Florian Durand
Désirée Deutou
Basile Beauprez

Audit tests de performances

I. Introduction

A. Avantages de la performance selon le type d'application

Nous sommes ici en présence d'un site destiné à des activités communautaires, plus particulièrement, le partage de figures acrobatiques en ski. Ce site n'a aucun but de rentabilité ou de visites. En ce sens, des performances élevées servent surtout à améliorer l'expérience utilisateur, ainsi qu'à attirer des utilisateurs mobiles. On peut également soulever le point des coûts au niveau de la machine abritant l'application, ou le SEO pour le référencement de la page dans les moteurs de recherche.

B. Moyenne des performances visées

Toujours dans l'optique d'un site communautaire à but non lucratif, les performances visées ne sont pas extrêmement élevées, une performance dite "moyenne" suffira amplement à optimiser l'expérience utilisateur au vu des fonctionnalités présentées par le site. En ce sens des temps de chargements inférieurs à 300ms et une charge processeur minimale pour le client suffiront.

C. Etat des lieux global de l'application

En l'état actuel des choses, le site est relativement lent et consommateur en ressource processeur, cela à cause des controller qui ne sont pas optimisés, notamment au niveau des appels à la base de données. La taille des images utilisée peut également entrer en cause, celle-ci n'étant pas compressée de manière optimale. Une meilleure optimisation du cache pourrait également aider à améliorer les performances.

D. Mise en place des outils accélérant le site

Afin d’optimiser le temps de chargement, on pourra installer Cloudflare sur le serveur afin de réduire la distance de l’utilisateur au serveur.

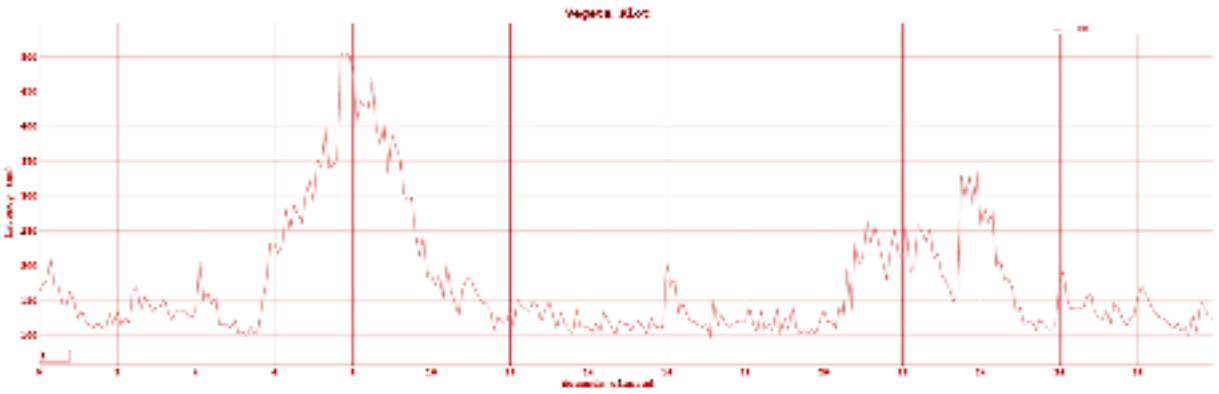
II. Corps

A. Performance avant modification

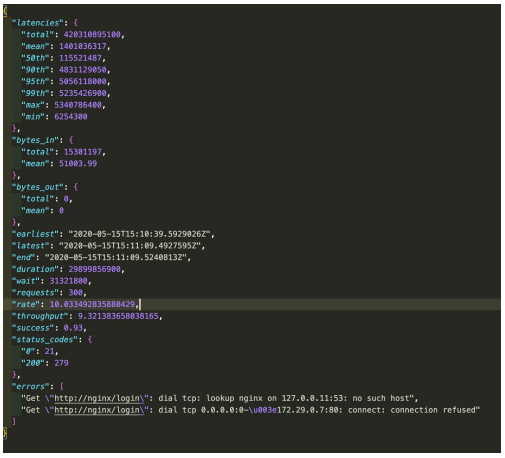
Vegeta:

Vegeta est un outil de test de charge HTTP polyvalent conçu pour répondre au besoin d’appeler des services HTTP avec un taux de requêtes constant. Il nous permet de savoir si notre application peut tenir une charge importante.

Cette commande :
“sh -c 'echo "GET http://nginx/login" | vegeta attack -duration=30s -rate=10 | vegeta plot > plot.html'”
permet de demander à Vegeta d’exécuter 10 requêtes par seconde pendant 30 secondes.



