

Profiling et optimisation des performances avec blackfire

a. Optimiser l'autoloader de Composer :

- **Problématique :**



A chaque fois que composer est appelé à charger une classe, il doit vérifier que cette dernière existe. Cette vérification conduit à un temps d'exécution supplémentaire.

200 GET https://localhost:8000/app.php/tasks



get_tasks

Created 13 minutes ago by moez thabti

Compare



php



0 rq

246 ms

1.33 ms

245 ms

10.7 MB

3.4 kB

0 µs / 0 rq

707 µs / 2 r...

get_tasks			
search			
Functions	Function calls	% Excl. ▾	% Incl.
	...r\Autoload\includeFile		257
	...toload\includeFile@1		122
	...findFileWithExtension		444
	file_exists		489
	...toload\includeFile@2		50
	...rojectContainer::load		14
	...patcher::sortListeners		5
	...ectContainer::load@2		7
	...ispatcher::doDispatch		5
	...ORM/UnitOfWork.php		1
	...ectContainer::load@1		11
	substr		1 728

1 callers (444 calls)



Composer\Autoload\ClassLoader
findFileWithExtension Q



35.2 ms



63.1 μ s



35.2 ms



0 B



0 B



file_exists

489

5 callers (489 calls)



file_exists Q



12.7 ms



64.4 μ s



12.6 ms



0 B



0 B



- **Optimisation :**

L'optimisation consiste à générer une Class Map.

L'autoloader de composer scannera l'ensemble de l'application une seule fois, ensuite il créera un tableau contenant la localisation de toutes les classes et enfin va stocker tous ces détails dans un fichier qui s'appelle :

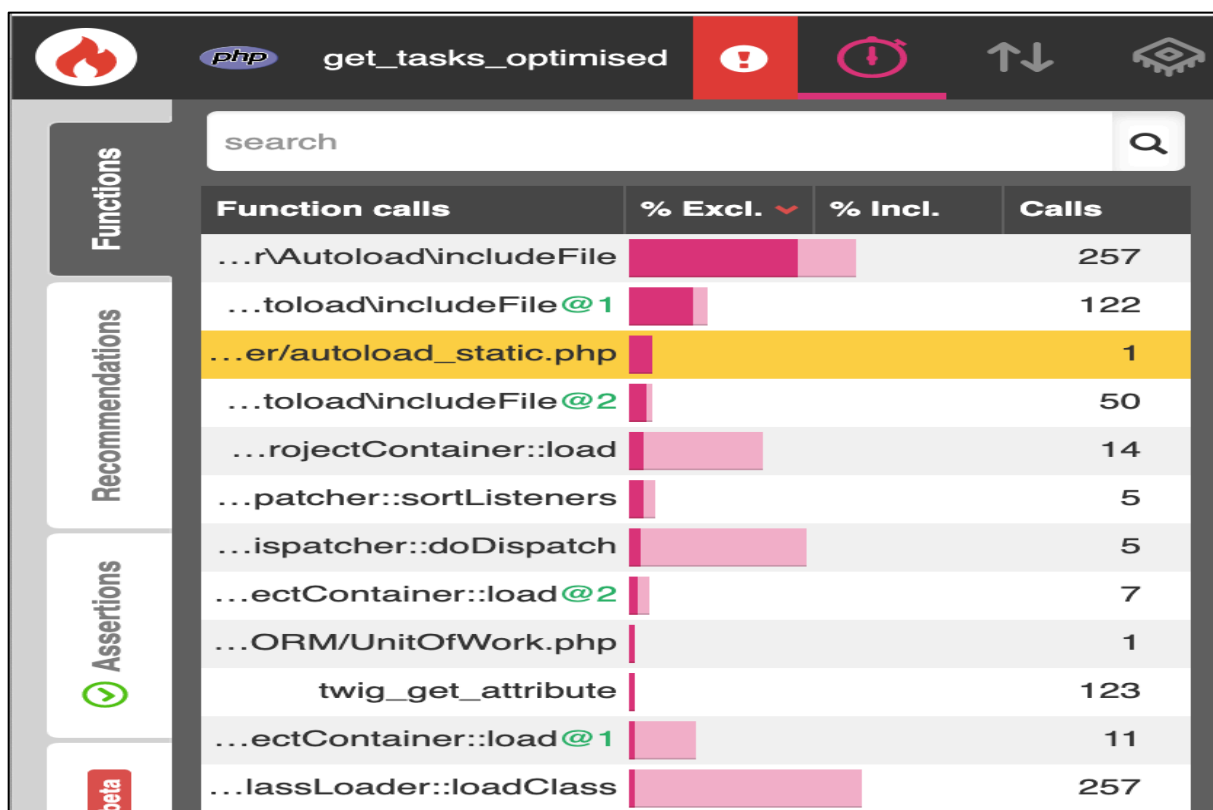
`vendor/composer/autoload_classmap.php`

Dorénavant à chaque fois qu'une classe est demandé, composer vérifiera le chemin nécessaire à partir de la carte de classe.

