

*Cachez cette donnée
que je ne saurais voir*



<https://stephanedaviet.github.io/cachez-cette-donnee>





Rendons à Molière ce qui lui appartient

« **Couvrez** ce sein, que je ne saurais voir.
Par de pareils objets les âmes sont blessées,
Et cela fait venir de coupables pensées. »

— *Le Tartuffe ou l'Imposteur* (1669),
Molière

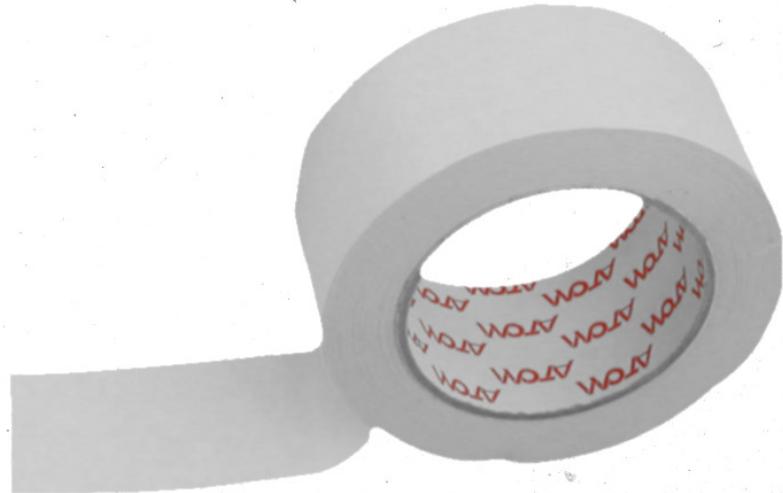
Qu'est-ce qu'un cache ?

« En informatique, un cache est un **composant matériel ou logiciel** qui **stocke de la donnée** de sorte à ce que de **futures requêtes** pour cette donnée puissent être **honorées plus rapidement**. Les données stockées en cache peuvent être le résultat d'un calcul antérieur ou la copie d'une donnée stockée ailleurs. »

— traduit depuis *Wikipédia – Cache (computing)*

Ceci n'est pas un cache

- ❖ Un **backup** : 2 types de stockage différents, l'un volatile et très performant, l'autre pérenne et généralement lent : ne pas utiliser un cache comme backup 😊.
- ❖ Une **mémoire de travail** : Une mémoire de travail est intègre et son cycle de vie est maîtrisé par le code qui la manipule, ce n'est pas le cas d'un cache par essence volatile.

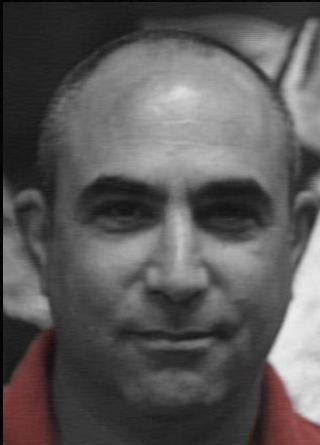


Ceci n'est pas un cache.

Les bénéfices d'un cache

- ❖ Augmentation de la **performance ↗** pour l'utilisateur,
- ❖ Augmentation de la **résilience ↗**,
- ❖ Diminution de la **charge réseau ↘**,
- ❖ Diminution de la **charge serveur ↘**.





A wise man!

« There are only two hard things in
Computer Science: **cache invalidation**
and naming things. »

— Phil Karlton

Deux ou trois choses dont il faut se méfier

- ❖ La **taille** de certains caches peut devenir un souci.
- ❖ Le **coût de démarrage à froid** (sans cache) ne doit pas être minoré, toujours tester sans (e.g. extension Lighthouse).
- ❖ L'**éiction/invalidation** du cache peut être problématique si pas bien prévue au départ !
- ❖ Le **temps d'interprétation** d'une ressource JavaScript, même récupérée depuis le cache client, peut être long.
- ❖ Les **caches en chaîne** qui peuvent compliquer le diagnostic et démultiplier les problèmes de purge.

