Tim Reeves
Internet Services

Ubuntu 12.04 + Plesk 11.5/12: Nginx + PHP-FPM

Inhaltsverzeichnis

Tim Reeves Internet Services
Flurstr. 10
D-85221 Dachau

Tel. 08131 273 529
Fax. 08131 273 654
Ust-IdNr. DE269216307

Einleitung, Zusammenhänge

Nginx (Gesprochen: Engine X) stammt aus Russland und wurde als Webserver und Reverse Proxy geschrieben, ausgelegt für Lastverteilung, Schnelligkeit und niedrigem Speicherbedarf. Im Vergleich zum bisherigen Platzhirsch Apache – das zwar sehr viel kann, doch meist *viel mehr* als man braucht – ist Nginx schlank und leicht handhabbar – und unter dem Strich erheblich schneller.

Aufgabe eines Webservers

Generell liefert ein Webserver die aus dem Web angeforderten Resourcen aus – meist sind das Dateien (Bilder, Webseiten, Programme, Zips, PDFs usw.) aber auch Streaming-Media (Audio, Video) oder gar Kontakte und Termine (mit Hilfe entsprechender Subsysteme).

Oft aber ist die geforderte Resource eine Programmdatei, die auf dem Server ausgeführt werden muss, und dessen Ergebnis als Antwort zu schicken ist. Webserver indes führen keine Programmiersprachen selber aus, sondern delegieren diese Aufgabe an ein entsprechendes Subsystem („Interpreter"); dies ist z.B. bei PHP-Dateien, aber auch bei anderen Programmiersprachen wie Perl oder Python der Fall. Solchen Subsysteme nennt man im Fachjargon „Händler".

Wie ein Webserver Programmdateien ausführt

Hinzu kommt, dass die Händler grundsätzlich (bei Apache jedenfalls) auf unterschiedlicher Weise in den Webserver eingebunden und angesprochen werden können:

- Als Modul im Webserver-Programm eingebettet: bei Apache „mod_php"
  Der PHP-Interpreter wird damit zwangsmäßig immer mit der Benutzerkennung (und Rechte) des Webserver-Programms ausgeführt. Vorteil: Schnell[1], weil alles im gleichen Prozess abläuft („kurze Kommunikationswege").
- Common Gateway Interface (CGI) – eine standardisierte Schnittstelle zu anderen Prozessen (Interpreter-Programme) die vom Webserver Aufträge erhalten und Ergebnisse an ihn zurückliefern. Da sie als eigenständige Prozesse laufen können sie (jeweils) unter passenden Benutzerkennungen laufen, d.h. sicherer und bequemer. Allerdings: Bei CGI wird pro Auftrag ein Prozess zu seiner Abarbeitung gestartet (der nach der Abarbeitung wieder verschwindet) – keine effiziente Lösung.
- Über die FastCGI-Schnittstelle – eine Weiterentwicklung von CGI.
  Im Gegensatz zu CGI bleibt bei FastCGI ein Abarbeitungsprozess über viele Aufträge hinweg aktiv – anstatt einen neuen Prozess zu starten übergibt der Webserver den Auftrag an einen bereits laufenden Prozess. Es können mehrere solche Abarbeitungsprozesse nebeneinander vorhanden sein, und wie bei CGI können sie je unter passenden Benutzerkennungen laufen.

[1] Schneller als Nginx + PHP-FPM – **wenn** man auf .htaccess Dateien verzichten kann. Das geht meist aber nur dann, wenn man einen Server für sich hat...

Bei dem Plesk-Hersteller Parallels gibt es einen sehr übersichtlichen Artikel über die verschiedenen PHP-Händler mit einer Tabelle ihrer Vor- und Nachteile.

Wie Nginx Programmdateien ausführt: PHP-FPM

Für Nginx gibt es kein Äquivalent für Apache's mod_php. Der einzig verbreitete Weg für PHP-Ausführung unter Nginx ist mittels der FastCGI-Schnittstelle. Auch dafür stellt die Nginx-Gemeinschaft keine Eigenentwicklung bereit, sondern stützt sich auf PHP-FPM: PHP FastCGI Process Manager. Dieses Open-Source Programm stammt ebenfalls aus Russland und ist seit 2010 ein „experimenteller“ Teil von PHP-Distributionen, seit Ende 2011 ohne Einschränkungen.

Dementsprechend:

- Man kann auch „nur“ Nginx installieren bzw. einsetzen, so dass Nginx alle statischen Resourcen selber ausliefert und Programmdateien an Apache weiterreicht;
- Oder aber man installiert zusätzlich PHP-FPM und konfiguriert die beiden zur Zusammenarbeit – so dass Nginx die PHP-Anfragen nicht an Apache sondern an PHP-FPM weiterreicht. In diesem Fall wird Apache gar nicht mehr verwendet.

Ladezeit / Auslieferungsgeschwindigkeit

Die Gesamtzeit die verstreicht, bis eine Webseite im Browser vollständig dargestellt wird, ist sowohl für Besucher als auch für Suchmaschinen inzwischen ein wichtiger Faktor. Dies setzt sich i.W. aus der Auslieferungsgeschwindigkeit des Servers, sowie die Komplexität der gelieferten Inhalte zusammen. Eingebundene Werbung anderer Anbieter kann dieses Gesamtergebnis „bis Ruhe einkehrt“ erheblich belasten!

Daher ist es wichtig, die Wirksamkeit der Tuning-Maßnahmen objektiv zu messen; hierfür bietet Google ein sehr gutes Analysewerkzeug an.

Stufen des Performance-Tunings

Wieviel kann man machen, wie aufwendig sind die Maßnahmen durchzuführen, was lohnt sich in einer bestimmten Situation?

1: Nginx als Durchreicher vor dem Apache-Webserver stellen

Hierfür muss nur Nginx installiert werden; Plesk konfiguriert das Gesamtsystem darauf hin automatisch so, dass alle Anforderungen zunächst an Nginx gehen, der sie an Apache zur Abarbeitung durchreicht. Die Auslieferung übernimmt wiederum Nginx. Klingt spanisch, doch der Plesk-Hersteller Parallels erklärt in einem sehr verständlichen Artikel die damit erzielte Performance-Verbesserung.
Mein subjektiver Eindruck davon ist äußerst positiv: Plötzlich lieferte der Server alle meine Websites fehlerfrei und (subjektiv) in rasendem Tempo!
Mit anderen Worten: ***Viel Leistungsverbesserung für wenig Einsatz.***

2: Nginx erlauben, statische Resourcen direkt auszuliefern

Wählt man bei Plesk diese Betriebsart, so frägt Nginx bei den aufgeführten Dateitypen gar nicht mehr bei Apache nach, was damit zu tun ist – er liefert sie einfach selber aus. Das spart im Server das Hin- und Hergeschiebe. Allerdings: Die bisherige Apache-Konfiguration (vor allem das Umschreiben von Anfragen, das z.B. bei WordPress essentiell ist) greift bei den ausgewählten Dateitypen nicht mehr. Man muss ggf. dieses „verlorene Wissen“ in der

Konfiguration von Nginx nachbilden. Meist jedoch beschränkt sich das Umschreiben auf PHP-URLs, die weiterhin an Apache durchgereicht werden – einfach umstellen und prüfen! Sitemap-URLs dabei nicht vergessen!
Wer sich darum kümmert, kann die Performance erneut etwas steigern.

