend resource setting up your own logging and alerting system to track the performance of Varnish and alert you if something goes down. It can also be quite complex managing Varnish-specific setup issues such as the SSL termination needed to use Varnish with HTTPS.

Another downside of installing Varnish Cache yourself is the lack of structured support. While you have access to the Varnish Cache community you will not have immediate access to a designated support team who can answer questions or assist you while you are learning VCL or troubleshooting issues.

Varnish Software

Varnish Software is the commercial arm of Varnish Cache and the company offers several paid services built on top of the open source Varnish Cache. These include their core product Varnish Plus, Varnish Plus Cloud which deploys Varnish Plus on cloud infrastructure, and Varnish API Engine for API management. Varnish Software also offers Varnish Extend, a type of Content Delivery Network which is discussed in the next section.

Varnish Plus is useful for your organization if you are looking for additional Varnish Cache modules and configurations and professional support offered by Varnish Software. Varnish Software can also handle SSL/TLS termination for you and provides an administration console. This can be a good option if you are looking to deploy Varnish on-premise but want a better interface to work with than the bare VCL, need support, or have an advanced use case.

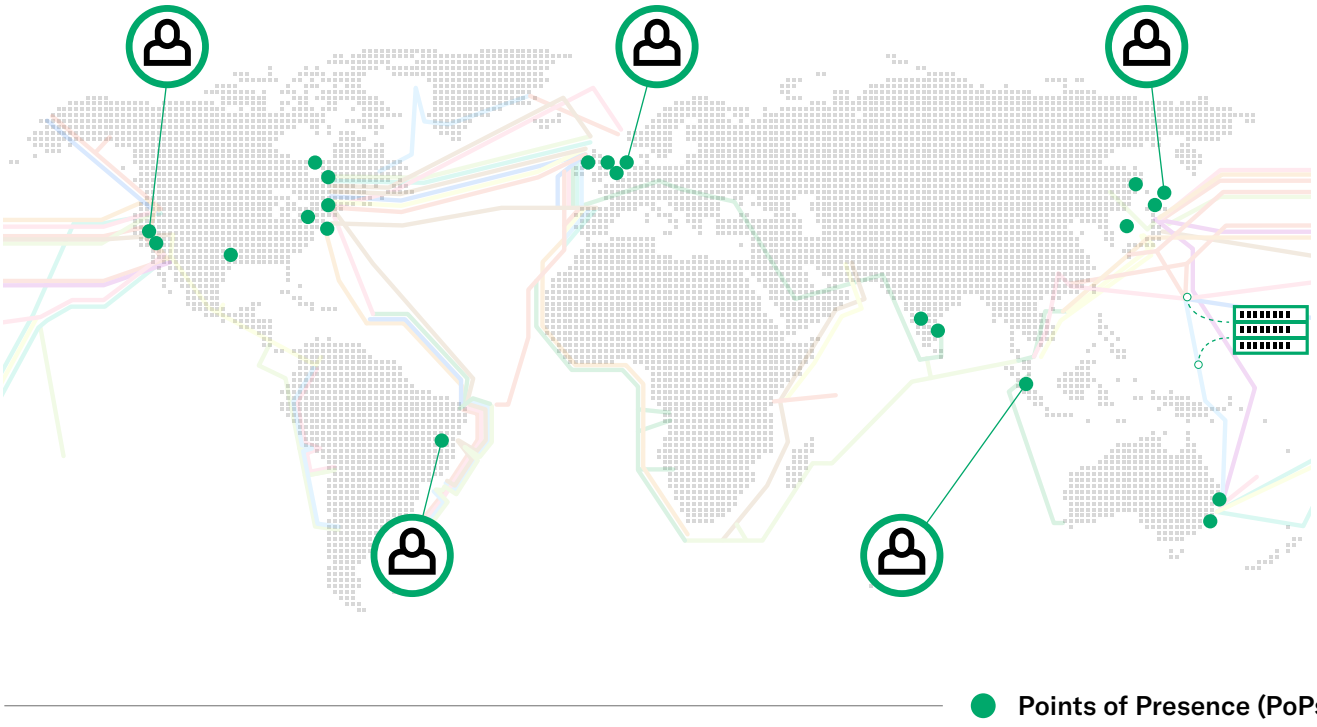
The downside of these solutions is that they can be prohibitively expensive, and some support levels offer just 20 support requests a year. The plans start at $31,000 for a three node cluster and go up based on needs and additional features.

