ENI Service - WordPress - PHP

{.marp-bg-img}

Paul Schuhmacher

Durée : 28h

Novembre 2025

Condensé (dernier jour) : Thème parent-enfant, optimisations, déploiement et configuration, misc

Thèmes enfants

> Depuis WordPress 2.7

- Créer un **thème enfant** pour **étendre** un thème existant (*starter theme*, thème officiel, etc.);
- Possible pour **tous les types de thèmes** :
  - **Classiques**, basés sur des templates,
  - **Mixtes**, basés sur des templates et des blocs,
  - **FSE**, basés entièrement sur des blocs et *l'éditeur de site* Gutenberg.

> L'extension de thème s'arrête à la relation parent-enfant. Pas de troisième niveau (thème petit-enfant impossible)

Avantages à utiliser un thème enfant

- Garder vos **personnalisations séparées** du thème parent ;
- Permettre la **mise à jour du thème parent sans perdre vos modifications**. Si vous développez directement dans les fichiers du thème parent **vous perdrez tout à la mise à jour** du thème !
- **Gagner du temps** en n’écrivant que le code nécessaire (fonctionnalités, templates, assets CSS/JS, etc.);
- Se **créer** (ou utiliser) un **starter theme** bien *configuré, personnalisé* (que l'on peut mettre à jour dans le temps) et **l'utiliser comme parent** pour tous les sites que l'on développe.

Créer un thème enfant

Renseigner la directive `Template:` dans le fichier `style.css` du thème enfant :

```
/**
 * Theme Name: Mon Theme
 * Template:   twentytwentyfour
 * ...other header fields
 */
```

> `Mon Theme` étend le thème `twentytwentyfour`

Activer le thème comme n'importe quel thème.

Développer le thème enfant : charger le CSS du thème parent

Si un thème *classique*, **charger** le CSS du thème parent avec la fonction `get_parent_theme_file_uri()` :

```
add_action( 'wp_enqueue_scripts', 'mon_theme_enqueue_styles' );

function mon_theme_enqueue_styles() {
//Charger la feuille de style du thème parent
 wp_enqueue_style(
  'parent-style',
  get_parent_theme_file_uri( 'style.css' )
 );
 //Charger la feuille de style du thème enfant (override parent)
 wp_enqueue_style(
  'mon-theme-style',
  get_stylesheet_uri()
 );
}
```

Templates, parts et patterns

Dans le thème enfant, **créer uniquement les templates qui doivent personnaliser** ceux du thème parent. Pour cela, il suffit de créer le template PHP correspondant dans le thème enfant.

Lorsque WordPress va choisir le template à utiliser, il va d'abord scanner le répertoire du thème enfant. **S'il y trouve le template il l'utilise**, sinon il utilise celui du parent (*fallback*).

On peut ainsi surcharger (*override*) uniquement un ou plusieurs templates dans le thème enfant, tout en bénéficiant des templates du parent.

Le fichier `functions.php`

À l'activation d'un thème, WordPress cherche un fichier `functions.php` à la racine du thème. S'il le trouve, il l'**inclut automatiquement** (`require`).

Le fichier `functions.php` du thème enfant **ne remplace pas** (*override*) celui du thème parent. **Les deux fichiers sont chargés**.
Le fichier `functions.php` du **thème enfant est chargé en premier**, celui du thème parent en second.

```
themes/
    parent/
        functions.php    <= chargé en second
        style.css
        index.php
    enfant/
        functions.php    <= chargé en premier
        style.css
        index.php
```

Inclure un fichier PHP

Pour **construire les bons paths**, par exemple lors de l'inclusion d'un fichier source PHP dans un template ou functions.php, **utiliser** la fonction `get_theme_file_path()` :

```
//Par ex, dans functions.php du thème enfant
require_once get_theme_file_path( 'inc/functions-helpers.php' );
```

Développer des bons sites WordPress fiables et performants

- **Bien développer et suivre les usages** du framework ;
- Mesurer/Profiler ;
- Optimiser au besoin.

Valider le markup HTML auprès du W3C

Utiliser le service du W3C pour valider le markup de vos pages web. Un **markup valide est à la base** d'un rendu de pages web *fiable*, *stable* et *accessible*.

Un markup *cassé* (*c'est mal !*) :

- Peut ralentir le rendu par le navigateur (corrections du DOM, recalcul de layouts) ;
- SEO : Moteurs de recherche mettent en valeur les URL fournissant du markup valide ;
- Peut briser l'accessibilité des pages web.

Suivre les recommandations de la communauté WordPress

Consulter les recommandations sur le site officiel :

- Optimisation des sites WordPress ;
- How to improve performance on WordPress.

