La mise en place de Nginx avec PHP-fpm

Voici comment installer le serveur web Nginx ainsi que PHP avec le patch fpm, sur la distribution “Release 2” d’OVH en version 2.15, la dernière version disponible lors de la rédaction de ce document.

Par commodité, le nom **example.com** désignera une adresse IP associé à la machine sur laquelle l’installation est effectuée.

L’INSTALLATION DE NGINX

Le code source de Nginx peut être téléchargé à l’adresse : **http://nginx.net/**

Il existe principalement deux “branches” : les versions stables (actuellement 0.6.x) et les versions en développement (actuellement 0.7.x). Les versions de développement de Nginx sont généralement tout à fait utilisables en production et lorsqu’elles présentent des bogues majeurs, une nouvelle version arrive toujours dans les jours qui suivent pour les corriger.

La procédure qui suit prend pour exemple la version 0.7.22, mais elle s’appliquerait aussi à une version 0.6.x

La compilation et l’installation

On télécharge tout d’abord le code source que l’on décompresse :

```
wget http://sysoev.ru/nginx/nginx-0.7.22.tar.gz
tar xzvf nginx-0.7.22.tar.gz
cd nginx-0.7.22
```

Le serveur Nginx peut tourner sous n’importe quel utilisateur système. Par sécurité, on peut créer un nouvel utilisateur qui sera dédié à la consultation des éléments des sites web. On va par exemple créer un utilisateur **www** et un groupe associé : **www**. Cet utilisateur n’aura pas de shell et pas de répertoire de travail, sauf configuration explicite, il sera donc impossible de se connecter sous ce nom pour avoir une session FTP ou SSH.

```
groupadd www
useradd -d /dev/null -s /etc -g www www
```

Passons à la configuration et à la compilation, avec l'activation des modules les plus courants :

```
env CFLAGS=-O2 ./configure \
--prefix=/usr/local \
--user=www --group=www --with-http_sub_module --with-http_flv_module \
--with-http_gzip_static_module --with-http_stub_status_module \
--with-http_realip_module --http-log-path=/var/log/nginx/access.log \
--http-client-body-temp-path=/var/nginx/tmp/client_body_temp \
--http-proxy-temp-path=/var/nginx/tmp/proxy_temp \
--http-fastcgi-temp-path=/var/nginx/tmp/fastcgi_temp \
--conf-path=/etc/nginx/nginx.conf \
--error-log-path=/var/nginx/log/error.log \
--http-log-path=/var/nginx/log/access.log \
--pid-path=/var/run/nginx.pid --lock-path=/var/run/nginx.lock


make install
```

La post-installation

Et l'application devrait se trouver installée dans **/usr/local/sbin/nginx**, accompagnée de fichiers de configuration dans **/etc/nginx/**. Des fichiers temporaires, ainsi que les fichiers d'historiques (erreurs et accès) se trouveront dans **/var/nginx/**. On va créer les répertoires nécessaires, leur donner les bons droits d'accès et ajouter un lien vers les fichiers d'historiques dans **/var/log/nginx/** pour qu'ils soient facilement accessibles à partir du répertoire **/var/log** qui regroupe d'autres fichiers similaires :

```
mkdir -p /var/nginx/{tmp,log}
chown -R www:www /var/nginx
ln -s /var/nginx/log /var/log/nginx
```

Si Apache était démarré et programmé pour être relancé à chaque démarrage du système, il est temps de le désactiver :

```
rc-update del apache default
/etc/init.d/apache stop
```