CDN or Edge Compute Platform

The final option is to deploy Varnish Cache globally using a Content Delivery Network (CDN) or Edge Compute Platform. These solutions consist of two layers - the DNS layer which routes requests to the server closest to the user, and the reverse proxy layer which includes software such as Varnish Cache (or other caching module), along with other edge modules, such as web application firewalls, bot blockers, and more. Many traditional CDNs offer older caching proxies like Nginx or Squid, but there are several more modern solutions, including Section's Edge Compute Platform, which base their caching on Varnish Cache.

As mentioned above, Varnish Software has a Content Delivery Network “Varnish Extend” which is described as a self-assembled CDN. Varnish Extend provides traffic management through Cdexis and instructions on setting up a custom CDN. This custom CDN could be a hybrid of commercial CDNs and private servers, a private CDN, or use Varnish as an origin shield.

This option allows you to create a CDN specific to your needs, however this will require a large amount of ongoing work to set up and maintain, and Varnish Extend does not assist you with the actual implementation. In addition, Varnish Extend only deploys Varnish as a reverse proxy and applications would need to separately configure other edge modules such as WAFs. Therefore Varnish Extend is only applicable for very large enterprises who want to build a custom CDN using Varnish Cache and get support directly from Varnish Software.



Points of Presence (PoPs)

Fastly is a Content Delivery Network which uses a modified version of Varnish Cache for caching static and dynamic content and also offers a rules-based WAF. Fastly also includes metrics, logs, varying levels of support and SSL/TLS certificates at an additional cost to their monthly fee.

While Fastly takes advantage of the speed of Varnish Cache, it can be difficult to fully configure and test Fastly’s modified VCL without paying their professional services team for assistance. Because the VCL is modified, the open source documentation and community may not be able to assist you in configurations. Fastly does offer a rules-based WAF but does not have the advanced security solutions that other CDNs offer.

Section is a modern, flexible Edge Compute Platform which offers users all Varnish Cache versions up to the newest release. Section also includes ELK Stack logs, robust Grafana metrics, free SSL/TLS certificates and certificate management, and a local Developer PoP for testing the full Varnish Cache and edge delivery setup before pushing to production. By testing the full Varnish Cache configuration locally, Section users can cache more content without the risk of broken caching or session leakage.

Section gives developers full control over their Varnish Cache configuration, so they are able to edit VCL directly to get the best possible performance out of Varnish Cache, or get assistance from the Section support team. Section is also unique for content delivery solutions in that it allows users to deploy Varnish Cache on the Section global PoP network, on a private PoP network, or on-premise with all of the DevOps tools Section’s platform provides.

Choosing The Right Solution For You

When choosing the Varnish Cache deployment mode that is right for your application you should consider costs (including cost to maintain and set up the open source version), ease of use, accessibility to logs and metrics, and the type of support you think will work best for your development team.

Writing custom Varnish Cache configurations involves learning VCL and continually tuning your VCL setup to get the best cache performance. For small organizations with fewer developer resources, a solution that provides guidance on VCL and access to Varnish Cache experts may be preferable to an open source installation.

Applications that serve global audiences will benefit from a global deployment of Varnish Cache rather than a single node installed in one location. Utilizing a CDN or a more modern Edge Compute Platform with Varnish Cache will allow them to get the fast performance of Varnish Cache's caching combined with additional speed that comes with reducing the distance content needs to travel to users.

