Tue, March 14, 2017

Web Cache - Everything you need to know

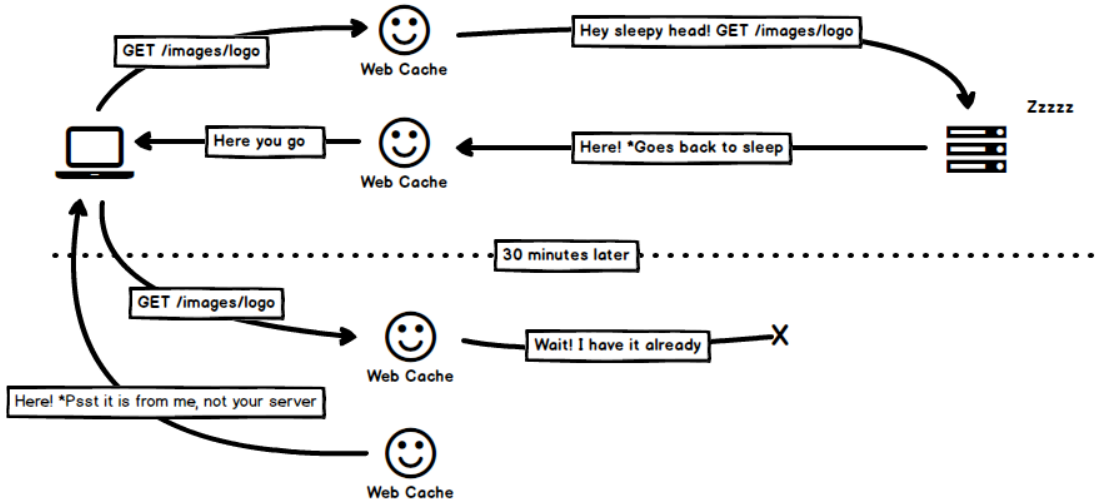
In one of the previous posts, I discussed about HTTP and where it stands at this point. This is one is going to be specifically about the caching.

As users, we easily get frustrated by the buffering videos, the images that take seconds to load, pages that got stuck because the content is being loaded. Loading the resources from some cache is much faster than fetching the same from the originating server. It reduces latency, speeds up the loading of resources, decreases the load on server, cuts down the bandwidth costs etc.

Introduction

What is web cache? It is something that sits somewhere between the client and the server, continuously looking at the requests and their responses, looking for any responses that can be cached. So

that there is less time consumed when the same request is made again.



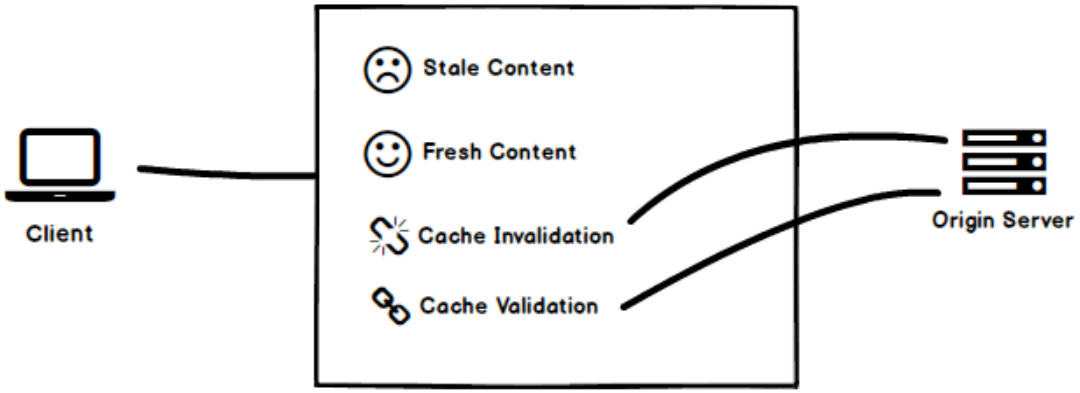
Note that this image is just to give you an idea. Depending upon the type of cache, the place where it is implemented could vary. More on this later.

Before we get into further details, let me give you an overview of the terms that will be used, further in the article

- **Client** could be your browser or any application requesting the server for some resource
- **Origin Server**, the source of truth, houses all the content required by the client and is responsible for fulfilling the client

requests.