3: PHP via Nginx + PHP-FPM ausführen lassen

Diese Möglichkeit muss man mit der – sonst standardmäßig bei Plesk eingestellten – Verfahren vergleichen, **wobei Apache auch das FastCGI-Schnittstelle benutzt um PHP-Aufträge abzuarbeiten. Das heißt, die Methoden sind grundsätzlich gleich,** es kommt lediglich auf die Effizienz der Implementierung darauf an.

Hier heißt es, dass **PHP-FPM** eine fortgeschrittene Implementierung der FastCGI-Schnittstelle sei, die signifikante **Vorteile bietet bei Websites, die unter hoher Last stehen**. Hat man den Aufwand für Schritt (2) oben ohnehin schon geleistet, so kann man dann mit einem Mausklick auch diese Verbesserung „mitnehmen“.

Achtung: In dieser Konfiguration wird nichts mehr an Apache durchgereicht, d.h. **Plesk File-Sharing**, das sich auf das DAV-Modul in Apache stützt, **geht nicht mehr!**

Plesk: Nginx und PHP-FPM ggf. installieren

Laut Parallels sind Nginx und PHP-FPM standardmäßig in Plesk 11.5/12 dabei; in der HostEurope und Strato Varianten sind die Module nicht standardmäßig dabei, lassen sich aber leicht nachinstallieren:

Server | Tools and Settings | Panel | Updates and Upgrades | (neuer Reiter) Add/Remove Components | Plesk Hosting Features | Nginx web server and reverse proxy support

Nginx Version bei Ubuntu 12.04 + Plesk 11.5/12

Version: **1.6.0** (nginx -v) (auch bei Ubuntu 14.04 + Plesk 12)

Module verfügbar zusätzlich zu den Standard-Modulen (nginx -V)

- TLS SNI support enabled (SNI = Server Name Indication)
- --with-ipv6
- --with-file-aio
- --with-http_ssl_module
- --with-http_realip_module
- --with-http_sub_module
- --with-http_dav_module
- --with-http_gzip_static_module
- --with-http_stub_status_module

SPDY ist leider nicht dabei (--with-http_spdy_module).

HeadersMore, eine 3rdPartyModul, ist nicht dabei. Spannend: Mod PageSpeed von Google.

Nginx installiert, der Server rast!

Nach der Installation und Aktivierung von Nginx (und ggf. PHP-FPM) steht jeweils unter <Domain> | Web Server Settings | Abschnitt „nginx settings“ standardmäßig ein Häkchen bei „Smart static files processing“ (die übrigen zwei Checkboxen sind ohne Häkchen). Verblüffend wie sich diese „minimale“ Nutzung von Nginx sich auf die Antwortzeiten auswirkt!

Plesk 11.5/12: Wo finde ich bloß...

PHP5 – Dateien, Verzeichnisse, Config-Hierarchien

Vorweg: Nicht genau da wo sie in Standard-Ubuntu stehen! Zumal Plesk die möglichkeit bietet, für jede {Sub-} Domain eigene PHP-Einstellungen vorzunehmen (Websites & Domains | { Website } | PHP Settings). Für meine eigene Domain befindet sich z.B. die php.ini (für alle Apache2-Händler) unter

- /var/www/vhosts/system/timreeves.de/etc/php.ini
- /etc/php5 – hierunter die Standardversionen der php.ini Dateien
- include_path = .:/usr/share/php
- extension_dir: /usr/lib/php5/20090626
- Domain Error Logs: /var/www/vhosts/system/{(sub-)domain}/logs/error_log

Nginx – Dateien, Verzeichnisse, Config-Hierarchien

- Binary: /usr/sbin/nginx Server Error Log: /var/log/nginx/error.log
- Domain Access Log: /var/www/vhosts/system/{(sub-)domain}/logs/proxy_access_log
- Domain Access Log: /var/www/vhosts/system/{(sub-)domain}/logs/proxy_access_ssl_log
- Config Wurzel: /etc/nginx => /etc/nginx/nginx.conf
- /etc/nginx/conf.d/zz010_psa_nginx.conf =>

```
include /etc/nginx/plesk.conf.d/server.conf;              # Basics
include /etc/nginx/plesk.conf.d/webmail.conf;             # Webmail all domains
include /etc/nginx/plesk.conf.d/vhosts/*.conf;            # Links domains to (1)
include /etc/nginx/plesk.conf.d/forwarding/*.conf;        # Only domain forwards
include /etc/nginx/plesk.conf.d/wildcards/*.conf;         # Nur Plesk 12
(1) /var/www/vhosts/system/<domain>/conf/<timestamp>_nginx.conf
```

- In den meisten o.g. Dateien passiert nichts aufregendes; die Musik spielt in den domainspezifischen Config-Dateien (1). Da ist allerdings nicht sonderlich viel Musik...

PHP-FPM – Dateien, Verzeichnisse, Config-Hierarchien

- Binary: /usr/sbin/php5-fpm Konfig-Wurzel: /etc/php5/fpm
- Konfig des PHP-FPM Systems: /etc/php5/fpm/php-fpm.conf
- Pool Definitions: include=/etc/php5/fpm/pool.d/*.conf
- pid = /var/run/php5-fpm.pid error_log = /var/log/php5-fpm.log

- Achtung: Für /var/log/php5-fpm.log fehlt in Plesk 11.5 eine Log-Rotation!
- Restart per SSH: /etc/init.d/php5-fpm restart
- /etc/php5/fpm/pool.d – hier setzt Plesk ein <domain>.conf ab für jede Domain, für die PHP-FPM als Händler der PHP-Dateien eingestellt wird. Diese Dateien sind in sich komplett, sie includieren keine weiteren Dateien! Im Gegesatz zu Nginx, gibt es für PHP-FPM unter /var/www/vhosts/system/<domain> keine weiteren Config-Dateien, lediglich das Socket /var/www/vhosts/system/<domain>/php-fpm.sock

PHP-FPM konfigurieren

Bei wenig Domains bzw. wenig Last läuft die Plesk-Konfig für PHP-FPM „out of the box“ soweit fehlerfrei. Hier das FPM-Manual auf php.net.