Ajoutons maintenant un script de contrôle et de démarrage de Nginx, en recopiant ceci dans un fichier nommé **/etc/init.d/nginx** :

```
#!/sbin/runscript
opts="${opts} reload configtest"

depend() {
        need net
        use dns logger
}

start() {
        configtest || return 1
        ebegin "Starting nginx"
        start-stop-daemon --start --pidfile /var/run/nginx.pid \
                --exec /usr/local/sbin/nginx -- -c /etc/nginx/nginx.conf
        eend $? "Failed to start nginx"
}

stop() {
        configtest || return 1
        ebegin "Stopping nginx"
        start-stop-daemon --stop --pidfile /var/run/nginx.pid
        eend $? "Failed to stop nginx"
        rm -f /var/run/nginx.pid
}

reload() {
        configtest || return 1
        ebegin "Refreshing nginx' configuration"
        kill -HUP `cat /var/run/nginx.pid` &>/dev/null
        eend $? "Failed to reload nginx"
}

configtest() {
        ebegin "Checking nginx' configuration"
        /usr/local/sbin/nginx -c /etc/nginx/nginx.conf -t
        eend $? "failed, please correct errors above"
}
```

N'oublions pas de donner les droits d'exécution sur ce fichier :

```
chmod +x /etc/init.d/nginx
```

La configuration

Le principal fichier de configuration lu par Nginx se trouve dans **/etc/nginx/nginx.conf.** Le fichier **nginx.conf.dist** qui se trouve dans le même répertoire n'est là qu'à titre d'exemple.

Voici pour commencer ce que nous pouvons mettre dans le fichier **nginx.conf** :

```
user www;
worker_processes 2;
daemon on;
error_log /var/log/nginx/error.log warn;
timer_resolution 500ms;

events {
   worker_connections 512;
}

http {
   include       mime.types;
   default_type  text/plain;
   server_name_in_redirect on;
   server_names_hash_bucket_size 64;
   client_max_body_size 25M;
   client_body_buffer_size 128k;
   client_body_timeout 60;
   proxy_max_temp_file_size 25M;
   proxy_buffering on;
   proxy_read_timeout 100;

   access_log /var/log/nginx/access.log combined buffer=4k;
   log_not_found on;

   sendfile        on;
   tcp_nopush      on;
   tcp_nodelay     on;

   ignore_invalid_headers on;

   keepalive_timeout  65;
   send_timeout 300;

   autoindex off;
   msie_padding on;

   gzip on;
   gzip_comp_level 7;
   gzip_min_length 1000;
```

```
    gzip_types text/css text/plain text/javascript application/x-javas-
cript text/xml;
    gzip_disable msie6;
    gzip_vary on;
    gzip_http_version 1.1;
    gzip_proxied any;


    server {
        listen 80 default backlog=1024;
        server_name .example.com;
        index  index.html;
        root   /var/www/www/default;
    }
}
```

Ouhla, c'est long !

Pas de panique, il y a principalement des paramètres globaux. Ce qui concerne l'hébergement d'un site se résume aux quelques lignes de la section **server { ... }** qui se trouve à la fin.

La signification des différents paramètres est documentée sur le site de Nginx, mais pour résumer ceux qui se trouvent ici :

- **worker_process 2 :** c'est le nombre de processus Nginx qui vont traiter les requêtes. Le nombre idéal dépend beaucoup du site, de la façon dont il va être consulté, du nombre de disques présents sur le serveur et de leur agencement. Pour des machines avec au moins 2 cores et 2 disques comme la plupart des machines OVH, la valeur **2** est un bon choix.
- **timer_resolution 500 :** indique la précision des dates dans les fichiers d'historiques. Elle est ici d'une demi-seconde, ce dont on peut généralement très bien se contenter. Par défaut, la meilleur précision possible est choisie, ce qui est fort sympathique, mais consomme un peu de ressources CPU pour pas grand chose.
- **worker_connections 512 :** le nombre maximal de requêtes simultanées que le serveur va accepter.

Les paramètres au début de la section http concernent la taille des tampons (pour éviter les fichiers temporaires dans le cadre des communications entre PHP et Nginx), la longueur maximale d'un nom de site (**server_names_hash_bucket_size**) et le temps d'inactivité à partir duquel on considère qu'un script PHP est dans les choux (**proxy_read_timeout**).