Bien utiliser WordPress 1/2

- **Images de tailles réduites** :
  - Pour le web, **inutile** d'avoir des images en très haute définition (*qualité print*), utiliser la **compression** ! *En règle générale, les grandes images de haute qualité doivent être conservées entre **100K et 60K** (< 1 Mo). Les images plus petites devraient être plus proches de **30K** et moins*.
  - Toujours **utiliser un plugin pour compresser automatiquement les images**, ne pas compter sur la bonne volonté (même sincère) des admins ou éditeurs·ices !
  - Privilégier le format webP, réduit la taille de ~40% par rapport à JPEG ou PNG, sans perte de qualité perceptible à l'oeil ;
  - **Définir des formats d'image WordPress** pour produire autant de versions d'image adaptées à chaque page, composant et usage (miniature, etc.),
  - Enfin, ou plutôt, pour *commencer* : avez-vous *vraiment* besoin de *cette* image ? **Réduire le nombre d'images au strict nécessaire !**
- **Assets compressés et/ou minifiés** (images et logos, vidéos, CSS, JS, fonts) ;
- **Réduire** le nombre de revisions ;
- **Supprimer** anciens posts, révisions anciennes, vider la *corbeille* (suppression définitive) ;
- Bien **utiliser les CPT et CT** (une bonne architecture de l'information produit des requêtes plus performantes qu'une table *posts* "obèse") ;
- **Minimiser le nombre de plugins** (fichiers PHP supplémentaires à compiler et exécuter, requêtes SQL, etc. même si aucun "résultat visible" et posifif) ;
- Assets **vidéos publiées ailleurs**, par exemple sur YouTube (ou autre service).

Bien utiliser WordPress 2/2

- **Pas de requêtes SQL inutiles**;
- **Utiliser** les API `WP_Query` ou `get_posts`, **pas** `query_posts`;
- **Éviter** requêtes **SQL custom** dans les **templates** (perte des bénéfices du cache objet de WordPress !).

Optimisation

Techniques d'*optimisation* d'applications WordPress (comme tout site/service web) : obtenir la réponse HTTP et la ressource demandée par le client le plus **rapidement possible**.

Optimisation des applications web

Sur le web, les performances ne se gagnent pas spécialement sur l'optimisation du code*, **mais sur les accès lecture/écriture (IO)** : requêtes HTTP, accès disques, accès DRAM, accès base de données, connexions TCP/IP, etc.

Visionnage recommandé : Performance Optimization - PHP Tour 2016, de Fabien Potencier (créateur de Symfony)

> *Dans des langages haut niveau comme PHP, il y a **une grande différence** entre le code que vous écrivez et le code qui est réellement exécuté, même si cela n'empêche pas d'écrire du code efficace, idiomatique et performant.

Optimiser APRÈS avoir fait des mesures (profiling et monitoring)

Optimiser... :

- **Quoi** ? Quelles métriques ?
- **Où** (quelle *couche*, à quel niveau de la pile logicielle) ?
- Quels **gains** espérer ? **Quels sont les objectifs** ?
- Quels chemins d'execution les plus utilisés et lesquels posent problème ?
- Etc.

Il faut **identifier les points de congestion**. Inutile d'optimiser "tout", *aveuglément*.

Utiliser des outils pour monitorer/profiler :

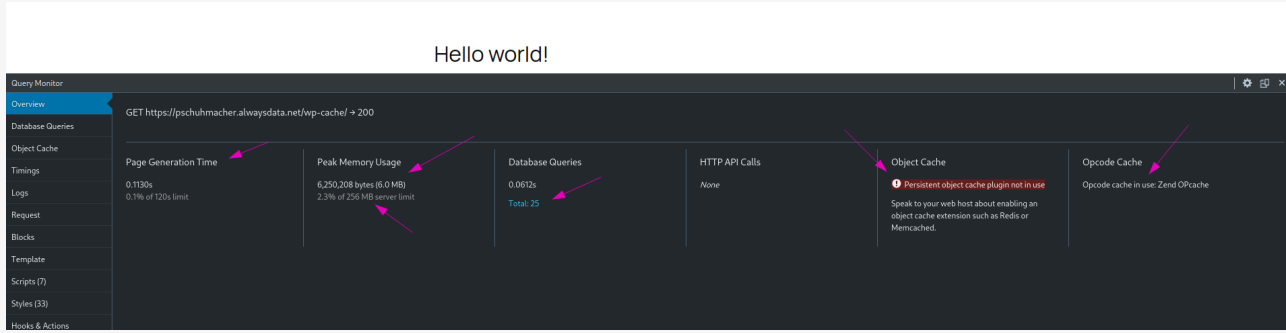
- **Couche transport et HTTP** : (réseau, protocole HTTP, etc.) ;
- **PHP** (mémoire, temps d'execution) :
  - **Zend Engine (Machine Virtuelle PHP)** : **SAPI**, configuration de l'**Opcache**, JIT, monitoring avec phptop, un profiler de scripts php, développé et maintenu par Bearstech,
  - **Vos scripts (exec)** : utiliser des outils comme XDebug, php-profiler avec les primitives de PHP, module xhprof ;
- **WordPress (*core* + vos développements)** : utiliser par exemple le plugin Query Monitor (requêtes SQL, quantité de mémoire utilisée, temps de réponses, cycle de vie détaillé de l'application) ;
- **Le chargement de la page web** dans le navigateur (temps entre requête et rendu de la page, taille de pages, plusieurs métriques) ;