- **Stale Content** is the cached but expired content
- **Fresh Content** is the content available in cache that hasn't expired yet
- **Cache Validation** is the process of contacting the server to check the validity of the cached content and get it updated for when it is going to expire
- **Cache Invalidation** is the process of removing any stale content available in the cache

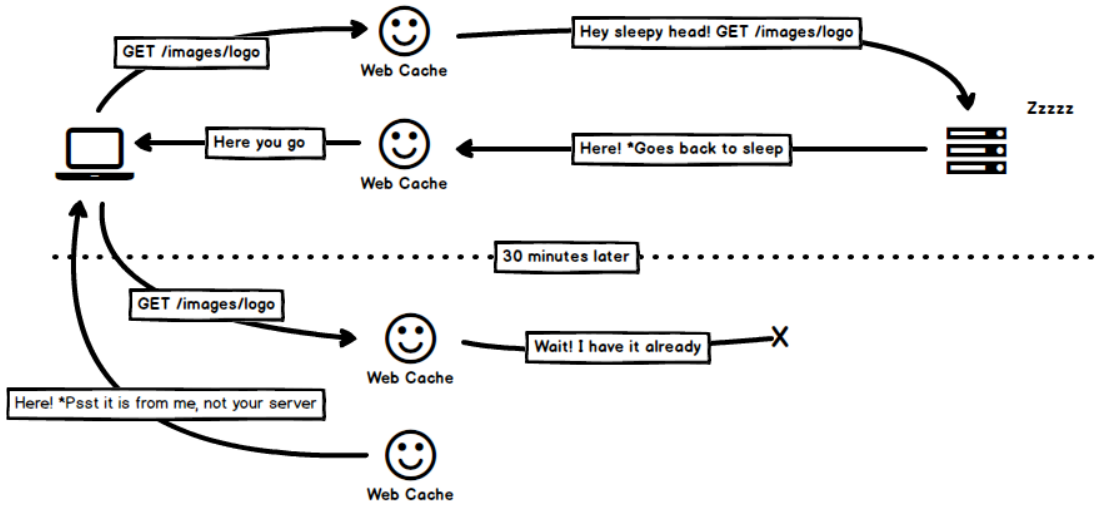


Caching Locations

Web cache can be shared or private depending upon the location where it exists. Below is the list of caching locations

Browser Cache

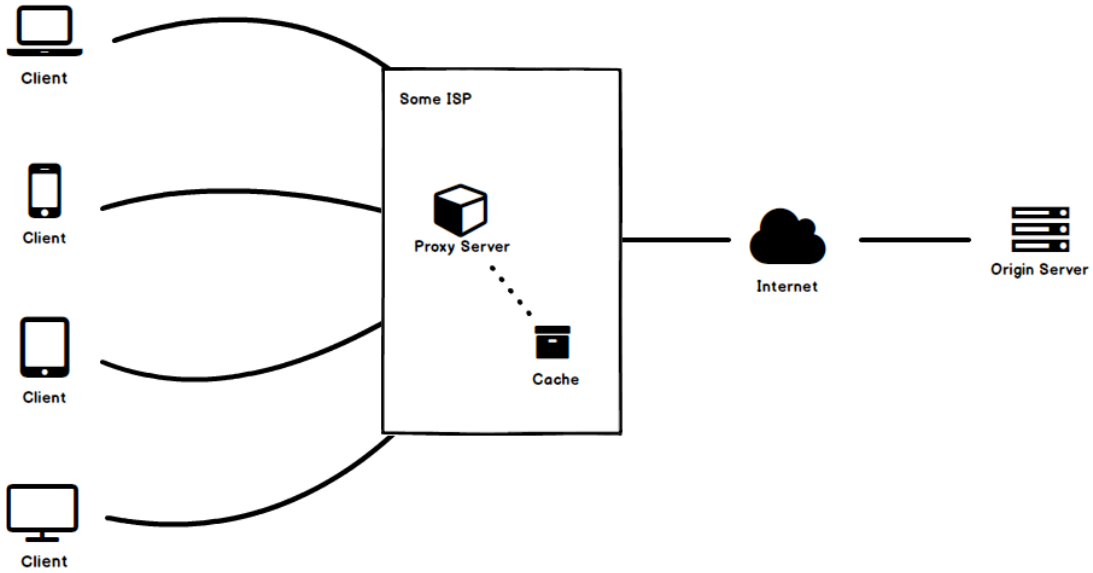
You might have noticed that when you click the back button in your browser it takes less time to load the page than the time that it took during the first load; this is the browser cache in play. Browser cache is the most common location for caching and browsers usually reserve some space for it.