Above all you should choose a solution which can adapt to your needs, gives you full control over your configuration, and allows you to test VCL before it goes to production. Many websites do not take advantage of the full power of Varnish Cache because they do not have the power to edit VCL and have no way of testing the full Varnish Cache configuration before going to production. This often leads to websites not utilizing the dynamic caching features Varnish Cache is known for out of fear that this caching will break in production. We strongly recommend using a tool like Section's Developer PoP to test Varnish Cache before it goes to production.

Measuring Success In Varnish Cache

Without proper logs and metrics you have few ways of knowing if Varnish Cache is giving you the performance benefits you are looking for.

Once you have set up Varnish Cache one of the most important things you will need to do next is set up your metrics and monitoring systems. Without proper logs and metrics you have few ways of knowing if Varnish Cache is actually doing what you think and giving you the performance benefits you are looking for. In addition, Varnish Cache often needs to be tuned after the initial setup to ensure that new items are properly cached and you are accounting for the behaviors of your specific users in your cache setup.

How To Monitor Varnish Cache

To understand how Varnish is behaving you'll need to process logs provided by Varnish Cache's toolset. First you'll need to configure varnishncsa to start writing logs and set up a log file rotation. You can then search logs using grep to find the requests you want to examine. Varnishstat is another tool that provides several interesting statistics that you can examine when you have a problem, or use as ongoing statistics to regularly check the health of your system.

These logs will be in a very basic format and may need sifting through to fully comprehend what is going on. You can see a list of the Varnish Cache metrics and counters to look at here. A more advanced setup can aggregate and consolidate this data in an easy to use way that can be utilized by both development and operations teams to diagnose and resolve issues. Solutions such as Section will provide detailed logs and metrics out of the box, and we've also outlined some free tools below that you can use to set up logging yourself.

First, to understand things like cache hit rates by content-type, user-agent and GeoIP:

1. Use varnishncsa with a decent log format that captures a lot of data.
2. Ship those logs to a centralised log processing system, using rsyslog or syslog-ng.
3. Run Logstash using a syslog input or UDP input to receive the log lines.
4. During Logstash processing, use the GeoIP filter and user-agent filter detection to enrich the data.
5. Set up statsd from etsy, and point the Logstash output to statsd.
6. We set statsd to flush aggregate data with means, medians and various percentiles to carbon-relay, the component of the Graphite stack that receives data.
7. carbon-relay pushes to carbon-cache, which persists the files.
8. We then use graphite-web to perform ad hoc queries on the data.

Secondly, for statistics from the instances of Varnish Cache:

1. We run varnishstat as a collectd job periodically.
2. Collectd forwards the data obtained from varnishstat to our carbon-relay as above.
3. Carbon-relay sends the data to carbon-cache.
4. We can then perform ad hoc queries on a single instance of varnish or look at the varnish cluster as a whole.
5. Graphite-web supports creating dashboards, so you can use those ad hoc queries you find interesting and group them together to build reusable metrics for Varnish Cache that you can use to maintain the health of your system.

In order to make these more manageable, you can use Tessera or Grafana to build dashboards with better interfaces that suit your system's requirements without needing complex programming. At Section we use a combination of Graphite metrics with Grafana dashboards that are set up and ready to go in the Section portal.

Varnish Cache Logs

In addition to monitoring tools you'll also want to set up some detailed logging tools which will give you statistics on number of errors and other important information. We recommend setting up a centralized log system based on the ELK stack which will store logs and allow you to search and visualize them easily. As a quick refresher on the ELK stack, it consists of these three open source tools working together:

ElasticSearch is a near-real time search engine that, as the name implies, is highly scalable and flexible. It centrally stores data so documents can be searched quickly, and allows for advanced queries so developers can get detailed analysis. ElasticSearch is based on the Lucene search engine, another open-source software, and built with RESTful APIs for simple deployment.