Le cache, un truc vieux comme l'informatique

Quand cacher & que mettre en cache ?

- ❖ **Cacher autant que possible.** Selon que la réponse est statique, à cacher sans vergogne, ou dynamique, à cacher avec plus de précautions.
- ❖ **Cacher aussi longtemps que possible.** Les ressources statiques versionnées (e.g. JavaScript) peuvent être cachées indéfiniment.
- ❖ **Cacher aussi prêt des utilisateurs que possible.** Cacher à proximité de l'utilisateur (CDN ou idéalement cache client) réduit la latence.

Source : *Web Almanac par HTTP Archive (2019) – Part IV Chapter 16: Caching*

Des stats !

« Parmi toutes les requêtes HTTP, **80% des réponses** sont considérées comme pouvant être cachées »

— traduit depuis *Web Almanac par HTTP Archive (2019)*
Part IV Chapter 16: Caching

Levons le voile sur le cache HTTP

Un principe **éprouvé et très bien pensé** entre client & serveur, sur la base d'échanges d'*headers HTTP*.

Les headers qui mènent la danse :

- ❖ Expires & Cache-Control,
- ❖ Last-Modified & If-Modified-Since,
- ❖ ETag & If-None-Match,
- ❖ Vary & Clear-Site-Data.

Pour plus de détails : [MDN – HTTP Caching](#) et tous les détails [IETF – RFC2616 HTTP/1.1 – section 13](#) & [IETF – RFC7234 HTTP/1.1: Caching](#).

Expires & Cache-Control

❖ **Expires** (*HTTP/1.0, 1996*) : Permet d'indiquer à partir de quand une ressource est à considérée comme expirée (ou *stale* pour rassise), e.g.
`Expires: Thu, 21 december 2012 00:00:00 GMT.`

❖ **Cache-Control** (*HTTP/1.1, 1997, surclasse Expires si les deux sont précisés*) :

« Le header HTTP Cache-Control permet – que ce soit pour les requêtes ou pour les réponses – de contrôler la mise en cache dans les navigateurs et dans les caches partagés (e.g. Proxies, CDNs). »

— traduit depuis [MDN - Web/HTTP/Headers/Cache-Control](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control)

Exemple : `cache-control: public, max-age=43200, must-revalidate`.

Vous saurez tout (ou presque) sur Cache-Control

- ❖ `max-age` & `s-maxage` : temps en secondes pendant lequel la ressource peut être considérée comme fraîche (*fresh*) respectivement par le client ou par un cache partagé (e.g. `max-age=86400`),
- ❖ `no-cache` : la ressource peut être mise en cache mais doit systématiquement être revalidée auprès du serveur avant usage, même déconnecté,
- ❖ `must-revalidate` & `proxy-revalidate` : la ressource peut être mise en cache et doit être revalidée auprès du serveur quand elle est en état *stale*, respectivement par le client ou par un cache partagé,
- ❖ `no-store` : la ressource ne doit pas être mise en cache,
- ❖ `private` : la ressource ne doit pas être mise en cache par des caches publics (caches partagés), utile notamment quand elle est spécifique à un utilisateur, `public` par défaut sauf si header `Authorization` présent.

Et quelques autres, voir [MDN – Web/HTTP/Headers/Cache-Control](#).

Quelques exemples d'usage

- ❖ Pour garder en cache pendant un an une ressource JavaScript versionnée, e.g. `react.18.1.0.production.min.js` (*cache busting*) :
`Cache-Control: max-age=31536000, immutable,`
- ❖ Pour permettre la mise en cache de `index.html` mais obliger la revalidation auprès du serveur : `Cache-Control: no-cache,`
- ❖ Pour empêcher la mise en cache de ressources dynamiques, e.g. une métrique mesurée en continu : `Cache-Control: no-store,`
- ❖ Pour permettre la mise en cache d'informations spécifiques à un utilisateur, mais uniquement dans des caches non-partagés : `Cache-Control: private, max-age: 3600, must-revalidate.`

Last-Modified & If-Modified-Since

Premier accès : t_0



Last-Modified = date de dernière modification de la ressource.