- Les lignes concernant **gzip** servent à compresser à la volée les données qui peuvent l'être. Contrairement à Lighttpd, Nginx ne crée pas de fichiers de cache, mais la compression est très rapide.
  Internet Explorer 6 est sévèrement boggué lorsqu'il reçoit des données compressées. Cela se traduit en particulier par l'affichage de pages blanches bien que le contenu ait été correctement transféré. Pour éviter cela, il faut désactiver la compression pour ce navigateur. C'est le rôle de **gzip_disable msie6** qui est une directive de Nginx 0.7.x. Si Nginx 0.6.x est installé, on peut la remplacer par **gzip_disable "MSIE [1-6]\.(?!.*SV1)"**
- La partie **server** désigne la configuration d'un site, ici le site **example.com** :
  listen 80 indique le port à écouter pour ce site. Il est ici suivi de **default** qui précise qu'il s'agit du site par défaut du serveur ainsi que de l'option **backlog=1024** qui demande à Nginx de mettre jusqu'à 1024 connexions en attente dans le cas où le serveur aurait du mal à traiter les précédente aussi rapidement qu'il serait nécessaire. Ces deux dernières directives ne sont à préciser qu'une fois, pour le site par défaut uniquement. Pour tous les autres sites on peut se contenter de **listen 80** tout court.
  server_name .example.com : on indique ici le ou les noms par lesquels le site sera accesible. Un autre exemple serait :

server_name example.com example.net hebergement.example.net qui permettrait d'accéder au contenu via http://example.com, http://example.net et http://hebergement.example.net . La notation **.example.com** (point au début) est une convention de Nginx pour désigner "**example.com** tout court, mais aussi <**quelque chose**>**.example.com**). Cela comprend http://example.com et http://www.example.com .

index index.html précise quels sont les fichiers qui seront recherchés dans un répertoire s'ils ne sont pas explicitement communiqués dans l'URL. Pour en mettre plusieurs, qui seront testés dans l'ordre d'apparition, il suffit de les séparer d'une espace : **index index.html index.htm index.php**

root /var/www/www/default est le répertoire qui contient les fichiers du site qui doivent être servis.

Bien d'autres directives peuvent être ajoutées dans chaque section **server**, en particulier **error_log** et **access_log** afin d'avoir des fichiers d'historiques distincts pour chaque site.

Avant d'aller plus loin, testons si le serveur et le script de contrôle sont bien en place :

```
/etc/init.d/nginx configtest
```

La commande devrait afficher en retour quelque chose comme :

```
 * Checking nginx' configuration ...
2008/09/07 23:58:36 [info] 32505#0: the configuration file
/etc/nginx/nginx.conf syntax is ok
2008/09/07 23:58:36 [info] 32505#0: the configuration file
/etc/nginx/nginx.conf was tested successfully
```

En se connectant à l'aide d'un navigateur web sur une adresse du serveur (disons http://example.com) une page d'erreur devrait de Nginx s'afficher. C'est certes une page d'erreur (normal : il n'y a encore aucun fichier à servir) mais c'est pourtant signe que jusqu'ici tout va bien !

C'est néanmoins frustrant alors ajoutons un fichier pour vérifier que Nginx le trouve bien :

```
mkdir -p /var/www/www/default
cd /var/www/www/default

echo Test > /var/www/www/default/index.html
```

En se connectant à nouveau sur http://example.com on devrait maintenant avoir 4 magnifiques lettres qui s'illuminent.

Tout est bon ? Alors ajoutons PHP.

LA MISE EN PLACE DE PHP-FPM

PHP est déjà installé sur la Release 2 OVH et d'une façon non standard, spécifique à OVH.

Afin de conserver la cohérence du système, on va faire le maximum pour que la nouvelle version de PHP respecte les tous choix techniques de la Release 2., même si cela complique beaucoup les choses par rapport à une installation plus traditionnelle de PHP.