LogStash is the data collection pipeline which sits in front of ElasticSearch to collect data inputs and pipe said data to a variety of different destinations - ElasticSearch being the destination for this data when utilizing the ELK Stack. LogStash supports a wide range of data types and sources (including web applications, hosting services, content delivery solutions, and web application firewalls or caching servers), and can collect them all at once so you have all the data you need immediately.

Kibana visualizes ElasticSearch documents so it's easy for developers to have immediate insight into the documents stored and how the system is operating. Kibana offers interactive diagrams that can visualize complex queries done through ElasticSearch, along with Geospatial data and timelines that show you different services are performing over time. Kibana also makes it easy for developers to create and save custom graphs that fit the needs of their specific applications.

You can get detailed instructions on setting up the ELK Stack from Digital Ocean. To set up an ELK stack for Varnish Cache, follow these basic steps:

1. Run varnishncsa on your hosts, and use rsyslog or syslog-ng to ship the data to a Logstash endpoint.
2. Configure Logstash to accept your data, and enrich the data with the various filters that Logstash provides.
3. Configure Logstash to output your data to an elasticsearch cluster.
4. Use Kibana to query the data in an ad hoc fashion, or build your own Varnish Cache management console. Using a combination of these logs and the metrics described above will ensure you have the answers to any questions about Varnish hit rate, error rates, and more.

Metrics To Examine

Once you have a metrics system set up you'll need to determine which metrics to look at to properly assess how Varnish Cache is working. While you will likely want to look at some specific metrics for your website, here are the major Varnish Cache metrics you should utilize to monitor the health of your Varnish Cache configuration.

Cache hit rate: The overall cache hit rate is usually the most looked at cache metric because it demonstrates the percentage of requests that have been successfully served from cache. While you can use this to get a quick look at your cache performance, it is more important to understand what types of files are being cached and why certain file types are not being cached, as explained in the next section.

Cache hit by asset type: By looking at cache hit rate by asset type (images, CSS, HTML document, Javascript) you can get a much clearer picture of how much content is being cached by VCL. Because websites have a large number of images, if all images are cached this will inflate the overall cache hit rate and hide the fact that the files which take longer for your server to produce - such as HTML documents - are not being cached.

Cache miss: A cache miss means Varnish Cache looked for this item in the cache and did not find it - this could be because it is the first request for that item and it needs to be fetched from the back end before being cached, or because Varnish Cache thought it was a cacheable item but found for some reason it was not cacheable. If your VCL is configured well, you will have a low cache miss rate.

Cache pass: A cache pass means an item is marked as uncacheable in a subroutine. While the item still passes through Varnish Cache to be delivered, Varnish Cache does not try to look it up in the cache or try to cache it before it is delivered. A high cache pass rate could mean you are not caching as much content as you should to achieve optimal performance.

Time to serve by cache status: This metric will tell you how long an item takes to be delivered if it was a cache miss, hit, or pass. By looking at this metric you will see the difference in delivery times for each status.

Time to serve by asset type: This metric shows you how long each asset type takes to be delivered. You can use this metric to tell what file types are taking longer to deliver, and thus optimize your caching configuration to cache as many of those files as possible.

Varnish Cache Deployment Checklist

It can at first seem that Varnish Cache is relatively simple to install, but as this guide has demonstrated there are many aspects of Varnish Cache you will need to master before achieving a large performance improvement. Especially with an increased focus on DevOps collaboration, you will need to install several systems in addition to Varnish Cache itself to properly measure and troubleshoot the setup.

The most important decision you'll need to make is if you are going to run Varnish Cache on your own servers, as a managed service, or as part of your content or edge delivery solution. The below checklist is applicable for both those running Varnish Cache themselves and those planning to run Varnish Cache as part of a content or edge delivery solution. For those running Varnish Cache as part of a content or edge delivery solution, you should make sure you know the answers to each of these questions as it relates to your edge deployment so you can make an informed decision on which provider to go with.

