





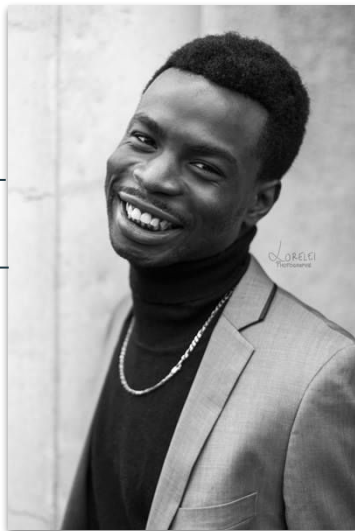
Tests de performance 4IW



Qui suis-je ?



Symfony



COYOTE

OPENCLASSROOMS

Qui êtes-vous ?

Déroulement du cours

- 15H
- Théorie et pratique
- Feedback en fin de cours
- Soutenance et contrôle continue

Règles d'or

Pas de retard

Respect

Plaisir

Au programme

1. Définir et comprendre ce qu'est la performance d'une application
2. Analyser et comprendre le profiler de Symfony
3. Analyser et comprendre les performances lors d'une montée en charge avec Vegeta
4. Installation et prise en main de BlackFire
5. Auditer et améliorer les performances d'une application (cache, doctrine, etc...)
6. Contrôle continue
7. Comment aller plus loin avec BlackFire
8. Soutenance

Qu'est-ce que la performance ?

- Optimiser l'utilisation des ressources matérielles à notre disposition
 - **Back** : Gestion des ressources côté serveur
 - **Communication réseau** : Envoie des données
 - **Front** : Affichage des données pour le client
- Plusieurs composantes à prendre en compte
 - Temps d'exécution
 - La mémoire utilisée
 - L'utilisation du réseau

En php native

Le temps d'exécution se mesure en faisant la soustraction de `microtime(true)` en fin de script et `microtime(true)` en début de script

La mémoire utilisée se mesure en faisant la soustraction de `memory_get_usage(true)` en fin de script et `memory_get_usage(true)` en début de script

Description

```
microtime ( [ bool $get_as_float = FALSE ] ) : mixed
```

microtime() retourne le timestamp Unix, avec les microsecondes. Cette fonction est uniquement disponible sur les systèmes qui supportent la fonction `gettimeofday()`.

Description

```
memory_get_usage ( [ bool $real_usage = FALSE ] ) : int
```

Retourne la quantité de mémoire allouée à PHP à cet instant.

Atelier 1

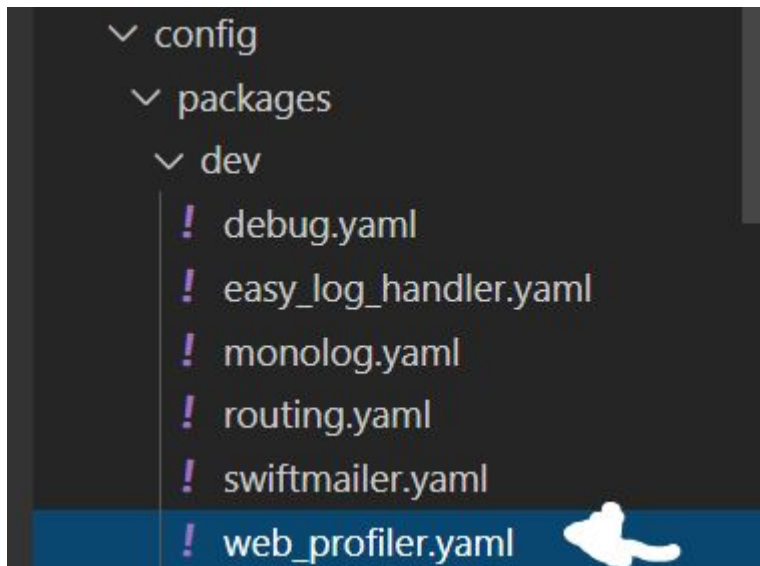
- Prendre le projet Symfony du semestre 1
- Regarder le temps d'exécution de la page d'index
- Regarder la mémoire utilisée de la page d'index
- Temps 20 minutes