- **Commande :**

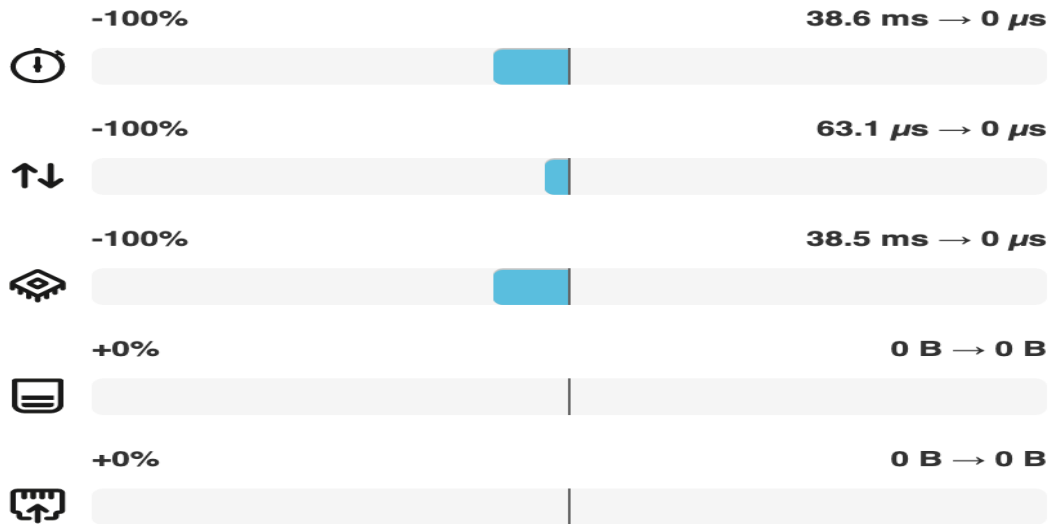
`$composer dump-autoload -o`

- **Résultat :**

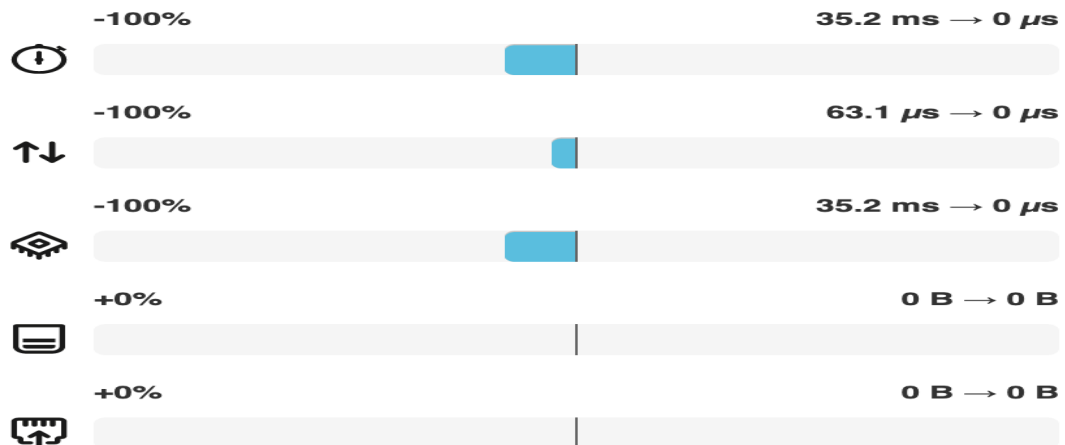