1. How will I develop and test Varnish Cache?
How are you going to run Varnish Cache locally, and where you are writing your application? To get a great result with Varnish Cache you need to tailor your VCL to meet the design of your application. Trying to introduce Varnish Cache in later phases of the development lifecycle, such as only using Varnish Cache in testing and production environments, usually gives a suboptimal result.

2. How will I run Varnish Cache?
How are you planning to size your cache and manage the resources required by Varnish Cache? On lower memory systems Varnish Cache can mean that Disk IO becomes a concern. Are you able to meet the needs of your cache as your application changes?

3. How will I monitor Varnish Cache's performance?
How will you know if Varnish Cache is working well? Does the managed service or edge solution provide detailed metrics? For those running Varnish Cache on-premise, you could build a Varnish Cache metrics infrastructure that does log shipping of the varnishncsa logs to Logstash, Statsd and Graphite. From there you could build the dashboards that you'll need to monitor during a continuous delivery system.

4. How will I debug Varnish Cache?
When something in your monitoring shows that you're not caching as expected, how will you know what Varnish Cache is doing? Perhaps you'll SSH into a Varnish Cache server and run varnishlog. You might be able to do this in your development environment, but you might also need to run that in your production environment. If you have load balanced, highly available Varnish Cache servers, you might need to run varnishlog in production.

5. How will I find opportunities to improve my caching in Varnish Cache?
How will you know what is the next big thing to cache? A Varnish Cache metrics system will provide you with a console that you could use to identify what is causing a cache miss in Varnish Cache.

6. How will I handle HTTPS and Varnish Cache?
How are you going to wrap HTTPS around Varnish Cache? You'll need some SSL/TLS termination proxy in front of Varnish Cache, such as Nginx. Does your edge solution provide this at no extra cost? Also, you'll need to consider the security between Varnish Cache and your application - that connection may need to be secured also.

7. How will I serve errors and custom messages?
How are you going to handle errors in Varnish Cache? You might want some custom error pages or other error handling to ensure that your customers get a good experience when your back end is unavailable or under maintenance.

8. How will I manage and store my Varnish Cache logs?
Your logs hold a lot of valuable data, and unfortunately they are often removed by logrotate or something similar. Getting value from your logs requires analysis tooling, like the ELK stack, Elasticsearch, Logstash and Kibana.

9. How will I know when Varnish Cache or my server is down?
You may need to get alerts when your Varnish Cache performance degrades, or when Varnish Cache starts to throw errors. Does the edge solution you are considering have a built-in alerting system?

10. How will I manage configuration of Varnish Cache and collaboration among my development team?
What will you use to manage changes to Varnish Cache? You might use Git to source control your VCL files, and then use Jenkins and Puppet or Chef to manage the deployment of those changes at the right time.

Installing Varnish Cache

Now that you know the benefits of Varnish Cache and some basic VCL, let's look at how to get started with Varnish Cache. Here is the quickest way to install the open source:

Quick Varnish Cache Install

```
Ubuntu, Debian:

    apt-get update
    apt-get install varnish

RHEL, CentOS and Fedora:

    yum update
    yum install varnish
```