Last-Modified & If-Modified-Since

$t_0 + 30 \text{ min}$



Pas de **revalidation** si la ressource n'est pas expirée (ressource *fresh*, i.e. : $t_0 + 30\text{min} < \text{max-age}$).

Last-Modified & If-Modified-Since

$t_0 + 90 \text{ min}$



Statut 304 si $\text{Last-Modified} \leq \text{If-Modified-Since}$.

Toujours renvoyer les headers de cache, notamment **Cache-Control**, même avec un **304**.

Last-Modified & If-Modified-Since

$t_0 + 96 \text{ min}$



200 si **Last-Modified > If-Modified-Since** & **nouvelle ressource**.

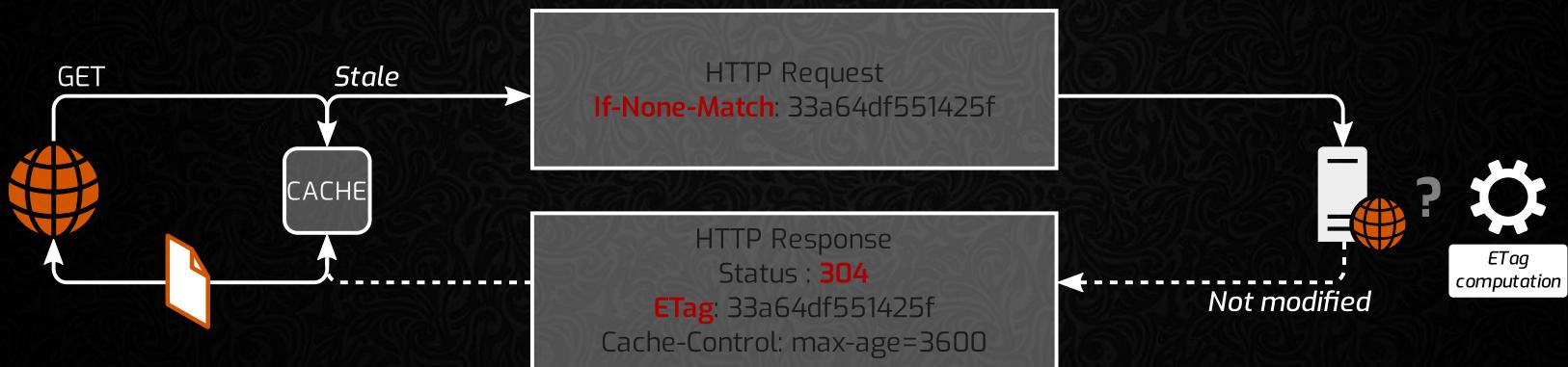
Nouveaux **max-age** et **Last-Modified**.

ETag & If-None-Match

ETag ~ Somme de hashage (*digest/checksum*) de la ressource.

« Typiquement, une valeur d'ETag peut être un **hash du contenu**, un hash du dernier timestamp de modification ou juste un numéro de révision. »

— traduit depuis [MDN – Web/HTTP/Headers/Etag](#)



If-None-Match prioritaire à If-Modified-Since si les 2 sont présents.

Plus coûteux à produire (mais peut être stocké), mais **plus fiable** (e.g. update/rollback de la ressource).

Vary

`Vary` décrit les parties de la requête autres que la méthode et l'URL qui ont une incidence sur le contenu de la réponse produite. Concrètement c'est une liste d'en-têtes HTTP.

- ❖ `Vary: *` : tout influe sur la réponse, implique que celle-ci ne peut être mise en cache,
- ❖ `Vary: Accept-Encoding, Accept, X-API-Version` : le format de réponse demandé et la version de l'API influent sur la réponse.

[Smashing Magazine – Understanding The Vary Header](#) : article sur les subtilités d'interprétation de `Vary` dans les navigateurs.

Clear-Site-Data

Clear-Site-Data permet d'indiquer au client les types de données associées au site courant pour lesquelles il convient de purger le cache. Ne fonctionne qu'en HTTPS. Peut sauver la mise à l'occasion 😊...