Les prérequis

La Release 2, du moins en version 64 bits, contient un bogue bloquant pour la compilation de certains logiciels, dont PHP. Voici la commande permettant de le corriger :

```
emerge --sync ; emerge binutils binutils-config
```

Ceci fait, nous allons nous munir du patch PHP-fpm disponible à cette adresse :
http://php-fpm.anight.org/download.html

```
wget http://php-fpm.anight.org/downloads/head/php-5.2.6-fpm-0.5.8.diff.gz
```

Évidemment il faut aussi télécharger le code source de PHP, à cette adresse : **http://www.php.net/downloads.php#v5**

```
wget http://fr.php.net/distributions/php-5.2.6.tar.bz2
```

On décompresse l'archive de PHP ainsi téléchargée et on y applique le patch fpm :

```
tar xjvf php-5.2.6.tar.bz2
```

```
cd php-5.2.6
gunzip -c ../php-5.2.6-fpm-0.5.8.diff.gz | patch -p1
```

Ceci fait, on va chercher la configuration du PHP 5 actuellement installé (celui d'origine d'OVH), qui va servir de base à celle de notre version de PHP.

Afin d'obtenir cette configuration, la commande suivante peut être d'un grand secours :

```
php5 -i | lynx -stdin
```

Cette commande affiche quelque chose comme ce qui suit :

```
'./configure' '--disable-cli' '--disable-discard-path' \
'--disable-force-cgi-redirect' '--prefix=/usr/local/php5' \
'--with-config-file-path=/usr/local/lib/php5' '--with-pear=/usr/share/php5' \
'--enable-exif' '--enable-ftp' '--enable-bcmath' '--enable-calendar' \
'--with-gd' '--enable-gd-native-ttf' '--with-freetype-dir' '--with-gettext' \
'--with-zlib-dir' '--with-imap' '--with-imap-ssl' '--with-png-dir=/usr' \
'--with-jpeg-dir=/usr' '--with-xpm-dir=/usr' '--with-openssl' \
'--with-kerberos' '--enable-sysvsem' '--enable-sysvshm' '--with-mcrypt' \
'--with-iconv' '--enable-mbstring=all' '--enable-mbregex' \
'--with-mysql=/usr' '--with-mysqli' '--with-curl' '--with-xsl'
```

Il est nécessaire de modifier le début pour PHP-fpm :

```
'./configure' '--enable-fastcgi' '--disable-cli' '--disable-discard-path' \
'--enable-fpm' '--with-zend-vm=GOTO' '--enable-force-cgi-redirect' \
'--prefix=/usr/local/php5' \
'--with-config-file-path=/usr/local/lib/php5' '--with-pear=/usr/share/php5' \
'--enable-exif' '--enable-ftp' '--enable-bcmath' '--enable-calendar' \
'--with-gd' '--enable-gd-native-ttf' '--with-freetype-dir' '--with-gettext' \
'--with-zlib-dir' '--with-imap' '--with-imap-ssl' '--with-png-dir=/usr' \
'--with-jpeg-dir=/usr' '--with-xpm-dir=/usr' '--with-openssl' \
'--with-kerberos' '--enable-sysvsem' '--enable-sysvshm' '--with-mcrypt' \
'--with-iconv' '--enable-mbstring=all' '--enable-mbregex' \
'--with-mysql=/usr' '--with-mysqli' '--with-curl' '--with-xsl'
```

C'est cette dernière commande qui est à taper dans le répertoire contenant le code source de PHP-fpm. La partie **--with-zend-vm=GOTO** est un excès de zèle qui n'est pas indispensable pour PHP-fpm mais qui rend l'exécution des scripts PHP un peu plus rapide sans prendre beaucoup de risques.

La configuration effectuée, la compilation et l'installation peuvent s'effectuer :

```
make -j3 install
```

La post-configuration

Le fichier de configuration de PHP-fpm est situé dans un endroit un peu tordu (**/usr/local/php5/etc/php-fpm.conf**). C'est une question de goût, mais y accéder en tant que **/etc/php-fpm.conf** n'est pas désagréable. On peut donc faire un petit lien symbolique, ça ne mange pas de pain :

```
ln -s /usr/local/php5/etc/php-fpm.conf /etc/
```

Le fichier d'historique de PHP-fpm est aussi dans un endroit tordu, que l'on peut remplacer par **/var/log/php-fpm.log** :

```
rm -f /usr/local/php5/logs/php-fpm.log
ln -s /var/run/php-fpm.log /usr/local/php5/logs/
```

Que l'on utilise son emplacement d'origine ou le lien **/etc/php-fpm.conf**, il faut désormais éditer le contenu de ce fichier. C'est rude, c'est moche, ça ressemble à du XML, mais c'est la vie.