Plugin Query Monitor

{.marp-bg-img}

Lire les statistiques :

- Memory Usage : Mémoire consommée par l'exécution du script PHP (`memory_get_peak_usage`) (~ 5-20 MB);
- **Database Queries** : Le nombre de requêtes SQL auprès de la base pour créer la page;
- **Page Generation Time**;
- **Object Cache** (aller voir ca dans le plugin Redis plutôt, plus détaillé);
- **Opcode Cache** (vérifier que c'est activé)



Bien choisir sa SAPI PHP : PHP-FPM ou FrankenPHP

Utiliser un serveur web qui supporte la SAPI **PHP-FPM** (Apache fast cgi, nginx) ou utiliser la nouvelle SAPI FrankenPHP (le futur).

Installer PHP-FPM (serveur Debian/Ubuntu)

```
apt install php8.5-fpm php8.5-opcache
```

Configurer PHP-FPM (processus maître)

Éditer le fichier `/etc/php/8.5/fpm/php-fpm.conf` :

```
# Nombre max de processus enfants qui peuvent échouer dans l'intervalle donné
# jusqu'à ce qu'on demande au processus maitre de restart (gracefully)
emergency_restart_threshold = 10
emergency_restart_interval = 1m
```

> Ici le processus maitre redémarre si 10 processus enfants échouent sur une période de 1 minute.

Configuration de pool de processus PHP-FPM (child processes)

Éditer le fichier (`www` est le pool par défaut, on peut en créer plusieurs) `/etc/php/8.2/fpm/pool.d/www.conf` :

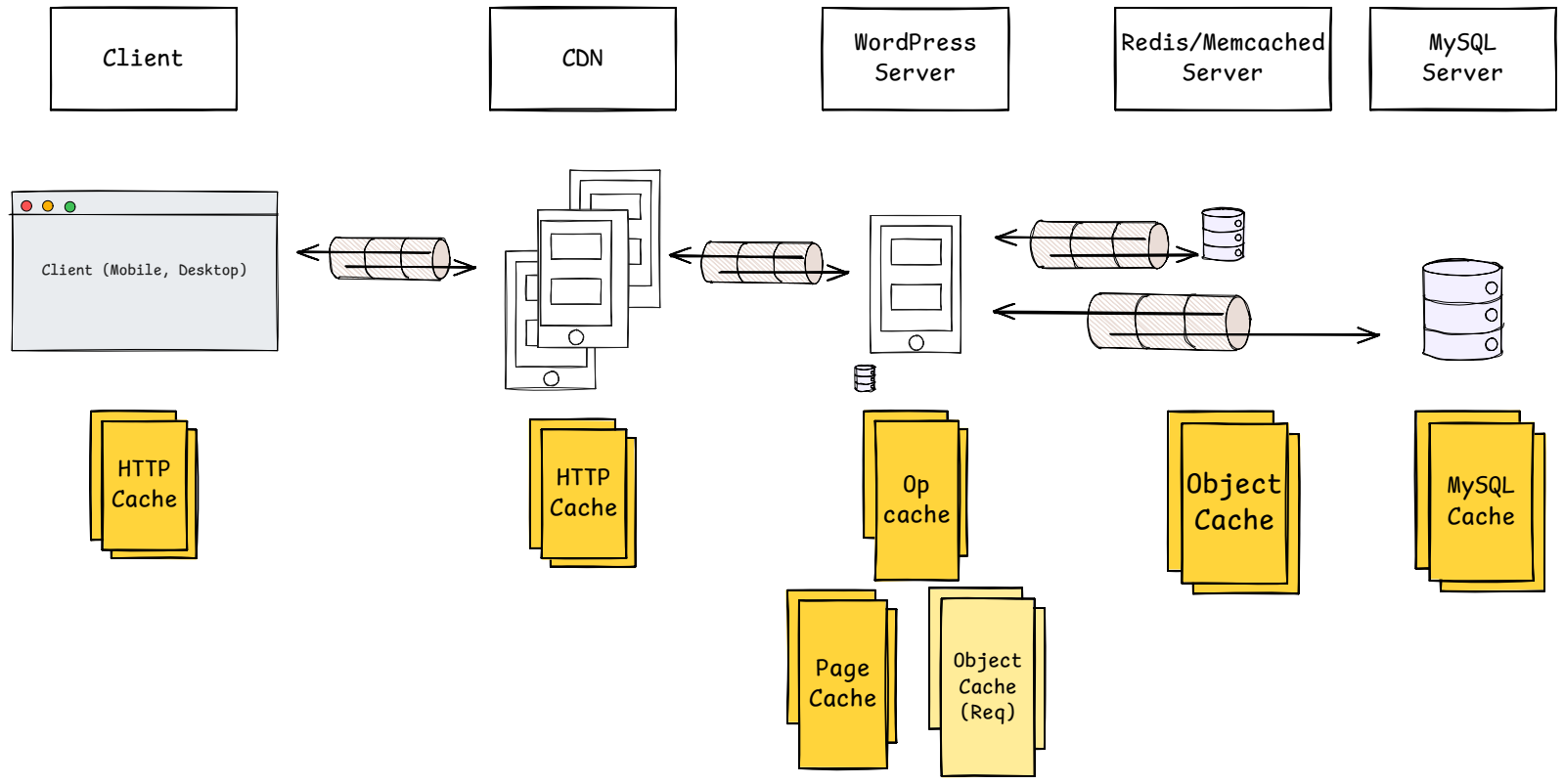
```
#proprietaire du pool (user deploy dédié)
user=deploy
group=deploy
#ip et port sur le quel php fpm accepts inbound request
listen = 127.0.0.1:9000
#Securité : n'autoriser que la machine hote de forward vers le pool
listen.allowed_clients = 127.0.0.1
#nombre max de process enfant qui peut exister chaque instant
#a adapter en fonction de la RAM de la machine (~5M par process, 512mB/5 ~ 51)
pm.max_children = 51
#Le nombre de pools dispos immédiatement qd php-fpm démarre
pm.start_servers = 3
pm.min_spare_servers = 2
pm.max_spare_servers = 4
#nb de requetes a traiter max par process avant d'etre libéré et recyclé
pm.max_requests = 1000
#path vers fichier de log pour les requetes qui prennent plus de [n] seconds a process.
#utile pour identifier les bottleneck
#slowlog = /path/vers/log
#request_slowlog_timeout = /
```