Erhält eine Domain bzw. Subdomain, die PHP-FPM bedient, mehr „gleichzeitige“ Anfragen (z.B. durch CalDAV und CardDAV Anfragen aus Thunderbird mit Lightning + SOGo Connector, womöglich noch der ownCloud Client dazu), so kann es zu folgendem Fehler kommen:

WARNING: [pool timreeves.de] server reached max_children setting (5),
consider raising it

Ein paar Stunden nach dem ersten Auftauchen dieser Meldung hatte sich bei meinem Server FPM aufgehängt, alle Websites waren nicht mehr erreichbar, Nginx sagte bei allen nur „502 Bad Gateway“. Nach einem Restart von FPM lief alles wieder. Also muss max_children erhöht werden – aber wo greift man ein in der Plesk-Konfiguration? Die Pool-Konfigs geben Aufschluss:

```
; If you need to customize this file, use either custom PHP settings tab in
; Panel or override settings in /var/www/vhosts/system/<domain>/conf/php.ini.
; To override pool configuration options, specify them in [php-fpm-pool-settings]
; section of /var/www/vhosts/system/<domain>/conf/php.ini file.
```

Die dort genannte Datei ist nicht mit system/<domain>/**etc**/php.ini, welche Plesk für Apache erzeugt, zu verwechseln. Vielmehr muss man sie selber händisch anlegen, z.B. mit Inhalt:

```
[php-fpm-pool-settings]
pm.max_children = 20
```

und anschließend in Plesk Panel die Domain PHP Settings erneut erzeugen lassen – dazu muss man auch tatsächlich in der Plesk-Maske etwas ändern, sonst macht er nichts! Ein Neustart von PHP-FPM auf SSH-Ebene reicht nicht.

Wenn Sie jedoch Alternative: Die Kontrolle an sich reißen, dann greifen auch etwaige /var/www/vhosts/system/<domain>/conf/php.ini Dateien nicht mehr, weil die von Plesk erzeugten Komposit-Dateien in /etc/php5/fpm/pool.d nicht mehr im Einsatz sind. In dem Fall editieren Sie einfach die Config in /etc/php5/fpm/pool.d/my_<domain>.conf zurecht.

Hier ist ein guter Artikel zu Nginx + PHP-FPM.

Weitere wichtige Infos zu PHP-FPM Config stehen weiter unten bei Nginx, da die beiden sich gegenseitig bedingen: Alternative: Die Kontrolle an sich reißen.

Prozess-Manager – static, ondemand oder dynamic?

Elementar wichtig für die Performance des Gesamtsystems ist die Wahl des Modells, wonach der Prozess-Manager Kindprozesse zur Beantwortung von Anfragen bereit hält. Zur Wahl stehen:

- static - die Anzahl der Kindprozesse ist fest eingestellt (pm.max_children).
- ondemand - die Kindprozesse werden gestartet, sobald sie benötigt werden, im Gegensatz zu dynamic, wo zu Beginn bereits pm.start_servers Prozesse gestartet werden.
- dynamic - die Anzahl der Kindprozesse wird dynamisch eingestellt, wobei die folgenden Einstellungen zugrundegelegt werden: pm.max_children, pm.start_servers, pm.min_spare_servers, pm.max_spare_servers.

Die Standard-Einstellung durch Plesk für eine Domain sind:

```
pm = ondemand
pm.max_children = 5
pm.process_idle_timeout = 10s
; Following pm.* options are used only when 'pm = dynamic'
pm.start_servers = 1
pm.min_spare_servers = 1
pm.max_spare_servers = 1
…
;pm.process_idle_timeout = 10s;  Default Value: 10s
```

Das ist natürlich im vorsichtigen Plesk-Manier: Server werden überhaupt nur dann gestartet, wenn sie benötigt werden, d.h. die (erste) Antwort verzögert sich um die Zeit, einen Server dafür zu starten. Bei Domains mit wenig Anfragen stirbt der Server nach 10 Sekunden (idle timeout), die nächste Anfrage startet wieder einen Server für sich. Bei Domains, deren Antwortzeit nicht kritisch ist (wenn sie eh nur so wenig angesteuert werden), ist das OK, weil es Hautpspeicher schont. Bei Websites mit hohen Besuchsraten bietet sich „dynamic“ an, so dass wenigstens ein Server Gewehr bei Fuss steht.

https://ma.ttias.be/a-better-way-to-run-php-fpm/

Für Domains mit erheblich mehr Anfragen muss unbedingt individuell auf die Hardware sowie Anfrage-Aufkommen eingegangen werden, siehe hier, hier und hier.

`pm.max_requests` int
Die Anzahl an Anfragen, die ein Kindprozesse ausführt, bevor er neu startet. Das kann hilfreich sein, um Memory Leaks in Bibliotheken von Drittanbietern zu debuggen. Für eine unbegrenzte Anfrageanzahl '0' verwenden. Vergleichbar mit *PHP_FCGI_MAX_REQUESTS*. Standardwert: 0.

Vorschlag für Domains mit sehr wenig Traffic:

```
# Nur wenn's sein MUSS, sonst nutze Apache FastCGI, der Apache-Vhost steht eh bereit
pm = ondemand
pm.max_children = 5
pm.process_idle_timeout = 10s
pm.max_requests = 200
```

Vorschlag für Domains mit etwas mehr Traffic:

```
pm = dynamic
pm.max_children = 16
pm.process_idle_timeout = 60s
pm.start_servers = 2
pm.min_spare_servers = 2
pm.max_spare_servers = 2
pm.max_requests = 500
```

Vorschlag für Domains mit noch mehr Traffic (z.B. ownCloud):

```
pm = dynamic
pm.max_children = 32
pm.process_idle_timeout = 120s
pm.start_servers = 4
pm.min_spare_servers = 4
pm.max_spare_servers = 6
pm.max_requests = 1000
```

Standard-Config: 4 Server für www-data – Upps!

In /etc/php5/fpm/pool.d/www.conf steht die Basis-Config für FPM. Dieser startet 4 Server für User www-data im „dynamic“ Modus, d.h. sie sind wirklich da. Soweit ich sehen kann werden sie allesamt nie benutzt, weil für jede Domain ein eigener Server definiert wird, also bleiben für diese 4 Server keine Anfragen übrig. Die Config scheint aus der Zeit zu stammen, eher FastCGI benutzt wurde bei Plesk und alle Websites und der Server liefen mit User www-data. Oder Plesk hat diese Datei (noch) gar nicht angepasst...

Ich habe daher die Datei editiert und pm = ondemand gesetzt.

Plesk PHP Settings bei Nginx + PHP-FPM

Egal ob Apache oder Nginx + PHP-FPM als Webserver benutzt wird, Plesk erzeugt immer aus den pro Domain angegebenen „PHP Settings“ folgende Ini-Datei:

- /var/www/vhosts/system/<domain>/etc/php.ini (aus /etc/php5/cgi/php.ini)

Die angepasste php.ini wird in den Apache-Configs eingebunden wie folgt:

- /etc/apache2/plesk.conf.d/vhosts/<domain> =>
  - /var/www/vhosts/system/<domain>/conf/<timestamp>_httpd.conf
    - SetEnv PP_CUSTOM_PHP_INI /var/www/vhosts/system/<domain>/etc/php.ini

Bei Verwendung von Nginx + PHP-FPM verwendet Plesk eine ähnliche Struktur:

- /etc/nginx/plesk.conf.d/vhosts/<domain> =>
  - /var/www/vhosts/system/<domain>/conf/<timestamp>_nginx.conf
    - (mitunter) => include /etc/nginx/fastcgi.conf
    - (zuletzt) => include "/var/www/vhosts/system/{domain}/conf/vhost_nginx.conf";

Die letzte Anweisung includiert die individuellen Domain-Settings für Nginx, wie sie bei Plesk unter *{Domain}* | *Web Server Settings* | *nginx settings* | *Additional nginx directives* eingegeben werden (können).

