Rendimiento

Symfony es muy rápido por si mismo, pero podemos realizar algunos ajustes en nuestra aplicación y/o en nuestro servidor para lograr todavía mayor rendimiento.

Instalar APCu Polyfill si nuestro servidor utiliza APC

Si nuestro servidor de producción todavía utiliza la extensión de PHP APC en vez de OPCache, podemos instalar el componente APCu Polyfill para mejorar el rendimiento.

https://github.com/symfony/polyfill-apcu

Compilar el Service Container en un único fichero

Symfony compila por defecto el Service Container en muchos ficheros pequeños . Configurando el parámetro **container.dumper.inline_factories** a **true** se compila todo el Service Container en un único fichero lo que podría mejorar el rendimiento si se utiliza el **class preloading** de PHP 7.4 o versiones superiores:

```
# config/services.yaml
parameters:
    # ...
    container.dumper.inline_factories: true
```

OPcache

OPcache almacena los ficheros PHP compilados para evitar tener que compilarlos en cada petición.

En versiones antiguas de PHP se utilizaba APC, pero a partir de la versión PHP 5.5 se introdujo OPCache.

http://php.net/manual/es/opcache.installation.php

Configurar OPcache para máximo rendimiento

La configuración por defecto de OPCache no está ajustada para aplicaciones symfony, por lo que se recomienda cambiar la configuración con estos valores:

```
; php.ini
; maximum memory that OPcache can use to store compiled PHP files
opcache.memory_consumption=256
```

```
; maximum number of files that can be stored in the cache
opcache.max_accelerated_files=20000
```

No comprobar los Timestamps de los ficheros PHP

En los servidores de producción, los archivos PHP nunca deberían cambiar, excepto por actualizaciones de la aplicación. Sin embargo, por defecto OPCache comprueba si los ficheros han cambiado desde que se cachearon.

Para evitar esta sobrecarga innecesaria podemos configurar OPCache como sigue:

```
; php.ini
opcache.validate_timestamps=0
```

Después de cada deploy, habría que resetear la cache de OPCache. Hay varias opciones para hacer esto:

- Reiniciar el servidor web
- Lla mar a las funciones *apc_clear_cache()* o *opcache_reset()* a través del web server, por ejemplo a través de un script accesible desde la web.
- Utilizar las herramientas que nos ofrezca OPcache en el CLI.

Configurar OPCache con Preloading

A partir de PHP 7.4, OPcache puede compilar y cargar clases en el *start-up* y tenerlas disponibles para todas las peticiones hasta que el servidor se reinicie.

Symfony por su parte puede generar un fichero con la lista de clases que se necesitan precargar. Para ello deben estar a **true** las dos siguientes variables:

```
# config/services.yaml
parameters:
    # ...
    container.dumper.inline_factories: true
    container.dumper.inline_class_loader: true
```

En el php.ini habría que indicar la localización del fichero:

```
; php.ini
opcache.preload=/path/to/my/project/var/cache/prod/App_KernelProdContaine
r.preload.php
```

Configurar el PHP realpath Cache

Cuando un path relativo se transforma en un path real y absoluto, PHP cachea el resultado para mejorar el rendimiento. Las aplicaciones que abren muchos ficheros (como las aplicaciones Symfony) deberían tener como mínimo estos valores de configuración.

```
; php.ini
; maximum memory allocated to store the results
realpath_cache_size=4096K

; save the results for 10 minutes (600 seconds)
realpath_cache_ttl=600
```

Optimizar el Autoloader de Composer

La clase que gestiona la autocarga está pensada y optimizada para encontrar nuevas clases durante el desarrollo.

En producción, como los ficheros PHP no cambian excepto por actualizaciones de la aplicación, se puede optimizar el autoloader de composer generando una "class map" que no es más que un gran array de los lugares en los que se encuentra cada clase.

Ese *class map* se almacena en *vendor/composer/autoload_classmap.php*.

Con el siguiente comando, regeneremaos este fichero *class map*.

```
composer dump-autoload --optimize --no-dev --classmap-authoritative
```

Las opciones utilizadas en el comando son las siguientes:

--optimize realiza el dump en el fichero class map de todas las clases compatibles PSR-0 y PSR-4 utilizadas en nuestra aplicación.

--no-dev excluye del class map las clases que solamente se utilizan para desarrollo.

--classmap-authoritative configura Composer para no escanear el systema de ficheros cuando no encuentre una clase en el class map.