Les lignes à modifier sont :

```
<value name="pid_file">/var/run/php-fpm.pid</value>
<value name="error_log">/var/log/php-fpm.log</value>
...
<value name="user">www</value>
<value name="group">www</value>
<value name="max_children">50</value>
```

max_children correspond au nombre maximal de scripts PHP qui pourront tourner simultanément. Si cette valeur est atteinte, les connexions seront simplement mises en attente, le temps que des places se libèrent. Il s'agit réellement du nombre de scripts PHP en fonctionnement, en même temps. Cette valeur est quasiment toujours largement inférieure au nombre de clients simultanés (qui vont charger le contenu du script PHP, mais aussi les images, CSS, javascripts, etc. Lors du chargement de ces derniers, ils n'occupent plus de PHP). **50** est déjà une valeur assez large même pour un site assez fréquenté.

Inutile d'écrire un script de démarrage, il en existe déjà un, il faut juste le mettre à l'endroit attendu pour une Release 2 :

```
ln -s /usr/local/php5/sbin/php-fpm  /etc/init.d/
```

Et maintenant, démarrons les serveurs PHP :

```
/usr/local/php5/sbin/php-fpm  start
```

Vérifions si tout s'est bien passé :

```
pstree
```

La commande doit afficher parmi les processus :

```
      |-nginx---2*[nginx]
      |-php-cgi---50*[php-cgi]
```

Ajoutons un fichier PHP à notre site d'exemple :

```
echo '<?php phpinfo(); ?>' > /var/www/www/default/index.php
```

En accédant à **http://example.com/index.php** on obtient donc... eeeuuuuuhhhh ? Le code source PHP ? Bigre !

Certes, les serveurs PHP sont à l'écoute, mais Nginx ne le sait pas. Nous allons donc modifier légèrement la configuration de Nginx à cet effet.

Plus exactement, dans le fichier **/etc/nginx/nginx.conf**, dans la section **server** les lignes suivantes vont être ajoutées :

```
location ~ \.php(/|$) {
   include fastcgi_params;
}
```

Ce qui nous donne quelque chose comme ce qui suit pour la section **server** :

```
server {
   listen 80 default backlog=1024;
   server_name .example.com;
   index  index.php index.html;
   root   /var/www/www/default;
   location ~ \.php(/|$) {
      include fastcgi_params;
   }
}
```

index.php ayant été ajouté au passage au fichiers par défaut via la directive **index**, on pourra désormais accéder au script **index.php** en tapant simplement http://example.com

location ~ \.php(/|$) { ... } signifie que ce qui se trouve entre les accolades s'applique à tous les fichiers dont l'extension est **.php**

include sert à insérer le contenu d'un fichier à cet endroit du fichier de configuration. Si l'on héberge qu'un seul site, ce n'est pas indispensable. Mais dès le second site, cette astuce permet d'éviter les copier-coller et les fichiers de configuration inutilement longs.