Die PHP-Einstellungen sind also generell nicht bei den Nginx-Direktiven zu suchen, sondern bei PHP-FPM:

- Der Master-Prozess wird gestartet mit config: /etc/php5/fpm/php-fpm.conf
- Am Ende jener Datei steht: include=**/etc/php5/fpm/pool.d/*.conf**
- Dort deponiert Plesk eine .conf Datei für jede Domain, für die PHP-FPM konfiguriert wurde; am Ende dieser Datei wiederum steht ein Abschnitt „php.ini custom configuration directives“, dort stehen (nur) die nicht-standard Werten, die man in den PHP Settings der Domain in Plesk setzt.
- Wer z.B. generell ein höheres memory_limit in PHP haben will, *könnte* die Standard-Datei */etc/php5/fpm/php.ini direkt* editieren; oder es pro Domain in den PHP Settings eingeben. Letzteres hat den Vorteil dass es von Änderungen des Systems bzw. Plesk-Dateien unabhängig ist, und greift unabhängig davon ob man Apache oder Nginx + PHP-FPM als Webserver benutzt.
- Folgende Einstellungen in den Domain-Settings für PHP heben Unterschiede in Plesk bei Apache / FPM auf und bekommen die meisten Anwendungen flott (und sicherer):

```
# Im Abschnitt: Peformance settings
memory_limit = 384M
max_execution_time = 60
upload_max_filesize = 128M
# Im Abschnitt: Common settings
include_path = … # Falls nötig
open_basedir = … # Wird oft benötigt; Plesk default: {WEBSPACEROOT}{/}{:}{TMP}{/}
                 # Plesk-Default erlaubt Übergriffe im WorkSpace, besser {DOCROOT}
# Im Abschnitt: Additional directives
expose_php = off
date.timezone = Europe/Berlin
; mbstring.func_overload = 7   ; Default 0, ownCloud: 0, eGroupware 1.8: 7, 14.1: 0
```

Ansonsten...

- Die zu-ladenden PHP-Module werden durch ein Symlink */etc/php5/fpm/conf.d* nach */etc/php5/conf.d* auf die gleichen Module gelenkt, die auch für Apache gelten.
- PHP's „Loaded Configuration File“ ist nicht wie bei Apache die von Plesk gemäß den „Settings“ erzeugte Datei, sondern die Standard-Datei */etc/php5/fpm/php.ini*
- *Dies scheint eine recht standardmäßige, also von Plesk nicht veränderte Datei zu sein (z.B. mit memory_limit = 128M).*
- Achtung: Bei FastCGI – egal ob per Apache oder PHP-FPM – wird ein Prozess pro Workspace gestartet, damit es mit den jeweiligen Benutzer Uid + Gid laufen kann. Dementsprechend kann man (muss man) eine eigene Konfiguration dafür nutzen.
- include=/etc/php5/fpm/*.conf => Man könnte z.B. ein „z.conf“ dort deponieren, doch das würde die Konfig von PHP-FPM betreffen, nicht die vom PHP-Interpreter.
- Letztere ist in den INI-Dateien in /etc/php5{/fpm}/conf.d, das durch SymLinking für alle PHP-Inkarnationen (apache2, cgi, cli, fpm) wiederverwendet wird. Diese Dateien werden

automatisch von PHP geladen, doch man kann das Symlink auflösen und eine „spezielle“ Konfig für ein SAPI einbringen, siehe hier; kurz geschildert:

- cd /etc/php5/fpm ; unlink conf.d ; mkdir conf.d ; cp /etc/php5/conf.d/*.ini conf.d
- Und jetzt bel. .ini in /etc/php5/fpm/conf.d hinzufügen, /etc/init.d/php5-fpm restart
- Übrigens: Plesk ist es egal ob PHP-FPM läuft, er nutzt nichts davon.

Siehe auch here.

PHP-FPM Umgebung einsehen

Man kann dies mit einem eigenen Skript machen:

<pre><?php var_export($_SERVER)?></pre>

Achtung: Siehe hier: In both the CGI and FastCGI SAPIs, $_SERVER is also populated by values from the environment; S is always equivalent to ES regardless of the placement of E elsewhere in this directive.

Und wenn Sie dann in Plesk PHP Settings den „Additional configuration directives“ ein „variables_order = "EGPCS"“ hinzufügen werden Sie sehen können, dass $_ENV das gleiche enthält wie $_SERVER; phpinfo() indes zeigt, welcher Wert woher kam (aber nur ohne das „variables_order“, sonst sind überall beide arrays gleich).

PHP-FPM Status per URL einsehen

Zunächst muss die standardmäßig in FPM ausgeschaltete Möglichkeit in der <domain>/**conf**/php.ini Datei aktiviert werden, z.B.

```
[php-fpm-pool-settings]
pm = dynamic
pm.max_children = 20
pm.status_path = /status
ping.path = /ping
```

Die für „dynamic“ nötigen Anweisungen generiert Plesk standardmäßig wie folgt:

```
pm.start_servers = 1
pm.min_spare_servers = 1
pm.max_spare_servers = 1
```

Die status und ping URLs müssen aber Nginx passieren und PHP-FPM erreichen, hierzu in der Nginx-Konfiguration ergänzen:

```
location = /status { fastcgi_pass "unix:/var/www/vhosts/system/<domain>/php-fpm.sock";
include /etc/nginx/fastcgi.conf; }
location = /ping { fastcgi_pass "unix:/var/www/vhosts/system/<domain>/php-fpm.sock";
include /etc/nginx/fastcgi.conf; }
```

PHP-FPM Error Log Rotation

Plesk 11.5 hat hierfür keine Vorkehrung getroffen – die Datei /var/log/php5-fpm.log wächst unendlich. Erzeugen Sie einen Script wie folgt:

```
#! /bin/bash
```

```
log=/var/log/php5-fpm.log
cp ${log} ${log}.1
> ${log}
chmod 644 ${log}
# echo "Rotated ${log}"
exit 0
```

und tragen Sie ihn als Cron-Job ein; hier führen mehrere Weg nach Rom, ich hänge einfach folgende Zeile an /etc/crontab an, der Cron-Daemon merkt selbst die Änderung:

```
1 0 * * 0 root [ -x {Ihr-Script-Pfad} ] && {Ihr-Script-Pfad} >/dev/null 2>&1
```

Nginx konfigurieren

Einiges was bei Apache2 (á la Plesk) standardmäßig dabei ist, oder in Apache .htaccess Dateien zurecht gebogen wird, muss man in Nginx selber konfigurieren. Ich empfehle hierzu, soweit möglich, eine eigene „nginx.conf“ im Wurzelverzeichnis der jeweiligen Website / Anwendung abzulegen, und diese in Plesk zu includieren:

Plesk: <Domain> | Web Server Settings | Additional nginx directives

include "/var/www/vhosts/<domain-path>/nginx.conf";

denn eine „nginx.conf“ im DocRoot wird nicht automatisch ausgeführt.

