Installation et configuration de Nginx et PHP-FPM pour CakePHP

Guillaume LINUX / TUTORIELS, TUTORIELS 10 Comments

On continue sur mon projet de fin d'année qui définit la ligne conductrice du blog en ce moment !

Source : nginx.org

Après avoir mis en place la gestion des fichiers plats avec GlusterFS et préparé la manière dont nous allions répartir la charge sur les serveurs hébergeant l'API grâce au couple LVS / Keepalived, vient maintenant l'heure de préparer ces fameuses machines hébergeant l'API du projet. Je rappelle que cette API est développée avec le framework PHP CakePHP.

Il nous faut donc un serveur HTTP capable de supporter des pics de trafic plus ou moins important sans flancher. Et plutôt que de me tourner vers la solution « classique » Apache2 / PHP 5, j'ai décidé d'en profiter pour découvrir un autre serveur HTTP dont le web ne tarit par d'éloge en matière de consommation de ressources : j'ai nommé Nginx (prononcez « Engine X »).

L'objectif de cet article va donc être double : me permettre de comprendre pourquoi Nginx est plus rapide qu'Apache et consomme moins de ressource, et apprendre à le configurer de façon simple dans un premier temps pour qu'il puisse héberger une application développée en PHP.

Apache et son module PHP

Lorsque vous installez Apache2 sur une Debian classique, il sera très souvent livré avec un ensemble de modules de base dont PHP5 fait partie. Il s'appelle d'ailleurs libapache2-mod-php5 si je ne m'abuse. De fait, Apache2 et PHP deviennent assez indissociables et cela n'est pas sans poser quelques souci. En effet, de nombreuses librairies PHP ne sont pas *threadsafe* et il devient indispensable d'utiliser le mode prefork-mpm d'Apache 2 pour tenter de pallier à ce problème.

Dans la pratique, les administrateurs de sites à fort trafic vous diront que ce mécanisme entraîne une forte consommation de RAM / CPU de la part d'Apache, ce qui peut devenir délicat à traiter ou réparer… Pourquoi des problèmes d'I/O ? Tout simplement parce qu'Apache lance un processus PHP à chaque requête et le referme une fois la réponse envoyée.

L'idée va donc être de dissocier notre serveur HTTP de PHP pour que le serveur HTTP ne traite QUE les requêtes HTTP qu'il reçoit et redirige éventuellement les requêtes PHP (s'il en reçoit) vers le ou les processus PHP tournant sur notre machine de manière indépendante. Et c'est là que Nginx intervient. Pour la version « indépendante » de PHP, tout est déjà prévu et ça s'appelle PHP-FPM ! Ainsi, PHP sera chargé en mémoire et répondra aux requêtes qu'il recevra de Nginx, d'autant plus que PHP-FPM utilise des mécanismes lui permettant de réutiliser ses threads pour répondre à plusieurs requêtes (plutôt que d'ouvrir / fermer un thread à chaque requête).

C'est parti !

Installation et configuration des paquets

INSTALLATION DE NGINX ET PHP5

Sous Debian Wheezy 7.5 tous les paquets sont présents dans les dépôts :

```
sudo apt-get install nginx php5-fpm
```

CONFIGURATION DE PHP-FPM

Nginx va donc transmettre les requêtes PHP à PHP-FPM, mais comment ? Deux choix s'offrent à nous :

- PHP et Nginx tournent sur la même machine. Dans ce cas, et pour des raisons de performances, nous utiliserons un socket Unix.
- Nos deux outils ne sont pas sur la même machine, on se tourne vers un socket TCP.

Voilà pourquoi

Étant dans le premier cas, je dois configurer PHP-FPM pour qu'il écoute sur un socket Unix. Ça se passe dans le fichier de configuration `/etc/php5/fpm/pool.d/www.conf` :

```
;listen = 127.0.0.1:9000
listen = /var/run/php5-fpm.sock
```

Et on redémarre le service pour appliquer les modifications :

```
sudo service php5-fpm restart
```

CONFIGURATION DE NGINX

On commence par déclarer le socket Unix de PHP-FPM au niveau de Nginx pour que celui-ci puisse lui transmettre les requêtes PHP qu'il reçoit, via `/etc/nginx/conf.d/php5-fpm.conf` :

```
upstream php5-fpm-sock {
        server unix:/var/run/php5-fpm.sock;
}
```

Tout comme Apache, Nginx fonctionne avec un système de vhost (appelés Server Blocks) qui vont nous permettre de paramétrer nos différents hôtes virtuels. Je propose de commencer par un test simple en reprenant la configuration du vhost par défaut de Nginx.