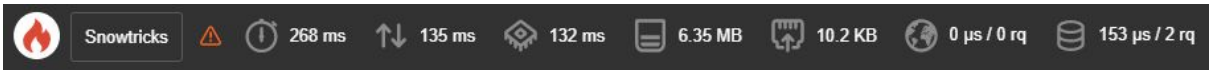
Ce graphique, généré par vegetta met en exergue un “pic” de latence au bout d’un certain nombres de requêtes. Cela qui pourrait s’expliquer par une consommation importante des ressources du processeur de certaines fonctions du corps de symfony. Nous constater que 99% des requêtes ont été servies en pratiquement 500ms. De plus elles n'ont pas toutes abouti, 21 sur 300 ont carrément expiré.



Profiling blackfire:

Blackfire est un outil qui permet aux développeurs d'assurer un suivi de l'application, que ce soit au niveau d'améliorer sa performance, de faire du debug ou bien du testing.

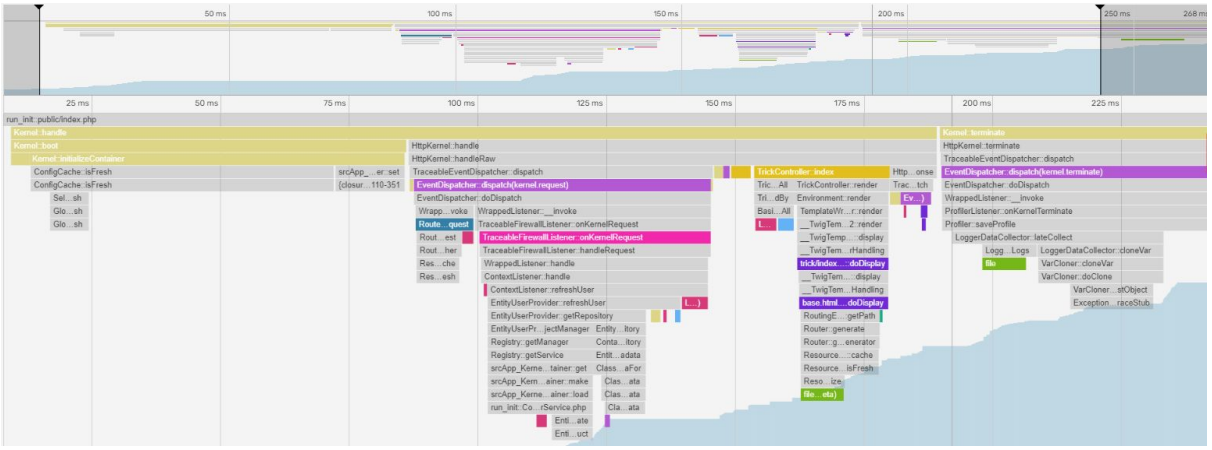
De plus il propose un retour visuel avec un graphique pour beaucoup mieux comprendre les problèmes et l'organisation générale du code .



Le profiler de blackfire nous permet d'avoir des informations de l'application tel que le temps de chargement de la page, la bande passante utilisée ou bien l'utilisation du processeur.

Dans cette application "snowtricks", avant application des optimisations de performances nous avons:

- Temps de chargement total: 268ms
- Utilisation du CPU: 132ms
- Temps de requêtage SQL: 153 micro seconde