Danach kann man die Datei bequem lokal bearbeiten und per FTP hochladen; ein Neustart vom Nginx-Service ist bei jeder Änderung jedoch nötig:

- Plesk: Server | Services Management | Reverse Proxy Server (nginx)
- ODER SSH: /etc/init.d/nginx restart

Nginx Wiki Übersicht aller Configuartion-Artikel **hier**.

Sehr guter Artikel über PHP-FPM und Nginx auf Ubuntu hier; und hier **Nginx Pitfalls**.

Eine Liste von Webseiten für Nginx-Config für diverse Anwendungen.

Wer zwecks Debuggen den kompletten Datenverkehr loggen will, siehe hier, oder man lässt das in Nginx und verwendet stattdessen tshark.

Wie Plesk eine Domain für Nginx konfiguriert

Vorweg: Eher karg, es ist nicht einmal GZIP-Komprimierung eingeschaltet.
Was es gibt ist:

- 4 Abschnitte, je einmal für IPv4 und IPv6, mit und ohne SSL.
- Aber nur 2 Abschnitte falls ohne IPv6 für das WorkSpace. **Je Abschnitt:**
- server_name, ggf. ssl_*, root, access_log, diverse „locations“:
    - Header-Setting, Interna, Dateinamen zum Probieren bei Verzeichnis-URLs
    - Falls in Plesk „Serve static files directly by nginx“ gewählt wurde:
      location ~ ^/(.*\.(<Datei-Endung-Liste>))$ { try_files $uri @fallback; }
      wobei @fallback die Weiterleitung an Apache via Port 7081 ist.
      Achtung @fallback existiert NUR wenn dieses Checkbox gewählt ist!

  - Falls in Plesk „Process PHP by nginx“ gewählt wurde:
    - Generelle Behandlung von .php Dateien: location ~ \.php(/.*)?$
      Matches .php mit optional ein String angehängt, der mit „/“ anfängt.
      Die nachstehende { … } Direktive regelt die Behandlung per FastCGI.
    - PHP-Skripte von Plesk „Web-Users“ werden auch behandelt
- Zum Schluss wird die Datei aus Plesk „Additional nginx directives“ includiert.

Das „Not found“ Problem

Nginx kennt nur EIN Benutzer-Id für alle Worker-Prozesse, siehe hier. Das bedeutet, dass Dateien, die besagtes User-Id nicht lesen kann, z.B. weil sie kein "world read" haben, von Nginx als "Not found" gemeldet werden, auch dann wenn sie da sind - es fehlt an Zugriffsrechten.

Die Tatsache, dass PHP-FPM, so es von Nginx angesteuert wird, dann mit den richtigen User und Group Ids läuft und alle Dateien der Website sehen (und schreiben) kann, ist zwar wichtig, hilft aber nicht weiter was Nginx betrifft.

Lösungsansätze:

- Wenn Sie nur ein Plesk WorkSpace benutzen, könnten Sie User und Group dieses Workspaces in die Nginx-Config ein. Sonst:
- Plesk setzt selbst Verzeichnisse mit Gruppe psaserv ab, Verzeichnisse und Dateien von FTP oder PHP haben Gruppe psacln. Es ist also nicht möglich, das Problem über die eine Gruppe der Nginx-Config zu lösen.
- Daher braucht der Nginx-User zugehörigkeit zu beiden Plesk-Gruppen.
  Von Plesk aus ist nginx bereits in der Gruppe psaserv (s. /etc/group):
  psaserv:x:1004:www-data,psaftp,psaadm,nginx
- Geben Sie daher als Super-User ein: **usermod -aG psacln nginx**

Der Haken am Häkchen: Process PHP by nginx

Falls bei Plesk | Domain | "Web Server Setting" das Checkbox "Process PHP by nginx" AUS (ohne Häkchen) ist, dann:

1. Alle ".php" Skript-Aufrufe werden von Nginx an Apache weitergereicht.
2. In /etc/php5/fpm/pool.d gibt es keine Konfigurationsdatei für die Domain, d.h. man kann in der eigenen Nginx-Config nichts an PHP-FPM zur Bearbeitung senden!
3. Dafür gibt es eine Apache2- Konfigurationsdatei für die Domain, ein Symlink unter /etc/apache2/plesk.conf.d/vhosts > /var/www/vhosts/system/<domain>/conf/httpd.conf

Falls bei Plesk | Domain | "Web Server Setting" das Checkbox "Process PHP by nginx" EIN (mit Häkchen) ist, dann setzt Plesk eine "location ~ \.php(/.*)?$ {" Directive ein, welche die ".php" Skripte an PHP-FPM weiterreicht. Mit den mit dieser Entscheidung verbundenen Plesk-Direktiven gibt es insgesamt Probleme:

1. Im Block der Direktive für ".php" Dateien gibt es keine "try_files" Anweisung. Daher leitet Nginx auch Anfragen für nicht-existenten ".php" Dateien an FPM weiter, der sie nicht findet und eine entsprechende Meldung an Nginx zurückliefert. Die Meldungen – meist durch Angriffe verursacht – landen im Nginx Log und müllen es zu. Das ist ineffizient; und „echte“ FPM-Fehlermeldungen gehen im aufgeblähten Log unter!
2. Da besagter Direktiven-Block außerhalb der eigenen Kontrolle liegt, kann man zudem z.B. nicht direkt eine eigene „fastcgi_param“ ergänzen. Es geht, aber nur indirekt – dafür gibt’s Die letzte Rettung: Custom Configuration Template.
3. Das Häkchen erzeugt zudem folgende Direktive: "location ~ /$ { index index.html ... }" Die Absicht ist Verzeichnis-Indexierung, doch das haut manchmal daneben – z.B. bei WordPress-Permalinks: Es ist ein Regex und steht weit vorne in der Gesamtkonfiguration, d.h. es greift bei jeder URI die mit „/“ endet. Permalinks aber sind Konstrukte jenseits von Plattenverzeichnissen. Bräuchte man eine andersartige Behandlung hierfür, so ist dies nicht mehr mit „location“ erreichbar – man muss auf ein „rewrite“ ausweichen.
4. Ferner kommen zwei locations für URIs beginnend „~“ - die schaden zwar nicht, sind aber meist überflüssing – das sind für „Web Users“ (erhöhen mögliche Angriffsflächen).
5. All das ist sehr ärgerlich, ist es nicht? Es ist!