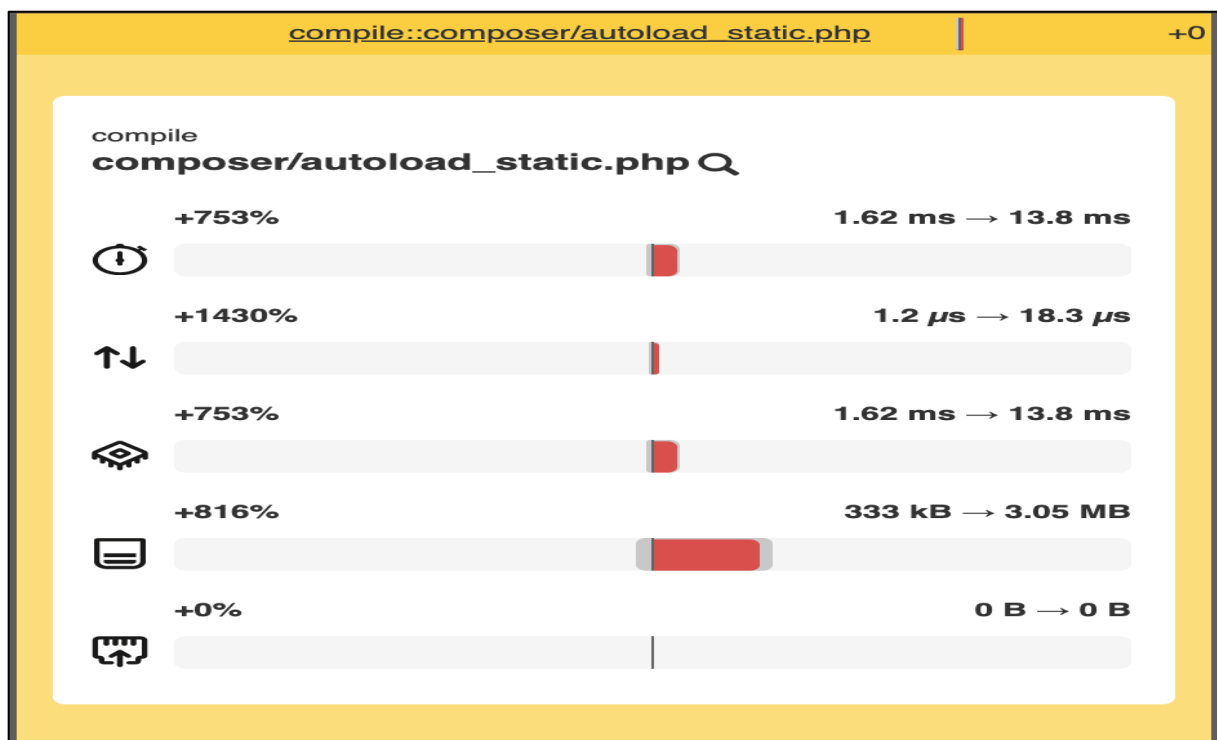
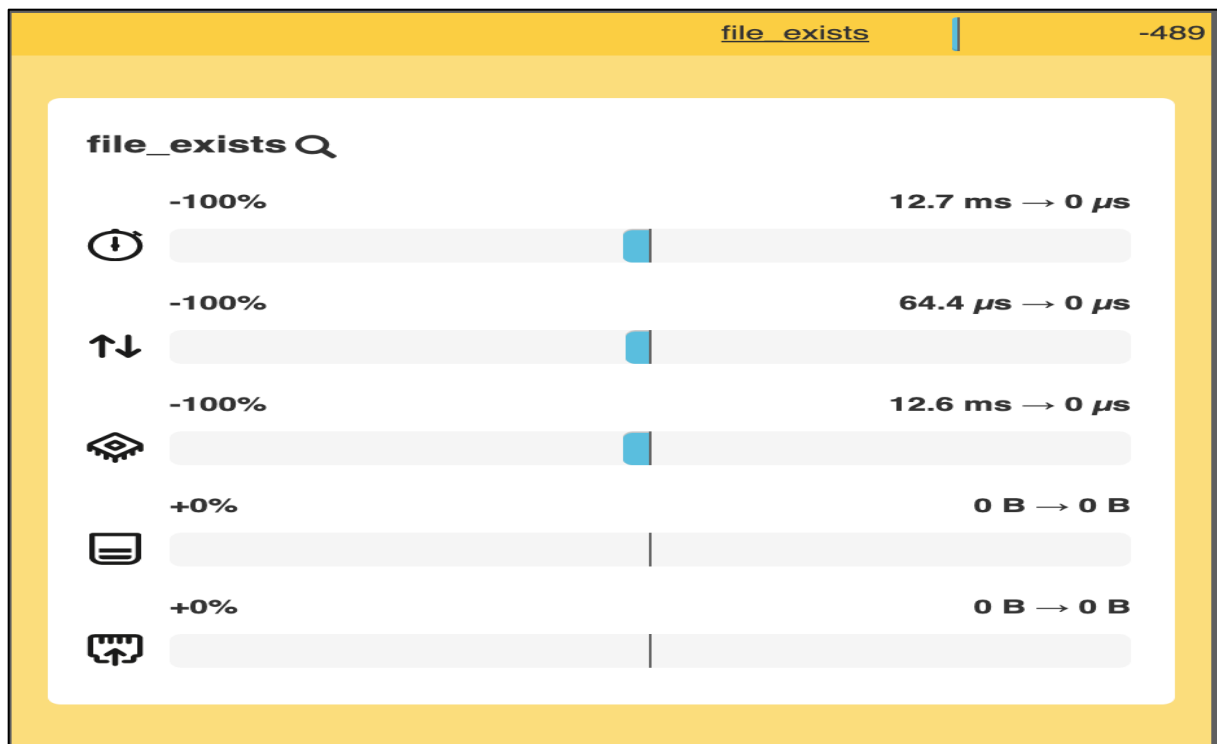
- **Comparaison de performance :**