- ❖ `Clear-Site-Data: "cache", "cookies", "storage", "executionContexts"` (~ `Clear-Site-Data: "*"`) : vider tous les caches associés au site.

Vite fait bien fait côté serveur avec Spring Web

```
1  public <T> ResponseEntity<T> checkCacheOrAnswer(Class<? extends Object> objectType,
2      String objectName,
3      WebRequest webRequest,
4      Supplier<T> noCacheSupplier) {
5      long lastModified = getLastModifiedFor(objectType, objectName);
6      String eTag = getETagFor(objectType, objectName);
7
8      if (webRequest.checkNotModified(lastModified, eTag)) {
9          return ResponseEntity
10             .status(HttpStatus.NOT_MODIFIED)
11             .cacheControl(CacheControl.maxAge(0, TimeUnit.SECONDS).cachePrivate().mustRevalidate())
12             .lastModified(lastModified)
13             .eTag(eTag)
14             .build();
15     }
16     return ResponseEntity.ok()
17        .cacheControl(CacheControl.maxAge(0, TimeUnit.SECONDS).cachePrivate().mustRevalidate())
18        .lastModified(lastModified)
19        .eTag(eTag)
20        .body(noCacheSupplier.get());
21 }
```


Un coup d'œil du côté des échanges inter-services

Caractéristiques des architectures distribuées et modulaires (dites **micro-services**) :

- ❖ protocole **HTTP** (généralement),
- ❖ **gros volume** d'échange de données,
- ❖ données dynamiques pour la plupart (car calculées), mais dont une bonne partie **stables sur une plage de temps plus ou moins longue**.

N'attendez plus, **mettez en cache** vos échanges inter-services !

Seul bémol : maintient d'un état côté service dans une architecture généralement *stateless*.

Côté clients Java : quid des rfc2616/7234 ?

Client	Support	Commentaire	Support induit
Java HttpURLConnection	Non 😞		
Java HttpClient	Non 😞		
Apache HttpClient	Oui 😊	via <code>CachingHttpClientBuilder</code>	Spring Rest Template , RestEasy (Quarkus), Netflix Ribbon
Apache HttpAsyncClient	Oui 😊	via <code>CachingHttpAsyncClientBuilder</code>	
OkHttp	Oui 😊	via <code>OkHttpClient.Builder().cache()</code>	Retrofit (Android)
Vert.x Web Client	Oui 😊	via <code>CachingWebClient.create</code>	
Spring WebClient	Non 😞	<i>Rien trouvé de probant à ce sujet</i>	
Jetty HttpClient	Non 😞		
Akka HTTP	Non 😞	<i>Supporte un cache basique, non RFC</i>	
Netty (Http2Client)	Non 😞	<i>Rien trouvé de probant à ce sujet</i>	

Et pour les autres langages/écosystèmes ?

Plutôt simple à mettre en œuvre et globalement supporté côté serveur. **Côté client, c'est une autre histoire :**

Et pour les autres protocoles ?

Malheureusement, pas grand chose, voir rien...

Envie d'un petit tour en sidecar ?

Sidecar container avec Kubernetes (également nommé « *companion* ») : permet, au sein d'un *pod* Kubernetes, d'adjoindre à un conteneur existant un autre conteneur en charge d'assurer des **fonctions transverses** (monitoring, sécu, cache, etc.).

Permet d'**instrumenter simplement une application sans la modifier** pour lui apporter un cache en proximité.

Quelques solutions ou pistes de solutions existantes :

- ❖ explication du principe : [Hazelcast Sidecar Container Pattern](#),
- ❖ sur la base d'Istio (Envoy++) : [Trendyol/sidecache](#),
- ❖ le plus prometteur car intégré au cœur d'Envoy : [Envoy Cache Filter \(pas encore production ready\)](#).

Pour combler les longues soirées d'hiver

Un peu de littérature

*Pas de cachotteries entre nous,
vos questions ?*



Concocté avec  avec une base de  , un trait de  , un soupçon de  , enfin un nappage de  , le tout sous  .