Alternative: Die Kontrolle an sich reißen

Aus o.g. Gründen empfehle ich, bei Websites mit hohem Traffic dennoch PHP-FPM einzusetzen und dabei selber die Kontrolle zu übernehmen, wie im Folgenden beschrieben. Sonst verzichten Sie besser auf PHP-FPM und nutzen Sie einfach Apache FastCGI: Denn im folgenden Verfahren wird Plesk „gelinkt“ - wir sagen ihm schlussendlich dass es keinen PHP-Support auf der Domain gibt, realisieren sie dann an ihm vorbei.

- Setzen Sie Ihre „PHP Settings“ in Plesk so, wie Sie für die Domain nötig sein werden.
- Setzen Sie bei „Process PHP by nginx“ das Häkchen und erzeugen Sie die „Web Server Settings“ neu.
- Dann im SSH:
- `cd /etc/php5/fpm/pool.d`
- `cp <domain>.conf my_<domain>.conf`
- Nun entfernen Sie bei „Process PHP by nginx“ das Häkchen wieder und erzeugen Sie die „Web Server Settings“ neu.
- Darauf hin entfernt Plesk wieder die – von ihm erzeugten – Datei <domain>.conf aus dem PHP-FPM Config Pool und startet PHP-FPM neu.
- So haben wir ein Klon von Plesk's Datei erzeugt, der aufgrund des anderen Namens nicht von Plesk angerührt wird – aber von PHP-FPM beim Erarbeiten der Konfiguration ganz normal gefunden und benutzt wird.
- Nicht vergessen: Jetzt... Plesk | Domain | Hosting Settings | **PHP support AUS** (Häkchen weg) – sonst konfiguriert Plesk eine Apache-FastCGI Instanz von PHP für die Domain! Die zwar die Funktionsfähigkeit der FPM-Lösung nicht beeinträchtigt, aber völlig überflüssig ist da sie nie angesprochen wird, weil Nginx nun alle PHP-Anfragen an FPM leitet:

- JETZT fehlen alle störenden Direktive (die oben erwähnten) in der Plesk-Config für Nginx (für diese Domain). Seufzer der Erleichterung! Allerdings, jetzt **müssen** wir in den eigenen Direktiven für Nginx für die Weiterleitung von ".php" an PHP-FPM sorgen – sonst käme es zu einem „Not Found“. So in etwa wie hier:

```
# PHP-Handling: Best of Plesk and ownCloud and Web-Forum articles
# A match here pre-empts Plesk's "location /", sends the script to PHP-FPM
# "(?U)" sets internal "ungreedy" option, i.e. to FIRST ".php" - COOL
location ~ ^((?U).+?\.php)(/.*)?$ {
  # TR: See http://wiki.nginx.org/HttpFastcgiModule#fastcgi_split_path_info
  # TR: Directive populates $fastcgi_script_name + $fastcgi_path_info
  # TR: BUT there's a problem: http://trac.nginx.org/nginx/ticket/321
  fastcgi_split_path_info ^((?U).+\.php)(/?.+)$;
  set $path_info $fastcgi_path_info;
  # Requested PHP Scripts MUST exist exactly as named
  try_files $fastcgi_script_name =404;
  # try_files has emptied $fastcgi_path_info
  fastcgi_param PATH_INFO $path_info;
  # You can add any application-special headers to $_SERVER here
  fastcgi_param XAUTHORIZATION "$http_authorization" if_not_empty;
  fastcgi_param HTTP_XAUTHORIZATION "$http_authorization" if_not_empty;
  fastcgi_pass "unix:/var/www/vhosts/system/<domain>/php-fpm.sock";
  # TR: Sets SCRIPT_FILENAME as $document_root$fastcgi_script_name
  # TR: Meaning that $fastcgi_script_name MUST be correct now!
  include /etc/nginx/fastcgi.conf;
  # access_log off;
}
# Plesk's Directives for Web User Access are not needed on this domain!
```

Wenn Sie nachträglich PHP-Einstellungen für die Domain anpassen wollen, so greifen die Einstellungen in Plesk nun ins Leere – sie werden für Apache erzeugt, nicht für PHP-FPM. Also müssen Sie ggf. händisch die Datei editieren:

- `cd /etc/php5/fpm/pool.d`
- `vi my_<domain>.conf` (editieren, speichern)
- `service php5-fpm restart`

Nginx „location“ Directive

Eigene „location“ Direktive greifen manchmal nicht, weil Plesk bereits welche abgesetzt hat, die Vorrang nehmen. Hier zunächst eine knappe Erläuterung der Direktive:

- location {modifier | empty} { prefix string | regex }
- prefix string: Matches the start of the URI; regex: Is a regex :)

To find the location matching a given request...

- prefix string locations are tested first, longest match wins;
- Nginx notes the winning prefix location;
- If modifier is one of **=** or **^~** then **no** regex-testing is performed **if** any prefix string match was found;

- Otherwise the search of regular expressions begins, **in the order of their appearance** in the config;
- Regex processing terminates on the first match, and the corresponding location configuration is used. **If** no match with a regular expression is found, **then** the configuration of the **prefix** location remembered earlier is used.
- **Modifiers:**

	No modifier implies string is a fixed prefix string, not a regex
=	Exact match of fixed prefix string and location (full uri)
^~	(NOT regex) If this prefix string matches, then don't do any regex testing
~	regex case-sensitive
~*	regex case-insensitive

Benutze 'rewrite' wenn 'location' nicht greifen würde

What that all effectively means is, that Plesk's regex location directive for ".php", appearing before any user directives, will ALWAYS win - IF the user directives are of the regex type. In order for user locations to win, they would have to be of type prefix string and have either the "=" or the "^~" modifier. The problem here is, that a lot of the rules we need to use, e.g. for security settings, cannot be formulated in that way, they need to be of the regex type to get the appropriate matches. Meaning that Plesks "location" will always pre-empt them.

That's why I looked for an alternative and found the rewrite directive: "The *rewrite* directives are executed **sequentially in order of their appearance** in the configuration file". For a better explanation see here (scroll down to section "Rewriting the URI"). So assuming no other rewrite has matched our URI before our directives are reached (in Plesk at least none has), then the URI gets rewritten as we specify; and then **location searching starts over**, using the new URI. Thus using our freedom to rewrite the URI to something like "/extrawurst", for which Plesk (and probably no other pre-configured system) has any location rules, well, we've just dug ourselves out of the hole!

Weitere Tipps bzgl. Nginx-Konfiguration gibt es im Web z.B. hier, hier und hier.

Hier welche Abkürzungen beim expires Direktive erlaubt sind.

Hier die Doku zu try_files; hier die Liste der Embedded Variables.

Die letzte Rettung: Custom Configuration Template

Plesk erzeugt Pro-Domain (und Subdomain) eine Nginx-Konfiguration mittels einer PHP Template-Datei. Hierin sind manche Direktive hartcodiert, die man nachher in der (zuletzt hinzugefügten) eigenen Direktiven nicht mehr übertrumpfen kann, z.B. client_max_body_size (128m), oder eben „location /“.