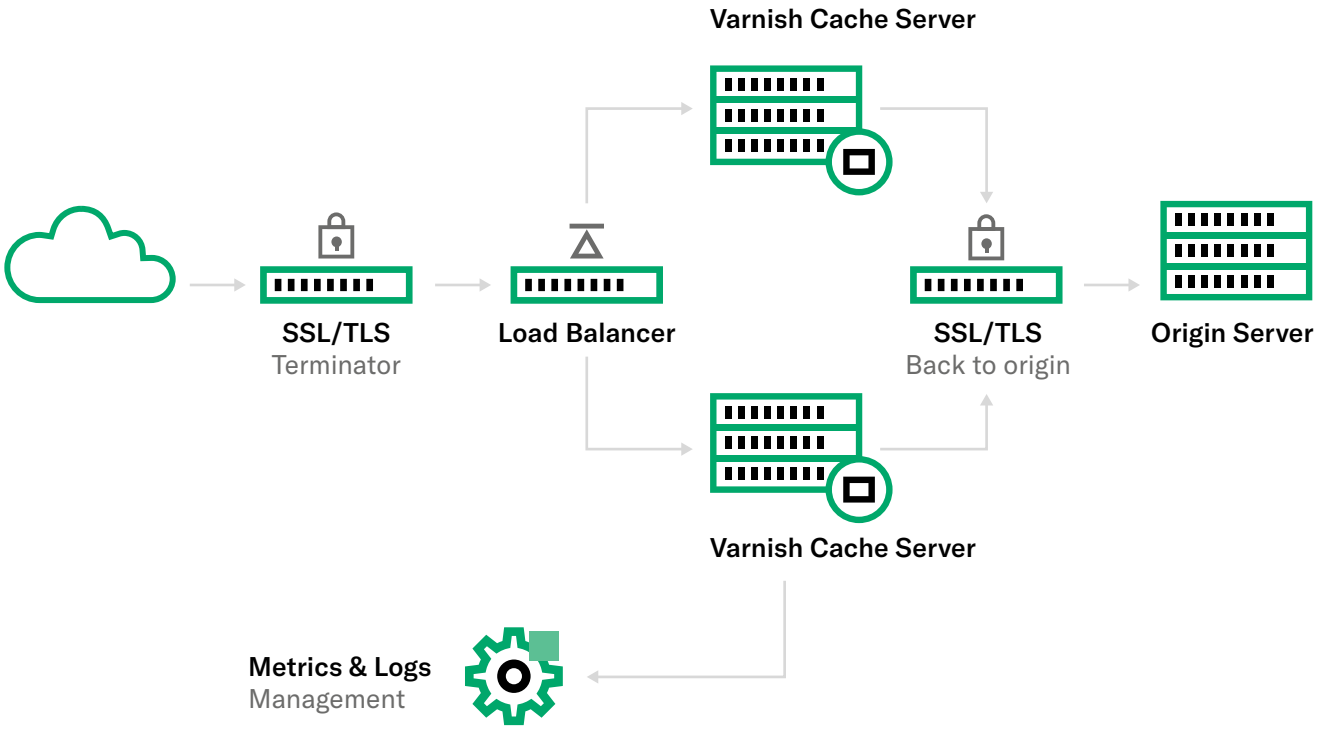
Congratulations! Varnish Cache is now installed...but how do you know it's working? Is it caching content effectively?

Read on for the more detailed version.

Longer Steps To Installing A Full Varnish Cache Solution:

All high performance Varnish Cache implementations need to start with a quick architectural plan in order to understand how the solution fits into your environment.

A successful implementation may look like the below:



Below are the components needed for a successful Varnish Cache deployment:

HTTPS/SSL management: The standard practice to support HTTPS is to implement a layer in front of your Varnish Cache servers that performs HTTPS negotiation and then passes traffic through to your Varnish Cache layer (Commonly with the addition of an X-Forwarded-Proto request header set to "https"). A common solution to resolve the lack of SSL is to deploy an nginx tier in front of your Varnish Cache server. Here is some sample nginx/SSL configuration.

Load Balancing: Installing Varnish Cache on a new server is great but this implements a single point of failure in your infrastructure. Options available to resolve this are:

1. Deploy Varnish Cache on each of your web applications servers: This can be a short term win as it doesn't require additional hardware and as long as you have multiple application servers you then have redundancy across your Varnish Cache implementation. This approach falls short when you have more than two web servers as your cache is split between each web server and cache hit rate will fall off dramatically as the web server count increases.

2. Implement a load balancer in front of your redundant Varnish Cache servers: This adds some complexity but is a scalable way to implement traffic balancing. Here is an example architecture link with code samples to deploy a load balanced environment.