Cache(s)

> "There are only two hard things in Computer Science: cache invalidation and naming things." Phil Karlton (Netscape, Xerox Parc)

Spoiler : L'utilisation du cache n'est pas *magique*, apporte généralement des bons résultats en terme de performances mais peut poser aussi des problèmes si mal configuré. Ne **remplace pas** de suivre les bons principes de *design* et d'usage du framework. **Permet d'amplifier les performances d'un bon système**.

Les différentes couches de cache



Toute les couches de caches pour WordPress

- **Cache HTTP**. Le cache est implémenté par le standard HTTP :
  - *Via* des headers HTTP (`Cache-Control` , `ETag` , `Last-Modified`). Permet de mettre en cache les pages web **côté client**. **Fonctionnalité fondamentale et centrale du protocole**, implémenté dès HTTP 1.1 (a sauvé le web, menacé par son propre succès !) ;
  - **Page Cache** : Les pages web sont mises en cache sur le disque ou en mémoire **côté serveur**. Resservies telles qu'elles, sans exécuter du PHP ;
- **Cache de la VM PHP** (Opcache) : les *opcodes* (bytecode issu de la compilation des scripts PHP) stocké en cache par la machine virtuelle PHP ;
- ***Object cache*** :
  - **Cache natif** de WordPress durant l'execution d'un script sous forme d'objet PHP. Si des données sont redemandées durant l'execution (par ex suite à une Query Custom), les données sont resservies depuis la DRAM. Cache non persisté. **Durée de vie : traitement d'une requête**,
  - **Cache objet persisté**, via une base de données en mémoire du type Redis ou Memcached. Les objets WordPress de cache sont enregistrés en base et **réutilisés d'une requête à l'autre**,
- **Cache MySQL** : la base de données MySQL met en cache le résultat de requêtes ;
- **Cache HTTP distribué via les CDN**.

Cache PHP : Opcache et JIT

Configuration de la machine virtuelle de PHP (*Zend Engine*) via des fichiers php.ini (à privilégier) ou des directives (.htacces par ex. avec serveur Apache) :

- **Activer** Opcache (mise en cache des optcodes);
- **Configurer** l'Opcache;
- **Activer** le mode *Just In Time compilation* (JIT)
- S'assurer que **le fichier INI est bien utilisé** (en fonction de la *SAPI* PHP utilisée)