Créez le fichier `/usr/share/nginx/html/index.php` suivant :

```
<?php phpinfo(); ?>
```

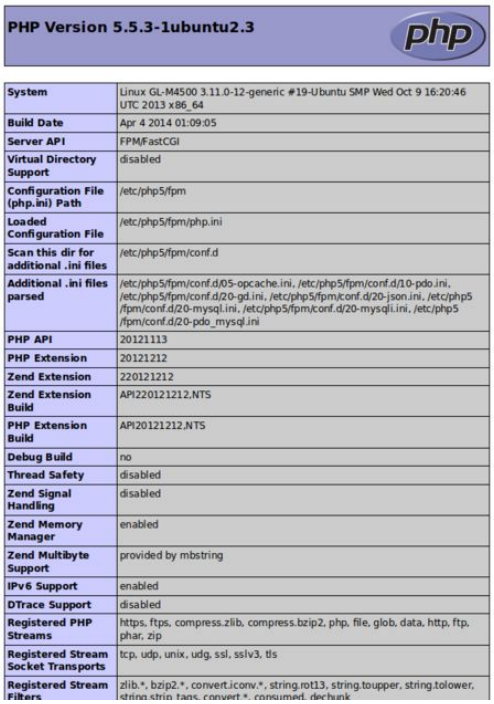
Et éditez le fichier de configuration `/etc/nginx/sites-available/default` pour qu'il ressemble à cela :

```
server {
        listen 80 default_server;

        root /usr/share/nginx/html;

        index index.php index.html;

        access_log /var/log/nginx/default-access_log;
        error_log /var/log/nginx/default-error_log;

        location / {
                try_files $uri $uri/ /index.php?$args;
        }

        location ~ \.php$ {
                try_files $uri =404;
                fastcgi_index index.php;
                fastcgi_pass php5-fpm-sock;
                fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
                include /etc/nginx/fastcgi_params;
        }
}
```

On redémarre le service :

```
sudo service nginx restart
```

L'accès à http://ip.de.mon.serveur/index.php devrait afficher quelque chose de similaire à :



PHP Version 5.5.3-1ubuntu2.3

System	Linux GL-M4500 3.11.0-12-generic #19-Ubuntu SMP Wed Oct 9 16:20:46 UTC 2013 x86_64
Build Date	Apr 4 2014 01:09:05
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/fpm
Loaded Configuration File	/etc/php5/fpm/php.ini
Scan this dir for additional .ini files	/etc/php5/fpm/conf.d
Additional .ini files parsed	/etc/php5/fpm/conf.d/05-opcache.ini, /etc/php5/fpm/conf.d/10-pdo.ini, /etc/php5/fpm/conf.d/20-gd.ini, /etc/php5/fpm/conf.d/20-json.ini, /etc/php5/fpm/conf.d/20-mysql.ini, /etc/php5/fpm/conf.d/20-mysqli.ini, /etc/php5/fpm/conf.d/20-pdo_mysql.ini
PHP API	20121113
PHP Extension	20121212
Zend Extension	220121212
Zend Extension Build	API220121212,NTS
PHP Extension Build	API20121212,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	https, ftps, compress.zlib, compress.bzip2, php, file, glob, data, http, ftp, phar, zip
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, tls
Registered Stream Filters	zlib.*, bzip2.*, convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk

Tout fonctionne à merveille !

Configuration d'un server block Nginx pour CakePHP

CakePHP propose un modèle de server block dans sa documentation. Pour les connaisseurs, l'idée est de faire pointer la directive Nginx « root » dans le dossier `App/webroot` comme suit (`/etc/nginx/sites-available/cakeapp` :

```
server {
        listen 80;

        server_name www.cakeapp.com;
        root /usr/share/nginx/cakeapp/app/webroot;

        index index.php index.html;

        access_log /var/log/nginx/cakeapp-access.log;
        error_log /var/log/nginx/cakeapp-error.log;

        location ~ \.php$ {
                try_files $uri =404;
                fastcgi_index index.php;
                fastcgi_pass php5-fpm-sock;
                fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
                include /etc/nginx/fastcgi_params;
        }

        location ~ /(\.ht|\.git|\.svn) {
                deny  all;
        }
}
```

> Je suis en phase d'apprentissage et je ne suis pas capable de vous expliquer tout cela en détail, je ne préfère pas dire de bêtise…

En tout cas il reste à créer un lien dans le dossier `/etc/nginx/sites-enabled` :

```
sudo ln -vs /etc/nginx/sites-available /etc/nginx/sites-enabled
```

Et à recharger Nginx :

```
sudo service nginx reload
```