A browser cache is limited to just one user and unlike other caches, it can store the “private” responses. More on it later.

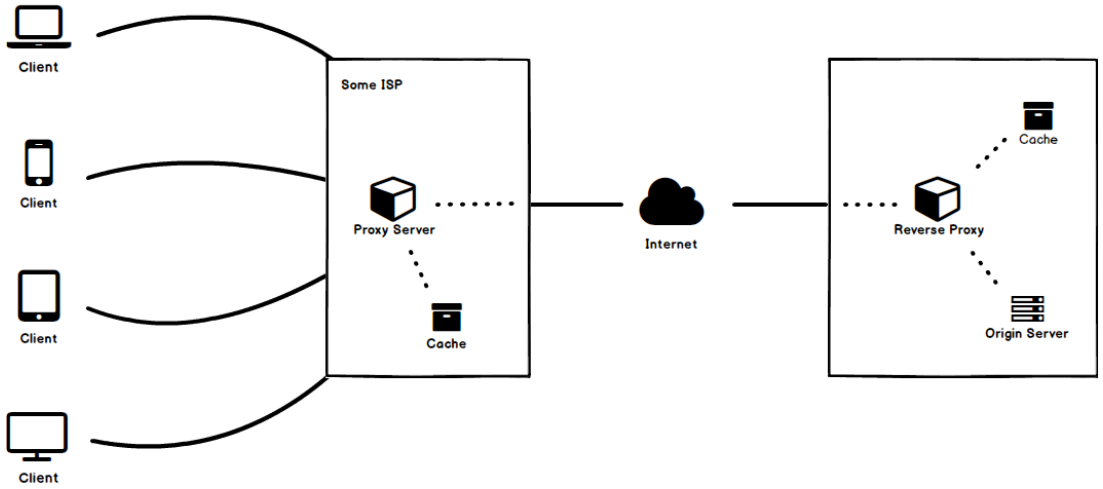
Proxy Cache

Unlike browser cache which serves a single user, proxy caches may serve hundreds of different users accessing the same content. They are usually implemented on a broader level by ISPs or any other independent entities for example.



Reverse Proxy Cache

Reverse proxy cache or surrogate cache is implemented close to the origin servers in order to reduce the load on server. Unlike proxy caches which are implemented by ISPs etc to reduce the bandwidth usage in a network, surrogates or reverse proxy caches are implemented near to the origin servers by the server administrators to reduce the load on server.



Although you can control the reverse proxy caches (since it is implemented by you on your server) you can not avoid or control browser and proxy caches. And if your website is not configured to use these caches properly, it will still be cached using whatever the defaults are set on these caches.

Caching Headers

So, how do we control the web cache? Whenever the server emits some response, it is accompanied with some HTTP headers to guide the caches whether and how to cache this response. Content provider is the one that has to make sure to return proper HTTP headers to force the caches on how to cache the content.

Expires

Before HTTP/1.1 and introduction of `Cache-Control`, there was `Expires` header which is simply a timestamp telling the caches how long should some content be considered fresh. Possible value to this header is absolute expiry date; where date has to be in GMT. Below is the sample header

```
Expires: Mon, 13 Mar 2017 12:22:00 GMT
```

It should be noted that the date cannot be more than a year and if the date format is wrong, content will be considered stale. Also, the clock on cache has to be in sync with the clock on server, otherwise the desired results might not be achieved.

Although, `Expires` header is still valid and is supported widely by the caches, preference should be given to HTTP/1.1 successor of it i.e. `Cache-Control`.