Configuration Opcache and JIT

```
;Activer l'opcache (mise en cache des opcodes, fichiers PHP compilés)
opcache.enable=1
;Taille mémoire (DRAM) alloué à l'opcache
opcache.memory_consumption=64
;Quantité de mémoire pour les 'interned string', string répétées mise en cache
opcache.interned_strings_buffer=16
; Nombre max de fichiers a stocker dans le cache opcode (> votre nombre de fichiers)
opcache.max_accelerated_files=5000
; Dev : Activé, check si cache est frais tous les revalidate_freq
; Prod : à désactiver (cache des scripts est toujours frais, a mettre à jour au déploiement !)
opcache.validate_timestamps=1
; Fréquence à laquelle valider le cache. A de l'effet uniquement si validate_timestamps est activé En dev, on revalide le cache à chaque fois
opcache.revalidate_freq=0
; Déléguer la libération de la mémoire au gestionnaire de mémoire du zend engine
opcache.fast_shutdown=1
; Utiliser le cache disque en plus du cache ram
opcache.file_cache=/var/cache/opcache
opcache.file_cache_only=0
opcache.file_cache_consistency_checks=1
opcache.enable_cli=1
; Just In Time (JIT) compilation
opcache.enable=1
opcache.enable_cli=1
opcache.jit_buffer_size=100M
opcache.jit=tracing
```

Proposition INI en mode développement

```
display_errors = Off
log_errors = On
error_log = /var/log/wordpress.log
error_reporting = E_ALL & ~E_NOTICE & ~E_DEPRECATED & ~E_STRICT
file_uploads = On
memory_limit = 64M
post_max_size = 64M
upload_max_filesize = 64M
max_execution_time = 5
zlib.output_compression = On
zlib.output_compression_level = 1
output_buffering=4096
implicit_flush=false
realpath_cache_size=64k ;Cache des path des include (require)

opcache.enable=1
opcache.memory_consumption=64
opcache.interned_strings_buffer=16
opcache.max_accelerated_files=5000
opcache.validate_timestamps=0
opcache.fast_shutdown=1
opcache.file_cache=/var/cache/opcache
opcache.file_cache_only=0
opcache.file_cache_consistency_checks=1
opcache.validate_timestamps=1
opcache.revalidate_freq=0
opcache.enable_cli=1
```

Proposition INI en mode production

> À *tuner* en fonction de l'application, son usage et de l'environnent

```
display_errors = Off
log_errors = On
error_log = /var/log/wordpress.log
error_reporting = E_ALL & ~E_NOTICE & ~E_DEPRECATED & ~E_STRICT
file_uploads = On
memory_limit = 64M
post_max_size = 64M
upload_max_filesize = 64M
max_execution_time = 60
zlib.output_compression = On
zlib.output_compression_level = 5

opcache.enable=1
opcache.memory_consumption=64
opcache.interned_strings_buffer=16
opcache.max_accelerated_files=5000
opcache.validate_timestamps=0
opcache.fast_shutdown=1
opcache.file_cache=/var/cache/opcache
opcache.file_cache_only=0
opcache.file_cache_consistency_checks=1

; Just In Time (JIT) compilation
opcache.enable=1
opcache.enable_cli=1
opcache.jit_buffer_size=100M
opcache.jit=tracing
```

Réinitialiser l'Opcache (PHP-FPM)

Reset le cache manuellement (au déploiement, **redémarrer le pool**) :

```
systemctl restart php8.5-fpm
```

Depuis un script PHP :

```
<?php
opcache_reset()
```

Cache Objet avec Redis

> Peut être fait avec une autre techno, comme Memcached

Installer Redis et le plugin Redis Object Cache

- Installer redis sur le serveur ;
- Configurer redis (auth) et lancer redis (*daemon*) ;
- Installer le plugin Redis Object Cache.