Plus de détails avec le profiler

- Visible en environnement dev ou test mais pas en environnement de production
- sinon, il faut l'installer : https://symfony.com/doc/current/reference/configuration/web_profiler.html



Configuration webprofiler



```
# config/packages/dev/web_profiler.yaml
web_profiler:
    toolbar: true
    intercept_redirects: false
```

Trois options :

- `excluded_ajax_paths` : pour exclure des requêtes ajax selon une regex
- `intercept_redirects` : à true, le profiler arrête le site pour vous montrer le profiler avant la redirection
- `toolbar` : permet d'afficher ou cacher la toolbar

Web profiler

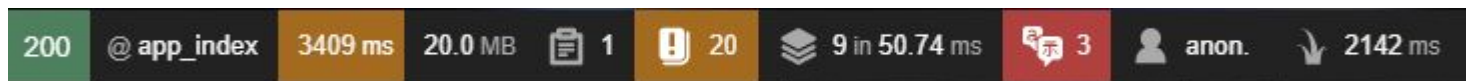


Dans l'ordre, le profiler montre :

1. *Le code HTTP*, en passant la souris on voit le controller, la route et si une session est active
2. *Le nom de la route*, en passant la souris on voit le controller, la route et si une session est active
3. *Le temps d'exécution de la page*, en passant la souris on voit le temps d'initialisation de Symfony
4. *La mémoire utilisée*, en passant la souris on voit la limite de mémoire autorisée
5. Le nombre de formulaire
6. Les erreurs, warnings et dépréciations
7. L'utilisation du cache
8. Les traductions manquantes
9. Les informations sur la session et la sécurité
10. Le temps d'exécution de Twig

Importance du cache

Première connexion de la journée



Réactualisation



Ouverture du profiler


Sur une application FullStack : il faut cliquer sur la toolbar

Sur une API : il faut regarder la réponse d'en-tête

The screenshot displays a web profiler interface. At the top, the 'Method' is 'GET' and the 'Request URL' is 'http://localhost:8001/api/blog/public/articles/apikeycecile98'. Below this, the 'Request parameters' section is collapsed. The 'HEADERS' tab is selected, showing options to 'COPY', 'SOURCE VIEW', and 'ADD HEADER'. The response status is '200 OK' with a duration of '477.20 ms'. The response details section shows the request method and URL, followed by tabs for 'RESPONSE HEADERS' (10 items), 'REQUEST HEADERS' (0 items), 'REDIRECTS' (0 items), and 'TIMINGS'. The response headers are listed as follows:

```
Host: localhost:8001
Date: Sat, 18 Jan 2020 13:44:36 GMT
Connection: close
X-Powered-By: PHP/7.2.25
Cache-Control: no-cache, private
Date: Sat, 18 Jan 2020 13:44:36 GMT
Content-Type: application/json
X-Debug-Token: f59b0a
X-Debug-Token-Link: http://localhost:8001/_profiler/f59b0a
X-Robots-Tag: noindex
```


Ouverture du profiler - Général


 **Symfony Profiler**


search on symfony.com


http://127.0.0.1:8003/
Method: GET HTTP Status: 200 IP: 127.0.0.1 Profiled on: Sat, 18 Jan 2020 13:27:33 +0000 Token: 0b561b


Last 10 Latest Search


 Request / Response


 Performance


 Validator


 Forms


 Exception


 Logs 20


 Events


 Routing


 Cache


 Translation 3


 Security

 Twig

 Doctrine

 E-mails

 Debug

 Configuration

Settings

AppController :: index

Request Response Cookies Session Flashes Server Parameters Sub Requests 1

GET Parameters

No GET parameters

POST Parameters

No POST parameters

Uploaded Files

No files were uploaded

Request Attributes

Key	Value
_controller	"App\Controller\AppController::index"
_firewall_context	"security.firewall.map.context.main"
_route	"app_index"
_route_params	[]