HTTPS / SSL to the Origin server: It's important to consider how traffic from your Varnish Cache tier passes back to your origin servers. Compliance and general security best practice mean that you should pass HTTPS traffic back to the origin application over HTTPS.

The best approach to resolving this is to have your front end HTTPS/SSL termination add an X-Forwarded-Proto request header containing the value "https". You can then add another nginx tier as a Varnish Cache back end which will have a proxy pass statement that will use the X-Forwarded-Proto value to determine whether to connect to the origin servers via HTTP or HTTPS.

Metrics: We've already discussed the importance of good metrics to assess Varnish Cache's performance in relation to your application. To recap, you'll want to monitor:

Individual requests: The best request specific debugger that's quickly available on all Varnish Cache implementations is varnishlog. This tool allows you to send individual requests and analyze how each request has been handled. varnishtop is a second command available that shows continuously updated counters of values that you specify.

Cache hit rates: Tracking Varnish Cache performance on an ongoing basis is critical to understanding whether the solution is performing effectively. varnishstat is an option for short term review as its run at the command line. Here is an article describing how to use varnishstat.

The most robust method for metric management is to send Varnish Cache access log data to a metric/log management product. This is generally a requirement for an enterprise level deployment of Varnish Cache and can have its own cost to implement if nothing else is already available. In order to output access logs in an Apache style format you need to use the varnishncsa command. Here is a guide to setting up the varnishncsa command.

The Quick And Painless Way To Get A Full Varnish Cache Installation:

If you are looking for all the above tools but don't have the time or resources to set them up yourself, look to a solution such as Section which already includes robust metrics and logs, SSL termination, a local environment for testing, and VCL suggestions. With Section you'll also get access to our team of Varnish Cache experts who are always available to help you get the best possible performance out of Varnish Cache.

Conclusion

Varnish Cache is a powerful tool that can dramatically speed up application delivery while reducing load on your origin servers, but it is often perceived to be difficult to set up and configure correctly. The reality is, with a basic understanding of VCL and the right tools to monitor Varnish Cache, you can get a much better performance result than with any other caching solution available on the market. We hope this guide has served as a good introduction to Varnish Cache and VCL, and as always if you have questions about Varnish Cache or want to try Section's Edge Compute Platform with Varnish Cache please contact us at contact@section.io.

Additional References

https://www.varnish-software.com/wiki/content/tutorials/varnish/sample_vclTemplate.html#cookie-manipulation
http://highscalability.com/blog/2015/5/6/varnish-goes-upstack-with-varnish-modules-and-varnish-config.html
http://book.varnish-software.com/4.0/chapters/VCL_Basics.html#vcl-vcl-backend-response
http://www.harecoded.com/varnish-vcl-delete-all-cookies-and-other-magic-2175580
https://www.varnish-cache.org/docs/trunk/reference/varnishncsa.html
http://book.varnish-software.com/4.0/chapters/VCL_Subroutines.html
https://varnish-cache.org/docs/4.0/users-guide/vcl-built-in-subs.html
https://spin.atomicobject.com/2013/01/16/speed-up-website-varnish/
https://www.datadoghq.com/blog/how-to-collect-varnish-metrics/
https://varnish-cache.org/trac/wiki/VCLExampleLongerCaching
https://www.section.io/blog/varnish-install-quick-and-detailed/
https://community.section.io/t/a-novel-way-to-cache-html/79
https://www.varnish-software.com/products/varnish-extend/
https://www.varnish-cache.org/trac/wiki/VCLExampleGrace
https://www.section.io/blog/true-costs-of-running-varnish/
http://book.varnish-software.com/3.0/VCL_Basics.html
https://www.fastly.com/blog/vcl-cookie-monster/
https://www.w3schools.com/xml/ajax_intro.asp
https://varnish-cache.org/docs/index.html
https://www.varnish-cache.org/vmods/