Pragma

Another one from the old, pre HTTP/1.1 days, is `Pragma`. Everything that it could do is now possible using the cache-control header given below. However, one thing I would like to point out about it is, you might see `Pragma: no-cache` being used here and there in hopes of stopping the response from being cached. It might not necessarily work; as HTTP specification discusses it in the

request headers and there is no mention of it in the response headers. Rather `Cache-Control` header should be used to control the caching.

Cache-Control

Cache-Control specifies how long and in what manner should the content be cached. This family of headers was introduced in HTTP/1.1 to overcome the limitations of the `Expires` header.

Value for the `Cache-Control` header is composite i.e. it can have multiple directive/values. Let's look at the possible values that this header may contain.

private

Setting the cache to `private` means that the content will not be cached in any of the proxies and it will only be cached by the client (i.e. browser)

```
Cache-Control: private
```

Having said that, don't let it fool you in to thinking that setting this header will make your data any secure; you still have to use SSL for that purpose.

public

If set to `public`, apart from being cached by the client, it can also be cached by the proxies; serving many other users

```
Cache-Control: public
```

no-store

`no-store` specifies that the content is not to be cached by any of the caches

```
Cache-Control: no-store
```

no-cache

`no-cache` indicates that the cache can be maintained but the cached content is to be re-validated (using `ETag` for example) from the server before being served. That is, there is still a request to server but for validation and not to download the cached content.

```
Cache-Control: max-age=3600, no-cache, public
```

max-age: seconds

`max-age` specifies the number of seconds for which the content will be cached. For example, if the `cache-control` looks like below:

```
Cache-Control: max-age=3600, public
```

it would mean that the content is publicly cacheable and will be considered stale after 60 seconds

s-maxage: seconds

`s-maxage` here `s-` prefix stands for shared. This directive specifically targets the shared caches. Like `max-age` it also gets the number of seconds for which something is to be cached. If present, it will override `max-age` and `expires` headers for shared caching.

```
Cache-Control: s-maxage=3600, public
```

must-revalidate

`must-revalidate` it might happen sometimes that if you have network problems and the content cannot be retrieved from the

server, browser may serve stale content without validation. `must-revalidate` avoids that. If this directive is present, it means that stale content cannot be served in any case and the data must be re-validated from the server before serving.

```
Cache-Control: max-age=3600, public, must-revalida
```

proxy-revalidate

`proxy-revalidate` is similar to `must-revalidate` but it specifies the same for shared or proxy caches. In other words `proxy-revalidate` is to `must-revalidate` as `s-maxage` is to `max-age`. But why did they not call it `s-revalidate`?. I have no idea why, if you have any clue please leave a comment below.

Mixing Values

You can combine these directives in different ways to achieve different caching behaviors, however `no-cache/no-store` and `public/private` are mutually exclusive.

If you specify both `no-store` and `no-cache`, `no-store` will be given precedence over `no-cache`.

```
; If specified both
Cache-Control: no-store, no-cache

; Below will be considered
Cache-Control: no-store
```

For `private/public`, for any unauthenticated requests cache is considered `public` and for any authenticated ones cache is considered `private`.

Validators

Up until now we only discussed how the content is cached and how long the cached content is to be considered fresh but we did not discuss how the client does the validation from the server. Below we discuss the headers used for this purpose.

ETag

Etag or "entity tag" was introduced in HTTP/1.1 specs. Etag is just a unique identifier that the server attaches with some resource. This ETag is later on used by the client to make conditional HTTP requests stating `"give me this resource if ETag is not same as the ETag that I ha` and the content is downloaded only if the etags do not match.

Method by which ETag is generated is not specified in the HTTP docs and usually some collision-resistant hash function is used to assign etags to each version of a resource. There could be two types of etags i.e. strong and weak

```
ETag: "j82j8232ha7sdh0q2882" - Strong Etag
ETag: W/"j82j8232ha7sdh0q2882" - Weak Etag (prefix
```

A strong validating ETag means that two resources are **exactly** same and there is no difference between them at all. While a weak ETag means that two resources are although not strictly same but could be considered same. Weak etags might be useful for dynamic content, for example.

Now you know what etags are but how does the browser make this request? by making a request to server while sending the available Etag in `If-None-Match` header.