Profiler - Request/Response

- La request pour savoir les requêtes HTTP effectuées
- La response pour savoir le retour de notre page
- Les cookies et la session que vous connaissez déjà
- Les Flashes qui sont des messages en session qui disparaissent au rechargement de la page
- Les sous requêtes pour les contrôleurs appelés dans le twig

Profiler - Performance

Performance metrics

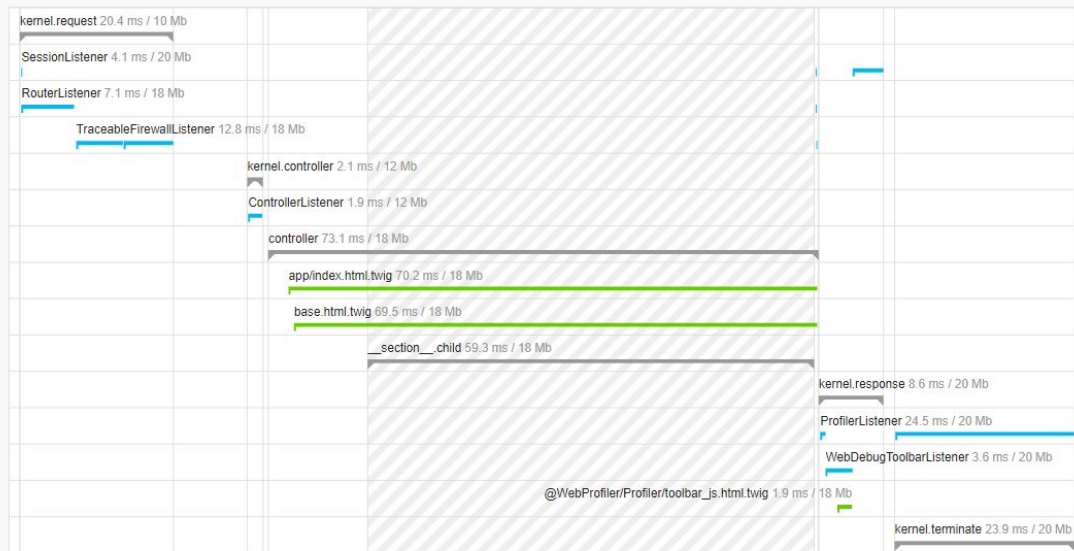
212 ms	74 ms	20.00 MB	1	59.3 ms
Total execution time	Symfony initialization	Peak memory usage	Sub-Request	Sub-Request time

Execution timeline

Threshold ms (timeline only displays events with a duration longer than this threshold)

Main Request 138.6 ms

■ default ■ section ■ event_listener ■ template



Note: sections with a striped background are shared to sub-requests

Profiler - Métrique doctrine

Last 10LatestSearch

Request / Response

Performance

Validator

Forms

Exception

Logs26

Events

Routing

Cache

Translation

Security

Twig

Doctrine2

Query Metrics

4	3	6.43 ms	2
Database Queries	Different statements	Query time	Invalid entities

Queries

[Group similar statements](#)

#	Time	Info
1	5.63 ms	<pre>SELECT b0.id AS id_0, b0.title AS title_1, b0.slug AS slug_2, b0.cover AS cover_3, b0.description AS updated_at_5, b0.created_at AS created_at_6, b0.body_text AS body_text_7, b0.deleted_at AS deleted_at_9, b0.user_id AS user_id_10, b0.category_id AS category_id_11 FROM user u1_ ON b0.user_id = u1.id INNER JOIN jwtuser j2_ ON u1.id = j2.user_id INNER JOIN client j2.belongs_to_client_id = c3.id INNER JOIN configuration c4_ ON c3.app_configuration_id = c4.b5_ ON b0.category_id = b5.id WHERE c4.api_key = ? AND b0.deleted_at IS NULL ORDER BY b0.id</pre> <p>Parameters:</p> <pre>["apikeycecile98"]</pre> <p>View formatted query View runnable query Explain query</p>
2	0.29 ms	<pre>SELECT t0.id AS id_1, t0.public_key AS public_key_2, t3.id AS id_4, t3.name AS name_5, t3.email AS password_7, t3.roles AS roles_8, t3.client_id AS client_id_9, t3.user_id AS user_id_10, t3.belongs_to_client_id_11 FROM user t0 LEFT JOIN jwtuser t3 ON t3.user_id = t0.id WHERE t0.id = ?</pre>