Et que contient donc ce fichier **/etc/nginx/fastcgi_params** ? Et bien par exemple pour une configuration standard :

```
fastcgi_connect_timeout 60;
fastcgi_send_timeout 300;
fastcgi_buffers 4 32k;
fastcgi_busy_buffers_size 32k;
fastcgi_temp_file_write_size 32k;

fastcgi_pass   127.0.0.1:9000;
fastcgi_index  index.php;
fastcgi_param  SCRIPT_FILENAME $document_root$fastcgi_script_name;
fastcgi_ignore_client_abort on;
fastcgi_intercept_errors on;
fastcgi_read_timeout 300;

fastcgi_param  QUERY_STRING       $query_string;
fastcgi_param  REQUEST_METHOD     $request_method;
fastcgi_param  CONTENT_TYPE       $content_type;
fastcgi_param  CONTENT_LENGTH     $content_length;

fastcgi_param  SCRIPT_NAME        $fastcgi_script_name;
fastcgi_param  REQUEST_URI        $request_uri;
```

```
fastcgi_param  DOCUMENT_URI       $document_uri;
fastcgi_param  DOCUMENT_ROOT      $document_root;
fastcgi_param  SERVER_PROTOCOL    $server_protocol;

fastcgi_param  GATEWAY_INTERFACE  CGI/1.1;
fastcgi_param  SERVER_SOFTWARE    nginx/$nginx_version;

fastcgi_param  REMOTE_ADDR        $remote_addr;
fastcgi_param  REMOTE_PORT        $remote_port;
fastcgi_param  SERVER_ADDR        $server_addr;
fastcgi_param  SERVER_PORT        $server_port;
fastcgi_param  SERVER_NAME        $server_name;

fastcgi_param  REDIRECT_STATUS    200;
```

Ce fichier a été crée, le fichier **nginx.conf** a aussi été mis à jour, il ne reste plus qu'à redémarrer Nginx pour que celui-ci communique désormais avec les serveurs PHP :

```
/etc/init.d/nginx restart
```

Désormais, se connecter à **http://example.com** devrait être beaucoup plus chouette que précédemment.

Bonus : l'installation d'APC

Sans "cache d'opcodes", PHP analyse le code source des scripts à chaque appel. Une opération un peu longue, bien plus que celle qui consisterait à évaluer le code sous forme prémâchée.

C'est pour cette raison qu'il existe des modules qui vont conserver en mémoire le code compilé des scripts, rendant le démarrage des scripts bien plus rapide.

Voici comment installer l'un d'eux, APC. La façon la plus simple est de taper la commande suivante :

```
/usr/local/php5/bin/pecl install apc
```

Lorsqu'une question concernant **apxs** est posée, il ne faut pas hésiter à répondre fermement pas la négative :

```
Use apxs to set compile flags (if using APC with Apache)? [yes] : no
```

Ensuite, PHP doit être informé de la présence de ce module, en éditant le fichier **/usr/local/lib/php5/php.ini**, plus exactement en changeant la ligne :

```
extension_dir = "./"
```

par:

```
extension_dir =
"/usr/local/php5/lib/php/extensions/no-debug-non-zts-20060613"
```

Ceci pour informer PHP du répertoire contenant les modules. Pour intégrer celui qui concerne APC, la ligne suivante peut être ajoutée n'importe où dans le fichier :

```
extension=apc.so
```

On relance ensuite les serveurs PHP :

```
/etc/init.d/php_fpm restart
```

En se connectant sur **http://example.com** la page d'information de PHP doit toujours s'afficher comme précédemment, mais maintenant ornée de mentions concernant la présence d'**APC**.

Bonus : l'installation de Suhosin

Suhosin est une extension permettant d'améliorer sensiblement la sécurité de PHP.

Elle se télécharge à l'adresse suivante : **http://www.hardened-php.net/suhosin/download.html**

Les commandes suivantes servent ensuite à le compiler et à l'installer :

```
wget http://download.suhosin.org/suhosin-0.9.27.tgz
tar xzvf suhosin-0.9.27.tgz
/usr/local/php5/bin/phpize
./configure --enable-suhosin
--with-php-config=/usr/local/php5/bin/php-config
gmake install
```

Et pour l'activer dans PHP, on édite comme précédemment le fichier **/usr/local/lib/php5/php.ini**, en y ajoutant juste cette ligne :

```
extension=suhosin.so
```

Et comme précédemment, redémarrer les serveurs PHP sert à recharger cette configuration :

```
/etc/init.d/php-fpm restart
```

Désormais, la page qui s'affiche à l'adresse **http://example.com** doit faire mention de Suhosin.

Le démarrage automatique de Nginx et PHP

Tout semble fonctionner ? Alors on peut indiquer au système de lancer automatiquement Nginx et PHP après un redémarrage du serveur.