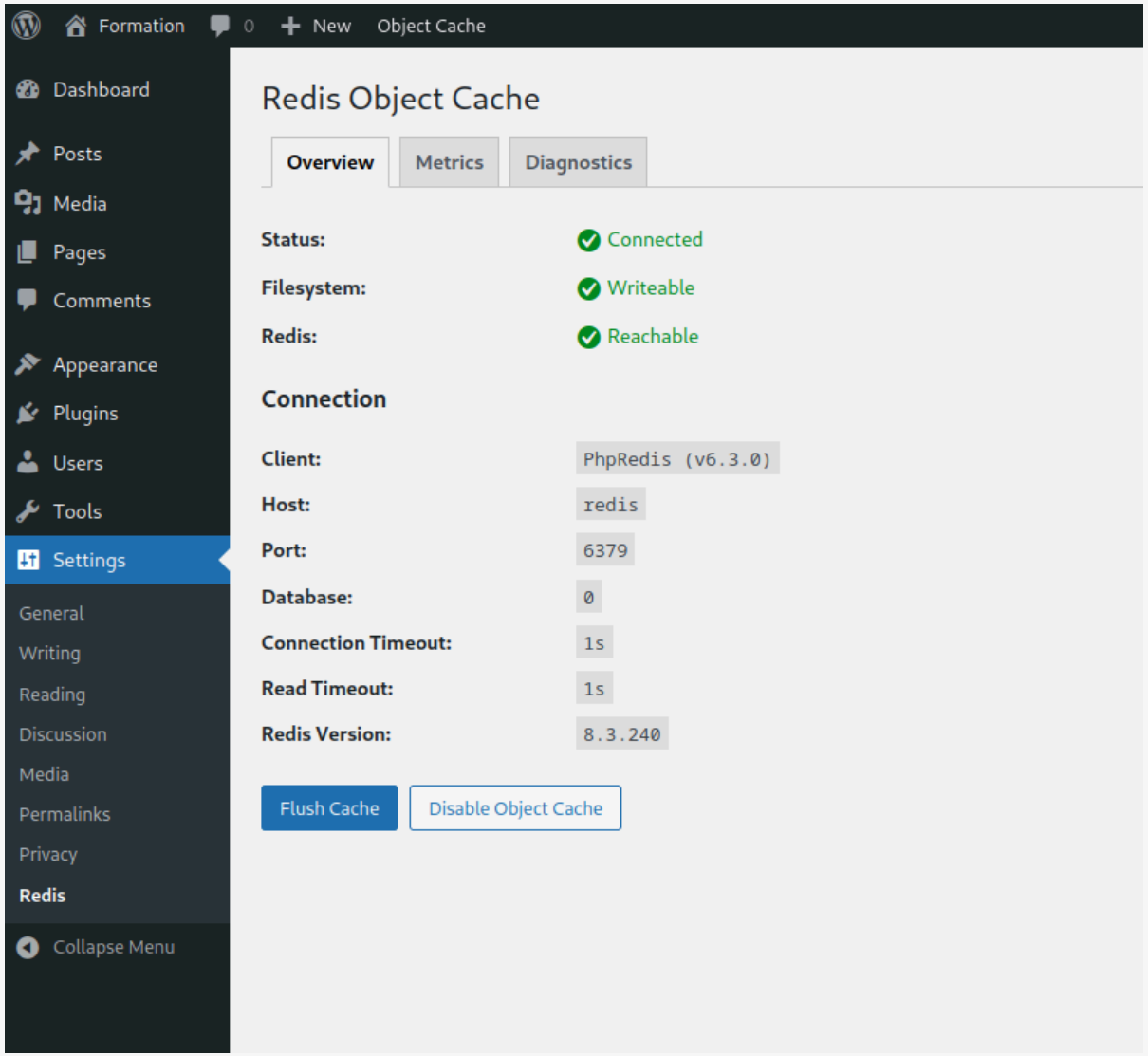
Configuration de WordPress

Dans `wp-config.php` :

```
define('WP_REDIS_HOST', 'redis');
define('WP_REDIS_PORT', 6379);
define('WP_CACHE', true);
//Autres configs importantes : (prod)
//Sécurité : créer un user redis et le renseigner ici sous la forme ['user', 'password']
//define('WP_REDIS_PASSWORD', ['user','password']);
//Fonctionne aussi avec un socket unix ('unix')
//define('WP_REDIS_SCHEME', 'tcp');
//Timeout connexion en secondes
//define('WP_REDIS_TIMEOUT', 1);
//Timeout ecriture/lecture
//define('WP_REDIS_READ_TIMEOUT', 1);
//Max TTL pour les clefs de WordPress en secondes (1h)
//define('WP_REDIS_MAXTTL', 3600);
```

> Voir toutes les configurations

Cache redis


Formation
0
New
Object Cache
Dashboard
Posts
Media
Pages
Comments
Appearance
Plugins
Users
Tools
Settings
General
Writing
Reading
Discussion
Media
Permalinks
Privacy
Redis
Collapse Menu
Redis Object Cache
Overview
Metrics
Diagnostics
Status:
Connected
Filesystem:
Writeable
Redis:
Reachable
Connection
Client:
PhpRedis (v6.3.0)
Host:
redis
Port:
6379
Database:
0
Connection Timeout:
1s
Read Timeout:
1s
Redis Version:
8.3.240
Flush Cache
Disable Object Cache

Cache redis, comment le lire (stats)

> Le plugin Query Monitor permet de l'afficher !

- **Hit ratio** H (résumé) : pourcentage de requêtes servies depuis le cache plutôt que la base de données. 98,5 % signifie que sur 100 requêtes, 98 sont servies depuis Redis et seulement 2 vont à la DB. **Plus ce chiffre est proche de 100 %, plus le cache est efficace.**
- **Hits / Misses** :
  - Hits (520) : nombre de fois où Redis a fourni un objet déjà en cache.
  - Misses (8) : nombre de fois où Redis n’a pas trouvé l’objet et WordPress a dû le générer depuis la base.
- **Size** : indique la mémoire réellement consommée par les objets mis en cache. Redis peut stocker beaucoup plus si nécessaire, selon sa configuration (`maxmemory`). En rajouter si trop de *miss*.