Atelier 2

- Ouvrir votre site internet et mesurer les performances
- Faire un rapport d'audit de performance pour chaque page => Important pour la soutenance
- Analyser ce qui est améliorable pour diminuer les performances
- Atelier d'1h30
- Ressources
 - <https://symfony.com/doc/current/performance.html>
 - <https://blog.nicolashachet.com/developpement-php/optimiser-les-performances-de-son-code-php/>
 - <https://symfony.com/doc/current/cache.html>
 - https://symfony.com/doc/current/http_cache.html
- **Correction**

Optimisation doctrine

Il est intéressant dans certains cas de s'intéresser au fetch dans les relations doctrine

```
*  
* @ORM\ManyToOne(targetEntity="Post", inversedBy="comments", fetch="EAGER")
```

Lazy (par défaut): Le mode LAZY est celui par défaut. Les données sont chargées uniquement si nécessaires et l'appel aux données de relations provoque une requête supplémentaire.

EAGER: Le mode EAGER précharge les données de la relation automatiquement en réalisant la jointure par défaut. Ainsi votre entité sera nettement plus grande mais vous économiserez des requêtes.

EXTRA_LAZY: les données ne seront pas entièrement chargées mais vous pourrez toutefois réaliser des opérations du type "contains()", "slice()" ou "count()" sur la collection.

Optimisation doctrine

Dans certains cas, une entité incomplète suffit pour réaliser une opération, pas besoin de récupérer l'entité complète.

```
$post = $this->getDoctrine()->getEntityManager()->getReference('Post', $postId);
```

```
$topic->setPost($post );
```

Dans ce cas là, vous pouvez mettre à jour Topic juste en chargeant Topic et non Post et Topic

Optimisation doctrine

Lorsque vous faites des modifications mineurs, éviter de persister un objet complet. Utiliser une Query pour Update juste un champs :

```
$qb = $this->createQueryBuilder('p');  
$qb->update()  
->set('p.expiryAt', ':newExpiryAt')  
->setParameter('newExpiryAt', $newExpiryAt);  
return $qb->getQuery()->execute();  
}
```

En plus cela évite de flush et de regarder tous les objets.

Optimisation doctrine

Les tableaux (grosses données) : Lorsque vous avez des listes trop grosses et donc gourmande en mémoire, il est intéressant d'utiliser des Array, donc un Repository on utilise `getArrayResult()` au lieu de `getResult()`, et de gérer un tableau.

Le cache le plus paresseux : Pour les modifications et les suppressions, il est intéressant de vider le cache entre chaque requête, car le cache de doctrine prend énormément de temps à se vider et garde énormément de données, dans une boucle, il est intéressant de flush chaque modification et immédiatement après de clear (`$em->clear()`)

Recommandation de doctrine project

Il est fortement recommandé d'utiliser un bytecode cache comme APCu ou Memcached. Un cache en bytecode supprime la nécessité d'analyser le code PHP à chaque requête et peut améliorer considérablement les performances.

doctrine:

orm:

```
metadata_cache_driver: apcu
```

```
query_cache_driver: apcu
```

```
result_cache_driver: apcu
```

ReadOnly pour les entités (@Entity(readOnly=true)) permet d'avoir des entités non modifiable (on peut quand même faire des insertions et des suppressions. De cette manière quand on Flush, les entités sont ignorées et vous pouvez gagner des précieuses milisecondes.

Le service container

Le service container est fait en multiple fichier par défaut, si vous avez activé le preloader de PHP7.4 vous pouvez gagner du temps en demandant que le service container soit dans un unique fichier

```
parameters:
```

```
    container.dumper.inline_factories: true
```

Ajax

Lorsque vous récupèrer du JSON il est plus simple de récupérer un objet et de le serialiser.
Cependant, il est plus intéressant de faire une requête directement et de renvoyer le résultat de la requête, soit via un Select du QueryBuilder, soit directement en NativeSql

=> <https://www.doctrine-project.org/projects/doctrine-orm/en/latest/reference/native-sql.html#native-sql>

Optimisation Front

Au niveau du front, effectivement il y a WebPack qui est souvent utilisé et qui permet de minifier les assets, couplé avec LiipImagineBundle pour minifier permet d'optimiser la partie Twig.