Les commandes suivantes réalisent cette opération :

```
rc-update add php-fpm default
rc-update add nginx default
```

LA CONFIGURATION AVANCÉE DE NGINX

Voici quelques exemples de ce que l'on peut mettre dans la section **server** de la configuration de Nginx pour réaliser des opérations courantes.

Interdire la consultation de certains fichiers ou répertoires

Si l'on souhaite interdire à Nginx de servir certains fichiers ou répertoires, par exemple tout ce qui contient **.svn** voici les lignes à ajouter :

```
location ~ \.svn {
   deny all;
}
```

Ce qui suit **location ~** est une expression régulière. Pour une expression exacte, **~** doit être remplacé par **=** :

```
location = /admin/README {
   deny all;
}
```

Servir une image vide

Il arrive encore fréquemment que l'on ait besoin d'une image de "pixel transparent". Nginx peut servir une telle image rapidement sans passer par le chargement d'un fichier :

```
location = /clear.gif {
   empty_gif;
   expires max;
}
```

Ici, c'est l'adresse **http://example.com/clear.gif** qui mène à l'image transparente.

Servir des vidéos FLV (Flash)

Il n'est pas indispensable d'installer un serveur tel que FMS, Demou, Haxevideo ou Red5 pour servir des vidéos FLV via Flash en offrant à l'utilisateur la possibilité de s'y déplacer dans le temps sans la charger dans son intégralité. Nginx sait réaliser cette opération, à l'aide de :

```
location ~* \.flv$ {
   flv;
}
```

Ce qui suit **location ~*** est une expression régulière, tout comme **location ~**, la seule différence entre les deux étant que **~*** ne fait pas de distinction entre les majuscules et les minuscules.

Seront donc servies comme des vidéos Flash aussi bien **video.flv** que **MA_VIDEO.FLV**.

Le temps d'expiration des éléments statiques

Afin d'éviter de recharger systématiquement tous les éléments composant une page web, ou au contraire de forcer leur mise à jour rapide, il est possible par l'intermédiaire de la configuration de Nginx de préciser au navigateur combien de temps garder en cache les différents éléments servis.

Voici un exemple :

```
location ~ ^/(clientscript|images)/ {
   expires 1h;
}
location /events {
   expires 5s;
}
location ~* \.(css|swf)$ {
   expires 15m;
}
```

Ici, on indique aux navigateurs de conserver dans leur cache pendant une heure tout ce qui se trouve dans **http://example.com/clientscript/** et **http://example.com/images/** tandis que les fichiers dont l'extension est **.css** et **.swf** seront conservés 15 minutes. Quant aux fichiers qui se trouve dans **http://example.com/events/** , ils seront rechargés après un délai maximal de 5 secondes.

La consultation de l'état du serveur

Il est toujours utile de pouvoir consulter l'état du serveur, qui comprend entre autres la liste des clients actuellement connectés ainsi que le détail de leur activité.

Si l'on souhaite pouvoir accéder à ces information à l'adresse **http://example.com/nginx-status**, rien de plus simple :

```
location = /nginx-status {
   stub_status on;
}
```

Définir une page d'erreur 404 personnalisée

Plutôt que la page d'erreur 404 par défaut de Nginx, on peut choisir de servir autre chose, par exemple ce qui se trouverait à l'adresse **http://example.com/errors/404.html** :

```
error_page 404 /errors/404.html;
```

Utiliser le PATH_INFO

Certains scripts PHP peuvent se baser sur la variable **PATH_INFO** pour fournir des URLs censées améliorer le référencement.