Calcul du Hit Ratio : $H = \frac{H}{H+M} \cdot 100$

Configuration de Redis

Via un **fichier de configuration** :

```
# Mémoire maximale : ajuster selon RAM disponible (ex : 512MB)
maxmemory 512mb

# Politique d’éviction : supprime les clés les moins utilisées
maxmemory-policy allkeys-lru

# Optionnel : durée de vie par défaut des clés expirables (TTL par défaut)
# Certaines clés WordPress n'ont pas de TTL, d'autres comme les transients ont leur TTL propre
```

Via la **CLI**, sans redémarrer Redis :

```
CONFIG SET maxmemory 512mb
CONFIG SET maxmemory-policy allkeys-lru
```

Monitoring :

```
redis-cli INFO stats
redis-cli INFO memory
redis-cli MONITOR
```

Cache Applicatif (HTTP)

Mettre en cache **les pages web et les assets** (CSS, JS, fonts, icons, etc.)

HTTP permet de définir des **politiques de cache** via les *headers* (notamment Cache-Control).

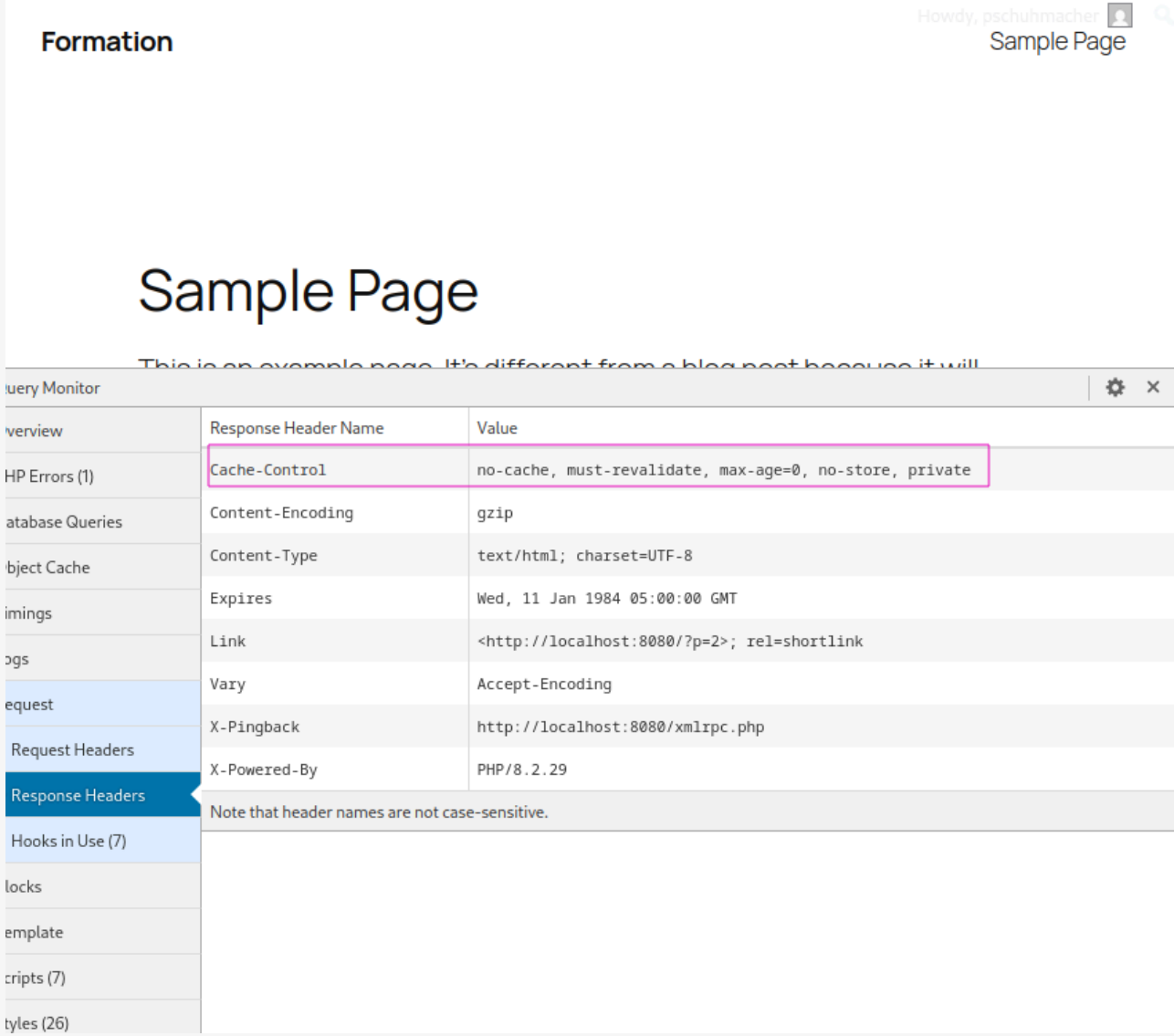
Le cache HTTP peut être placé :

- **Côté client** (navigateur);
- **Côté serveur** :
  - Sur le serveur applicatif (pages web persistées ou *Cache Page*),
  - Sur le réseau avec un *Content Delivery Network* (**CDN**), proxy.