Cependant, il est possible aussi d'utiliser HTTP/2 pour précharger les assets avec Symfony/WebLink (https://symfony.com/doc/current/web_link.html)

Dans le twig, plutôt que de faire appel à des alias qui doivent rechercher ce que vous souhaitez appeler, appeler la méthode `post.getAuthor()` au lieu de `poste.author`

Optimisation - Kernel Event

Les performances de base des applications Symfony sont également profondément affectées par les events kernel lents qui sont exécutés dans chaque requête.

Il est donc préférable de toujours s'assurer de ne pas appeler une base de données ou des services externes dans les events, ce qui pourrait ralentir considérablement leur processus.

Optimisation - Sous-requête

Les applications avec plusieurs vues sur une même page peuvent être découplées avec des sous-requêtes internes qui permettent d'appeler ou de rendre les multiples contrôleurs dans la même requête PHP.

Cependant, l'utilisation de sous-requêtes internes peut ralentir les performances de votre application Symfony car chaque sous-requête subira le cycle de vie des événements `HttpKernel` et entraînera donc une augmentation des temps de réponse.

Optimisation - Arrière plan

Une meilleure façon d'améliorer les performances de base de Symfony est de déléguer à l'arrière-plan les tâches essentielles comme l'envoi de courriers, le traitement des fichiers et des images téléchargés, etc.

L'événement `kernel.terminate` est exécuté après l'envoi de la réponse à l'utilisateur.

L'utilisation de `kernel.terminate` à sa limite, si plusieurs choses ont besoin d'être exécuté en arrière plan, il faut penser à utiliser un système de queue. (RabbitMQ)

Optimisation - Bundle

L'utilisation de bundle à foison est l'un des facteurs impactant le plus les performances d'un site.

Il faut les limiter au maximum.

Cache

Avec Symfony, on distingue la plupart du temps l'utilisation de 2 caches

Le cache côté serveur (APCu /Memcachedpar exemple) et le HTTPCache (côté client, ce que je vous ai montré la dernière fois)

APCu permet de cacher jusqu'à 512Mo (ce qui est déjà bien vous me direz) mais est lié à des instances de PHP, par conséquent on oublie le loadbalancing et le cache de requête SQL (mais on peut mettre en place tout de même le cache de résultats des requêtes SQL)

Memcached, permet d'aller plus loin en terme d'espace (site avec du beaucoup de contenu), mais il est plus complexe à utiliser, par contre il permet de cacher les requêtes SQL et donc de gagner du temps sur le passage de DQL à SQL

Déploiement

Optimisation du cache :

```
composer install --no-dev --optimize-autoloader
```

Savoir en détail le temps d'un script

```
use Symfony\Component\Stopwatch\Stopwatch;

$stopwatch = new Stopwatch();

// starts event named 'eventName'
$stopwatch->start('eventName');

// ... run your code here

$event = $stopwatch->stop('eventName');
// you can convert $event into a string for a quick summary
// e.g. (string) $event = '4.50 MiB - 26 ms'
```