B. Optimisations générales

- Upgrade à la version supérieur de Symfony
- Limiter l'utilisation de bundles

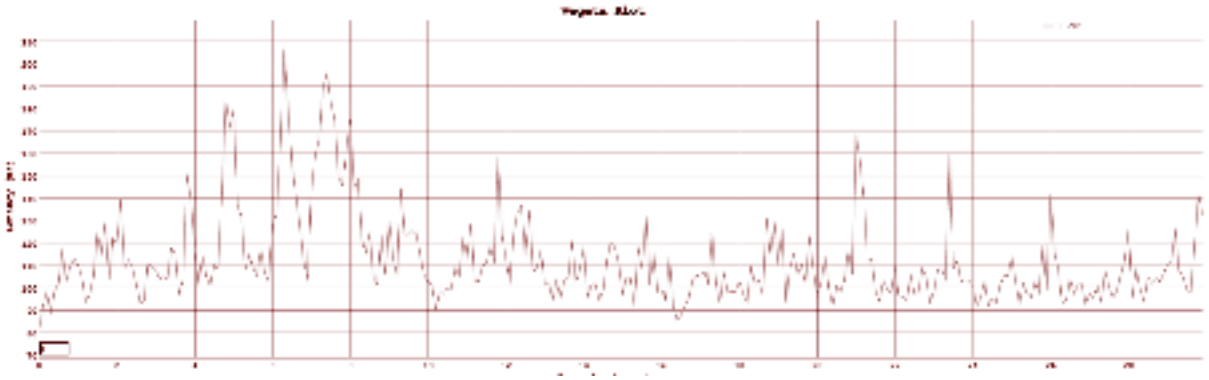
C. Optimisations spécifiques

- Éviter de persister un objet complet, créer une requête personnalisée, ce qui permet d'éviter un flush
- Optimisation du cache: composer install --no-dev --optimize-autoloader
- Changer la façon de fetch les entités => EXTRA_LAZY
- Changement de la gestion du cache de doctrine => apcu
- Demander à ce que le service container soit dans un unique fichier
- Installation de webpack encore pour gérer les assets et les minifier

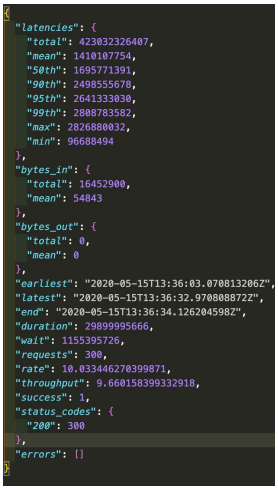
D. Performance après modifications

Vegeta :

Après modification on constate une nette amélioration du temps de latence, ici `"max": 2826880032` contre un pic à `"max": 5340786400` avant modification. Ainsi qu'un taux de réussite à 100% vu qu'il n'y a pas d'erreurs.

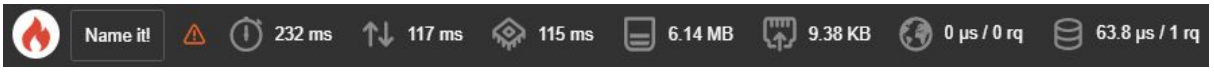


Le graphe est également plus homogène et équilibré.



```
{
  "latencies": {
    "total": 423032326407,
    "mean": 1410107754,
    "50th": 1695771391,
    "90th": 2498555678,
    "95th": 2641333030,
    "99th": 2808783582,
    "max": 2826880032,
    "min": 96688494
  },
  "bytes_in": {
    "total": 16452900,
    "mean": 54843
  },
  "bytes_out": {
    "total": 0,
    "mean": 0
  },
  "earliest": "2020-05-15T13:36:03.070813206Z",
  "latest": "2020-05-15T13:36:32.970808872Z",
  "end": "2020-05-15T13:36:34.126204598Z",
  "duration": 29899995666,
  "wait": 1155395726,
  "requests": 300,
  "rate": 10.033446270399871,
  "throughput": 9.660158399332918,
  "success": 1,
  "status_codes": {
    "200": 300
  },
  "errors": []
}
```


Profiling blackfire :



- Temps de chargement total: 232ms
- Utilisation du CPU: 115ms
- Temps de requêtage SQL: 63µs

On constate donc une amélioration de :

- 13,5% pour le temps de chargement
- 12,9% pour l'utilisation du CPU
- 58,4% pour le temps de requêtage SQL

III. Synthèse

A. **Qu’est ce que le client gagne ?**
   En améliorant la performance, le client améliore notamment l'attractivité de son site, cela réduisant le taux de rebond et améliorant l'expérience des utilisateurs, comme le site contient potentiellement beaucoup d'images et de vidéos, un bonne performance aura un impact visible sur l'attractivité de la plateforme.
B. **Quel temps estimé pour les modifications ?**
   Afin de mener à bien toutes les optimisations de la plateforme, ainsi que la mise en place des outils externes destinés à réduire le temps de chargement des utilisateurs, nous estimons une durée de travail d'environ 1 à 2 mois. Ce afin de pouvoir tester en profondeur l'application une fois les optimisation terminées.
C. **Qu’est ce qui sera fait ?**
   Le cache sera optimisé afin de permettre un chargement plus rapide des pages déjà visitées, les tâches répétitives seront déléguées à d'autres services afin de ne pas impacter la performance pour l'utilisateur. Les bundles de symfony installés seront également optimisés et limités, certains pouvant être très consommateur en terme de performance. En plus ce cela, les requêtes à la base de donnée seront optimisées en modifiant les méthodes de requête de celle-ci et en optimisant le cache de doctrine. Finalement, le package webpack sera installé afin de minifier les fichiers envoyés à l'utilisateur et ainsi réduire la quantitée de données envoyées.