Composer\Autoload\ClassLoader

findFile Q

Composer\Autoload\ClassLoader

findFileWithExtension Q



b. Optimisation possible des performances de PHP avec OPcache :

- **Problématique :**

L'exécution d'un script PHP passe par trois étapes :

- *Analyse syntaxique.*
- *Compilation en opcode (code compréhensible par la machine)*
- *Exécution de l'opcode*

- **Optimisation :**

La configuration de Opcache permet d'éviter l'étape répétitive de compilation en stockant le code initialement compilé en mémoire.

c. Optimisation possible de Doctrine :

- **Problématique :**

Nos entités peuvent avoir des relations entre elles. Par default ces relations sont configurées pour utiliser le « lazy loading » afin d'économiser les requêtes SQL, ceci veut dire que dans le cas ou en récupère une instance de l'entité « Task », l'instance de l'entité « User » attacher à elle n'est pas récupérer et il faudra faire une deuxième requête pour le récupérer s'il y a besoin.

- **Optimisation :**

Utiliser le « eager loading » pour récupérer les données :

Il faudra ajouter l'attribut « Fetch » avec en paramètre « Eager » afin que la relation soit récupérée en même temps que l'instance a laquelle elle est attaché.

```
public function listAction()
{
    return $this->render('task/list.html.twig',
        ['tasks' => $this->getDoctrine()->getRepository('AppBundle:Task')->findAll()]);
}
```

```

<div class="row">
  {% for task in tasks %}
    <div class="col-sm-4 col-lg-4 col-md-4">
      <div class="thumbnail">
        <div class="caption">
          <h4 class="pull-right">
            {% if task.isDone %}<span class="glyphicon glyphicon-ok"></span>{%
            else %}<span class="glyphicon glyphicon-remove"></span>{% endif %}
          </h4>
          <h4>
            <a href="{% path('task_edit', { id: task.id }) %}">{{
              task.title
            }}</a>
          </h4>
          <p>{{ task.content }}</p>
        </div>
        <div>
          <form action="{% path('task_toggle', { id: task.id }) %}">
            <button class="btn btn-success btn-sm pull-right">
              {% if not task.isDone %}Marquer comme faite{% else %}Marquer non
              terminée{% endif %}
            </button>
          </form>
          {% if task.user and task.user.id == app.user.id %}
            <form action="{% path('task_delete', { id: task.id }) %}">
              <button class="btn btn-danger btn-sm pull-right">Supprimer</button>
            </form>
          {% elseif not task.user and app.user.roles|filter(r=>r=="ROLE_ADMIN") %}
            <form action="{% path('task_delete', { id: task.id }) %}">
              <button class="btn btn-danger btn-sm pull-right">Supprimer</button>
            </form>
          {% endif %}
        </div>
      </div>
    </div>
  {% endfor %}
</div>

```

Dans l'exemple ci-dessus, nous avons une action qui récupère la liste des tâches. Ensuite la liste est envoyée au moteur de gabarit pour l'affichage. On remarque que dans l'affichage des tâches nous avons recours à une vérification du « user.id » liée à chaque tâche. Ce qui induit une requête supplémentaire pour récupérer ses données. Cette vérification peut avoir des conséquences sur la performance de l'application sur le long terme.