Vegeta - Performance lors d'une montée en charge

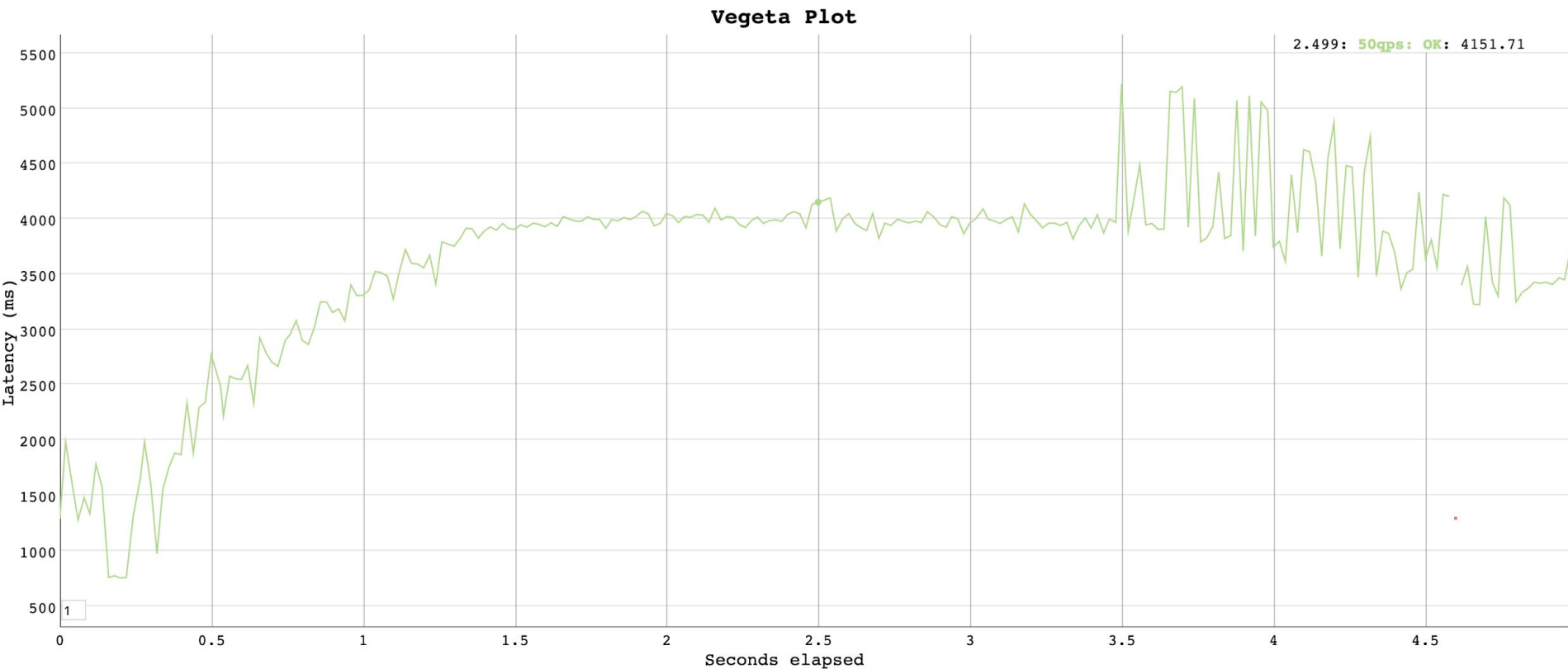
Vegeta est un outil de test de charge HTTP polyvalent conçu pour répondre au besoin de d'appeler des services HTTP avec un taux de requête constant.

<https://github.com/tsenart/vegeta>

Présentation de Vegeta et exemple

Installer et utiliser Vegeta pour améliorer votre Audit de performance

Vegeta - Exemple 50 requêtes par seconde



Vegeta - Exemple de script simple

```
1 echo "GET https://127.0.0.1:8000/" | vegeta attack -name=50qps -rate=50 -duration=5s > results.50qps.bin
2 cat results.50qps.bin | vegeta plot > plot.50qps.html
3 echo "GET https://127.0.0.1:8000/" | vegeta attack -name=60qps -rate=60 -duration=5s > results.100qps.bin
4 vegeta plot results.50qps.bin results.100qps.bin > plot.html
5
6 #info exemple post
7
8 #Lien de l'url en post
9 #POST https://127.0.0.1:8000/post
10 #X-Account-ID: 99
11 #@chemin/de/votre/json.json
```

Atelier

Installer et utiliser Vegeta pour faire des rapports de 5 pages de votre application et déterminer à quel moment l'application à un pic de charge.

Installer et configurer votre cache (ou autre optimisation) puis relancer Vegeta et faite un nouveau rapport

Quelle différence de charge constatez-vous sur votre serveur local ?