In diesem Fall kann man ein eigenes „Custom Template“ anlegen (d.h. kopieren und anpassen) und damit die Konfiguration neu erzeugen lassen. Hier in einem Blog erklärt, hier bei Parallels.* Caveat: Das neue Template ist nicht spezifisch für eine bestimmte Domain – es werden fortan ALLE Domains mit der neuen Konfiguration versehen! Von daher möglichst wenig auf diesem Weg machen!

* *Der Plesk-Link funzt nicht richtig: Die Sektion „Changing Virtual Hosts Settings Using Configuration Templates“ (weiter oben auf der Webseite) manuell aufmachen.*

```
cd /usr/local/psa/admin/conf/templates
mkdir custom custom/domain custom/domain/service
cp default/domain/nginxDomainVirtualHost.php custom/domain
cp default/domain/service/fpm.php custom/domain/service
cd custom/domain
# und nun nginxDomainVirtualHost.php bearbeiten...
cd service
# und nun fpm.php bearbeiten... **
/usr/local/psa/admin/bin/httpdmng --reconfigure-domain <{sub-}domain>
```

** Ist PHP_SELF leer? (a) Sie haben bei den „PHP Settings“ für die Domain „cgi.fix_pathinfo = 0“ gesetzt, was das Befüllen von PHP_SELF unterdrückt; oder (b) Sie haben in der Nginx-Config „try_files“ benutzt ohne Workaround – das leert ja $fastcgi_path_info!
Nutzen Sie statt PHP_SELF besser $_SERVER['SCRIPT_NAME'] + $_SERVER['PATH_INFO'].

Nginx Settings für diverse Anwendungen

Allgemeine Anmerkungen

Siehe auch oben Nginx konfigurieren. Hier ein Converter für .htaccess nach Nginx.

Standardmäßig ist bei Plesk/Nginx keine GZIP-Komprimierung der ausgelieferten Dateien eingestellt, siehe mein Artikel „Server-Tipps: Ubuntu 12.04 LTS + Plesk 11.5“. Dort stehen die Anweisungen zum Kopieren.

Testen-Tipp: Hier liest man: HTTP 1.1 verbietet Content-Length Header wenn ein Transfer-Encoding Header gesetzt ist: *3. If a Content-Length header field (section 14.13) is present, its decimal value in OCTETs represents both the entity-length and the transfer-length. The Content-Length header field MUST NOT be sent if these two lengths are different (i.e. if a Transfer-Encoding header field is present). If a message is received with both a Transfer-Encoding header field and a Content-Length header field, the latter MUST be ignored.*

Generisch für PHP-Websites

Eine solide Ausgangsbasis, bedarf ggf. Anpassung pro Website:

Serve static files directly by nginx: KEIN Häkchen, das regeln wir differenzierter!

Additional nginx directives (direkt in Plesk eingeben oder dort als Datei includieren):

```
# The "generic" handling of static resources
# Note that they are all formulated securely with "try_files".

charset utf-8;
location = /nginx.conf { deny all; }

# PUT MOST FREQUENT FIRST
# try_files "uses the first found file for request processing; the processing is performed
in the current context"

# If your website uses static html files rather than php, or a mix:
# Docu: "using an index file causes an internal redirect, and the
```

```
# request can be processed in a different location"
# location ~* /$ { index index.html; }

# If your website uses "pretty URLs", i.e. hiding the "index.php":
# location ~* /$ { index index.php; }

# Frequent possibly modified content valid 2 weeks in Cache
location ~* \.(js|css|htm|html|xhtml|xml|json)$ {
 try_files $uri =404;
 expires 2w;
 add_header Pragma public;
 add_header Cache-Control public;
 access_log off;
 # add_header Debug Frequent;
}

# Pics and Fonts valid 90 Days in Cache
location ~* \.(png|jpg|jpeg|gif|ico|bmp|img|ttf|otf|eot|svg|svgz|woff)$ {
 try_files $uri =404;
 expires 90d;
 add_header Pragma public;
 add_header Cache-Control public;
 access_log off;
 # add_header Debug PicFont;
}

# Zips + PDF valid 2 weeks in Cache
location ~* \.(bz2|exe|gz|pdf|rar|tgz|zip)$ {
 try_files $uri =404;
 expires 2w;
 add_header Pragma public;
 add_header Cache-Control public;
 access_log off;
 # add_header Debug ZipPDF;
}

# Media files (large) valid 1 week in Cache
location ~* \.(ac3|avi|flv|iso|mp3|mp4|mpeg|mpg|ogg|qt|rm|swf|wav)$ {
 try_files $uri =404;
 expires 1w;
 add_header Pragma public;
 add_header Cache-Control public;
 access_log off;
 # add_header Debug Media;
}

# Infrequent possibly modified content valid 3 days in Cache
location ~* \.(dat|doc|docx|dts|ppt|pptx|tar|txt|xls|xlsx)$ {
 try_files $uri =404;
 expires 3d;
 add_header Pragma public;
 add_header Cache-Control public;
 access_log off;
 # add_header Debug Infrequent;
```

```
}

# Copied from Plesk "Handle PHP by Nginx" config - access to statistics via Apache
location ~ ^/(plesk-stat|webstat|webstat-ssl|ftpstat|anon_ftpstat|awstats-icon) {
    proxy_pass https://##YOUR-IP##:7081;
    proxy_set_header Host             $host;
    proxy_set_header X-Real-IP        $remote_addr;
    proxy_set_header X-Forwarded-For  $proxy_add_x_forwarded_for;
    proxy_set_header X-Accel-Internal /internal-nginx-static-location;
    access_log off;
}
# NOTE: If you select "Process PHP by nginx" in Plesk, then Plesk adds a "location ~ /$
{ index index.html ... }"
# This Regex, providing indexing for anything ending in "/", is NOT ALWAYS HELPFUL,
scuppers WordPress Permalinks
# It pre-empts any later regex for "/$", causing an index to be searched for in the non-
existent permalink "directory"
# SO if your app uses URLs matching "^/.+/$" and needing special treatment, you'll need to
"rewrite" them!

# Virtually nothing should arrive here: We've already (hopefully) done any
# special protected / allowed locations, all ".php" scripts, all static assets.

location ~ ^/.+ {
 # Allow the unrecognised type to be a file or directory, reject all else
 try_files $uri $uri/ =404;
 # Let this be logged – we want to know of anything untoward!
}

# Finally, switch gzipping on
gzip on;
gzip_proxied any;
gzip_min_length 100;
gzip_buffers 8 16k; # number size, default 32 4k|16 8k
gzip_types text/css text/plain text/javascript application/javascript application/json
application/x-javascript application/xml application/xml+rss application/xhtml+xml
application/x-font-ttf application/x-font-opentype application/vnd.ms-fontobject
image/svg+xml image/x-icon application/rss+xml application/atom_xml;
gzip_vary on;
gzip_comp_level 9;
gzip_http_version 1.0;
gzip_disable "MSIE [1-6]\.(?!.*SV1)";
```

WordPress

Achtung Fixes nötig für Page-Cache-Plugins! Mehr Tipps hier und hier.

