Introducción

Poner en producción una aplicación Symfony puede ser muy sencillo o muy complicado según las tareas y los requisitos de nuestro sistema.

Deployment básico

Los pasos básicos del deployment son los siguientes:

1. Subir el código del proyecto al servidor
2. Instalar las dependencias
3. Configurar las variables de entorno y establecer SYMFONY_ENV=prod
4. Crear la base de datos, sus tablas e insertar los datos que hagan falta.
5. Limpiar la cache.

Estos son los pasos básicos. Después, según como sea nuestro proyecto, quizás necesitemos realizar más tareas como añadir CRON jobs, subir assets al CDN…

Tareas post-deploy comunes

Aunque el tipo de tareas a realizar en la puesta en producción puede ser muy variado según el proyecto, vamos a repasar algunas de las tareas más comunes:

Comprobar requisitos del servidor

Symfony tiene un comando para comprobar si nuestro servidor cumple con los requisitos para ejecutar una aplicación Symfony.

> *php bin/symfony_requirements*

Instalar las dependencias

> *composer install --no-dev --optimize-autoloader*

La opción **–optimize-autoloader** mejora el rendimiento del autoloader de composer construyendo una “class map”.

La opción **–no-dev** nos asegura que no instalamos los paquetes de desarrollo en el servidor.

NOTA: Si lanzando el comando interior se produce un error “class not found”, es posible que necesitemos ejecutar

> *export SYMFONY_ENV=prod*

o

> *export APP_ENV=prod (si usamos Flex)*

para que los scripts post-install-cmd se ejecuten en el entorno de prod.

Limpiar la caché

Hay que acordarse de limpiar la cache del entorno de producción:

> *php bin/console cache:clear --env=prod --no-debug*

Cómo configurar Symfony con un balanceador de carga o un reverse proxy

Es posible que nuestra aplicación esté detrás de un balanceador de carga o de un reverse proxy.

En principio esto no afecta a una aplicación symfony. Sin embargo, cuando una request pasa a través de un proxy, cierta información se manda en cabeceras X-Forwarded-* .

Por ejemplo, en vez de leer la cabecera REMOTE_ADDR (que realmente tendrá la IP del proxy, deberíamos leer una cabecera X-Forwarded-For)

Si no configuramos Symfony para mirar esas cabeceras, tendremos información equivocada sobre la IP del cliente, el protocolo (http o https), el puerto y el host name.

setTrustedProxies()

La solución que nos brinda Symfony es el método setTrustedProxies() con el que le decimos en qué IPs confiamos (las de nuestros proxys) y qué cabeceras utilizar para enviar la información:

```
// public/index.php
```

```
// ...
$request = Request::createFromGlobals();

// tell Symfony about your reverse proxy
Request::setTrustedProxies(
    // the IP address (or range) of your proxy
    ['192.0.0.1', '10.0.0.0/8'],

    // trust *all* "X-Forwarded-*" headers
    Request::HEADER_X_FORWARDED_ALL

    // or, if your proxy instead uses the "Forwarded" head
er
    // Request::HEADER_FORWARDED

    // or, if you're using AWS ELB
    // Request::HEADER_X_FORWARDED_AWS_ELB
);
```