Cache applicatif

{.marp-bg-img}

Aucun cache HTTP utilisé par défaut par WordPress!



Configurer cache applicatif

- Manuellement (définir les headers HTTP, son *cache pattern*)
- **Via un plugin** (W3 Total Cache, WP-Optimize, LiteSpeed Cache, etc.) car offre d'autres services en général (dont compression des images qui est indispensable). **All-in-one**

La majorité des plugins de cache WordPress sont **orientés serveur** (pas client*) : cache pages HTML sur disque ou mémoire côté serveur.

> *Les assets (CSS/JS/fonts) sont bien mis en cache côté client par contre.

Gains obtenus avec ces couches

Ces trois couches (cache objet persisté, cache applicatif, opcache) réunies et une fois le cache *chauffé* :

- **Réduisent à zéro la compilation** des scripts PHP !
- Optimisent l'**execution** du code PHP ;
- Mettent en **cache tous les assets côté client** (téléchargés une seule fois !) ;
- **Réduisent les requêtes à la base de données à** (presque) **zéro** pour toutes les pages publiques !

> Et on ne parle pas des gains obtenus en utilisant le pool de processus PHP de PHP-FPM !

Cache de la base de données

Il reste encore à configurer la cache du côté de la base de données.

> Avec Redis ou Memcached, cette optimisation sera moins critique étant donné que le cache objet va réduire le nombre de requêtes SQL. Mais c'est encore une piste d'amélioration possible !

Optimiser la base de données MySQL

Pistes : **Indexs**, **cache base de données** (*query cache*), thread pools, optimizing tables, compression, optimizing queries (SQL), data structures optimisations, etc.

Quelques liens utiles :

- MYSQL Optimization, guides officiels ;
- MariaDB Optimization and Tuning, guides officiels ;
- Configurer le serveur MySQL ;
- Utiliser des indexs, dans les tables InnoDB de la base de données WordPress .

Plugins recommandés

Bon *design*, optimisations, maintenance, sécurité et misc.

Plugins recommandés : Cache et performances

> Combiner les bons plugins pour activer toutes les couches de cache + optimisation des assets (CSS, JS et **images**)

- WP-Optimize – Cache, Compress images, Minify & Clean database to boost page speed & performance, de TeamUpdraft, un très bon vendor WordPress. Cache applicatif (page cache), minification des assets et compression des images, **nettoyage et maintenance de la base de données**
- WP Redis Cache, **dédié à la persistance du cache objet** de WordPress dans une base de données **Redis**, maintenu par Till Krüss, un spécialiste du cache;
- LiteSpeed Cache, de LiteSpeed Technologies, **solution cache tout en un** : cache applicatif (page cache, HTTP cache), cache objet avec Redis, cache CDN, minification CSS/JS, optimisation des images, etc. **Bonus : Bonne interface d'administration**, simple et intuitive (ne pas utiliser avec le plugin WP Redis Object Cache redondant OU avec désactiver Object Cache)

Plugins recommandés : Sécurité

Installer un **plugin dédié à la sécurité** du site (peu importe lequel : WordFence, BulletProof Security, etc.) est **indispensable** !

- WordFence, la version gratuite est déjà largement suffisante : **firewall**, two-factor authentication, brute force protection, **IP blocklist**, **malware scanner**, etc. Dispose aussi d'un outil CLI

> Voir une liste des meilleurs plugins de sécurité publié par WPMarmite

Plugins recommandés : ACF PRO

> What else... **Should be** WordPress core... Mais l'histoire en a décidé autrement.

Plugins recommandés : Formulaires

Les formulaires : édition des champs, markup et intégration (HTML, CSS), **navigation** (plusieurs pages ?), **validation**, **traitement** et **historique**.

> En avez-vous *vraiment* besoin ? Souvent... oui, car c'est du travail (surtout s'ils sont complexes, sur plusieurs pages, avec de la validation conditionnelle nécessitant du JavaScript, etc.) mais vous pouvez aussi développer vos propres formulaires et leurs traitements.

- Contact Forms 7, le plus **simple**, donc **élégant**. Coup de coeur.
- GravityForms, le plus puissant. Payant. **Comme ACF PRO, investissement rentable** :
  - *Développer* des formulaires complexes avec un excellent *Form Builder* (UI),
  - *Valider* les données,
  - *Collecter* les données,
  - **Orienté dev**, très **bien documenté** et **bonnes API**.

Références utiles

Consulter la bibliographie ("lectures" recommandées)