Voici comment prendre en compte cette variable :

```
location ~ \.php(/|$) {
   include fastcgi_params;
   set $path_info "";
   if ($uri ~ \.php/.) {
       set $path_info $uri;
   }
   fastcgi_param PATH_INFO $path_info;
}
```

Éviter le hotlinking

Rien de plus frustrant que de voir d'autres sites inclure sans permission des éléments (images, vidéos) dans les pages de leurs sites au lieu de les envoyer à partir de leurs propres serveurs. Nginx permet de s'en protéger :

```
location /galerie/ongles-nail-art/ {
   valid_referers none blocked *.example.com;
   expires 7d;
}
location /galerie/photos-nail-art/ {
   valid_referers none blocked *.example.com;
   expires 15d;
}
if ($invalid_referer) {
   return 403;
}
```

Sur cet exemple, outre le fait que les images contenues dans **/galerie/ongles-nail-art/** et **/galerie/photos-nail-art/** soient conservées en cache dans le navigateur du client pendant 7 et 15 jours, on autorise leur chargement uniquement à partir de pages des sites ***.example.com** .

Si ces images tentent d'être chargées à partir d'une page d'un autre site, une erreur **403** est renvoyée en guise de contenu.

Cette protection est assez limitée, ne serait-ce-que parce que certains navigateurs et firewalls n'envoient pas l'information permettant de savoir à partir de quelle page un élément a été chargé. Mais son coût en charge CPU est imperceptible alors pourquoi s'en priver ?

La réécriture d'adresses

Grâce à la directive **rewrite** on peut faire correspondre une adresse à un répertoire différent de celui auquel elle serait logiquement associée. C'est l'équivalent de **mod_rewrite** d'Apache.

Voici quelques exemple :

```
rewrite ^/photo.html$ /pic.html break;
```

Accéder à http://example.com/photo.html chargera le même contenu que http://example.com/pic.html

```
rewrite ^/$ /galerie/galerie2.php break;
```

Accéder à http://example.com/ chargera http://example.com/galerie/galerie2.php

```
rewrite ^(.*)$ http://www.shopesthetic.fr$1 permanent;
```

Accéder à http://example.com/abc/xyz.html redirigera le navigateur (et les moteurs de recherche) vers un autre site : http://www.shopesthetic.fr/abc/xyz.html.

```
rewrite ^/([^/])([^/])([^/]+)\.jpg$ /$1/$2/$1$2$3-thumb.jpg;
```

Accéder à http://example.com/voiture.jpg chargera http://example.com/v/o/voiture-thumb.jpg

```
rewrite ^/([^/]+)/(\d+)$ /show/galerie2.php?username=$1&page=$2 break;
```

Accéder à http://example.com/robert/12 chargera http://example.com/galerie2.php?username=robert&page=12

La documentation de Nginx, en particulier la section "Cookbook", détaille le fonctionnement de cette directive, accompagné d'autres exemples qui peuvent être utiles.

La conversion de règles en provenance d'Apache ne pose en principe aucun soucis. Par exemple, le logiciel Prestashop est livré avec un fichier **.htaccess** avec ces règles pour Apache :

```
# URL rewriting module activation
RewriteEngine on

# URL rewriting rules
RewriteRule ^([0-9]+)-([a-zA-Z0-9-]*)\.html(.*)$ /product.php?id_product=$1$3 [L,E]
RewriteRule ^([0-9]+)-([a-zA-Z0-9-]*)(.*)$ /category.php?id_category=$1$3 [L,E]
RewriteRule ^([0-9]+)__([a-zA-Z0-9-]*)(.*)$ /supplier.php?id_supplier=$1$3 [L,E]
RewriteRule ^([0-9]+)_([a-zA-Z0-9-]*)(.*)$ /manufacturer.php?id_manufacturer=$1$3 [L,E]
```

Leur équivalent sous Nginx serait :

```
rewrite ^/([0-9]+)\-([a-zA-Z0-9-]*)\.html(.*)$ /product.php?id_product=$1$3 break;
rewrite ^/([0-9]+)\-([a-zA-Z0-9-]*)(.*)$ /category.php?id_category=$1 break;
rewrite ^/([0-9]+)__([a-zA-Z0-9-]*)(.*)$ /supplier.php?id_supplier=$1$3 break;
rewrite ^/([0-9]+)_([a-zA-Z0-9-]*)(.*)$ /manufacturer.php?id_manufacturer=$1$3 break;
```

à mettre dans la section **server** du site concerné.