Consider the scenario, you opened a web page which loaded a logo image with caching period of 60 seconds and ETag of `abc123xyz`. After about 30 minutes you reload the page, browser will notice that the logo which was fresh for 60 seconds is now stale; it will trigger a request to server, sending the ETag of the stale logo image in `if-none-match` header

```
If-None-Match: "abc123xyz"
```

Server will then compare this ETag with the ETag of the current version of resource. If both etags are matched, server will send back the response of `304 Not Modified` which will tell the client that the copy that it has is still good and it will be considered fresh for another 60 seconds. If both the etags do not match i.e. the logo has likely changed and client will be sent the new logo which it will use to replace the stale logo that it has.

Last-Modified

Server might include the `Last-Modified` header indicating the date and time at which some content was last modified on.

```
Last-Modified: Wed, 15 Mar 2017 12:30:26 GMT
```

When the content gets stale, client will make a conditional request including the last modified date that it has inside the header called `If-Modified-Since` to server to get the updated `Last-Modified` date; if it matches the date that the client has, `Last-Modified` date for the content is updated to be considered fresh for another `n` seconds. If the received `Last-Modified` date does not match the one that the client

has, content is reloaded from the server and replaced with the content that client has.

```
If-Modified-Since: Wed, 15 Mar 2017 12:30:26 GMT
```

You might be questioning now, what if the cached content has both the `Last-Modified` and `ETag` assigned to it? Well, in that case both are to be used i.e. there will not be any re-downloading of the resource if and only if `ETag` matches the newly retrieved one and so does the `Last-Modified` date. If either the `ETag` does not match or the `Last-Modified` is greater than the one from the server, content has to be downloaded again.

Where do I start?

Now that we have got *everything* covered, let us put everything in perspective and see how you can use this information.

Utilizing Server

Before we get into the possible caching strategies , let me add the fact that most of the servers including Apache and Nginx allow you to implement your caching policy through the server so that you don't have to juggle with headers in your code.

For example, if you are using Apache and you have your static content placed at `/static`, you can put below `.htaccess` file in the directory to make all the content in it be cached for an year using below

```
# Cache everything for an year
Header set Cache-Control "max-age=31536000, public
```

You can further use `filesMatch` directive to add conditionals and use different caching strategy for different kinds of files e.g.

```
# Cache any images for one year
<filesMatch ".(png|jpg|jpeg|gif)$">
    Header set Cache-Control "max-age=31536000, pu
</filesMatch>

# Cache any CSS and JS files for a month
<filesMatch ".(css|js)$">
    Header set Cache-Control "max-age=2628000, pub
</filesMatch>
```

Or if you don't want to use the `.htaccess` file you can modify Apache's configuration file `http.conf`. Same goes for Nginx, you can add the caching information in the location or server block.

Caching Recommendations

There is no golden rule or set standards about how your caching policy should look like, each of the application is different and you have to look and find what suits your application the best. However, just to give you a rough idea

- You can have aggressive caching (e.g. cache for an year) on any static content and use fingerprinted filenames (e.g. `style.ju2i90.css`) so that the cache is automatically rejected whenever the files are updated. Also it should be noted that you should not cross the upper limit of one year as it might not be honored
- Look and decide do you even need caching for any dynamic content, if yes how long it should be. For example, in case of some RSS feed of a blog there could be the caching of a few hours but there couldn't be any caching for inventory items in an ERP.
- Always add the validators (preferably ETags) in your response.
- Pay attention while choosing the visibility (private or public) of the cached content. Make sure that you do not accidentally cache any user-specific or sensitive content in any public proxies. When in doubt, do not use cache at all.
- Separate the content that changes often from the content that doesn't change that often (e.g. in javascript bundles) so that when it is updated it doesn't need to make the whole cached content stale.

- Test and monitor the caching headers being served by your site. You can use the browser console or `curl -I http://some-url.com` for that purpose.

And that about wraps it up. If you have any comments or feedback, feel free to leave a comment below.

If you find any typos or blatant lies, please contribute by updating the article

👏 Don't like emails? Me neither. **Follow me on twitter** for the updates

0 Comments Kamran Ahmed 1 Login

Recommend 1 Tweet Share Sort by Best

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

Subscribe Add Disqus to your siteAdd DisqusAdd

© 2019 Kamran Ahmed, unless otherwise stated.