Variante bei Nutzung von Memcached: Hier.

Wer sich tief in die Materie einarbeiten will, und Nginx zum Caching nutzen: Hier.

?q=$uri&$args after /index.php ist weit verbreitet aber kontraproduktiv, es hindert Cache-Plugins, die ein GET-Parameter sehen und die Seite daher nicht cachen. WordPress ist

schlau und kann Permalinks aus der SERVER-Umgebung (QUERY_STRING, REQUEST_URI) selber ergänzen. Siehe Forum-Post.

In /etc/nginx/fastcgi.conf stehen mitunter die Anweisungen
o fastcgi_param QUERY_STRING $query_string;
o fastcgi_param REQUEST_URI $request_uri;
WordPress nutzt diese Angaben aus der Server-Umgebung wenn „index.php“ ohne GET-Parameter aufgerufen wird – somit kann man GET-Parameter weglassen, was Seitencache-Plugins wie Quick Cache zum Funktionieren bringt.

Tipp: Wird das angeforderte URL nicht gefunden, liefert WordPress die entsprechende Seite des Themes zurück – und zwar mit einem HTTP Status 404 Not Found – was so gesehen korrekt ist, also debuggen Sie nicht ein „Not Found“, der kein Fehler ist!

Serve static files directly by nginx: KEIN Häkchen – das regeln wir differenzierter!

Additional nginx directives (direkt in Plesk oder in einer includierten Datei eingeben):

```
# THIS Example file (WordPress) assumes Case A2!!!
# You MUST insert your domain path at line 40
# You MUST insert your domain name at line 87

# A) IF the Plesk Domain "Web Server Setting" "Process PHP by nginx" is OFF then...
# 1) all ".php" script calls will be passed to Apache; use here as is (edit line 27)
# 2) BUT if you have manually activated PHP-FPM by cloning the domains config
#    in /etc/php5/fpm/pool.d, then you can leave it OFF but yourself pass all the
#    script requests to php-fpm - a block below does that.

# B) IF the Plesk Domain "Web Server Setting" "Process PHP by nginx" is ON then...
# Plesk sets a "location ~ \.php(/.*)?$ {" which passes the scripts to PHP-FPM.
# The problem is, that Plesk's directive block does NOT include a "try_files",
# which leaves the setup insecure and passes attack calls to FPM.

# After each modification: /etc/init.d/nginx restart
# Server Error log: /var/log/nginx/error.log
# Domain Error log: /var/www/vhosts/system/<domain>/logs/error_log
# http://wiki.nginx.org/Pitfalls
# http://www.php.net/manual/de/regexp.introduction.php

# If you select "Process PHP by nginx" in Plesk, then Plesk adds a "location ~ /$ { index
index.html ... }"
# This Regex, providing indexing for anything ending in "/", is NOT HELPFUL, scuppers
WordPress Permalinks
# It pre-empts any later regex matching "/$", causing an index to be searched for in the
non-existent
# permalink "directory". So we must use a try_files or rewrite rather than a location!

# Begin for WordPress SEO Sitemap
rewrite ^/sitemap_index\.xml$ /index.php?sitemap=1 last;
rewrite ^/([^/]+?)-sitemap([0-9]+)?\.xml$ /index.php?sitemap=$1&sitemap_n=$2 last;
rewrite ^/([^/]+?)-sitemap.xsl$ /index.php?xsl=$1 last;
# End for WordPress SEO Sitemap
```

```
# The catchall (at end of this file) has a try_files...
# Careful: /wp-admin/{several}.php must be accessible
rewrite ^/wp-admin/$ /wp-admin/index.php last;
rewrite ^/wp-admin/([#?].*)$ /wp-admin/index.php$1 last;
# Include security directives from iThemes Security
include "/var/www/vhosts/<domain-path>/nginx.conf";

# Begin until corrected in iTS
rewrite /wp-config.php /forbidden last;
rewrite /install.php /forbidden last;
rewrite ^/wp-includes/(.*).php /forbidden last;
rewrite ^/wp-admin/includes(.*)$ /forbidden last;
rewrite ^(.*)/uploads/(.*).php(.?) /forbidden last;
location = /forbidden { deny all; }
# End until corrected in iTS

# error_log /var/log/nginx/error.log debug;

# Plesk's "location /" sends the resource request to Apache via Ports 7080/7081
# It is the ONLY prefix string location in the Plesk config, which means
# it's always the winning prefix location, and gets noted as such.
# - UNLESS we add any longer prefix string locations...
# Implying that if we want to use Nginx for everything, then we must add
# "location" statements matching ALL resource types we expect, with "try_files".
# Security via e.g. "try_files" is NOT an issue at "location /", that's then Apache's job.

# This prefix string location gets tested in phase 1, pre-empts "location /" since longer
location = /robots.txt {
 allow all;
 log_not_found off;
 access_log off;
}

# Then comes regex location processing, *** which terminates on the first match ***, and
# the corresponding location configuration is used. If no match with a regular expression
# is found, then the configuration of the prefix location remembered earlier is used.
# SO the order in which regex locations appear is IMPORTANT!

# Copied from Plesk "Handle PHP by Nginx" config - access to statistics via Apache
location ~ ^/(plesk-stat|webstat|webstat-ssl|ftpstat|anon_ftpstat|awstats-icon) {
    proxy_pass https://##YOUR-IP##:7081;
    proxy_set_header Host             $host;
    proxy_set_header X-Real-IP        $remote_addr;
    proxy_set_header X-Forwarded-For  $proxy_add_x_forwarded_for;
    proxy_set_header X-Accel-Internal /internal-nginx-static-location;
    access_log off;
}

# PHP-Handling: Best of Plesk and ownCloud and Web-Forum articles
# A match here pre-empts Plesk's "location /", sends the script to PHP-FPM
# "(?U)" sets internal "ungreedy" option, i.e. to FIRST ".php" - COOL
location ~ ^((?U).+?\.php)(/.*)?$ {
  # TR: See http://wiki.nginx.org/HttpFastcgiModule#fastcgi_split_path_info
```

```
  # TR: Directive populates $fastcgi_script_name + $fastcgi_path_info
  # TR: BUT there's a problem: http://trac.nginx.org/nginx/ticket/321
  fastcgi_split_path_info ^((?U).+\.php)(/?.+)$;
  set $path_info $fastcgi_path_info;
  # Requested PHP Scripts MUST exist exactly as named
  try_files $fastcgi_script_name =404;
  # try_files has emptied $fastcgi_path_info
  fastcgi_param PATH_INFO $path_info;
  # You can add any application-special headers to $_SERVER here
  fastcgi_pass "unix:/var/www/vhosts/system/<domain-name>/php-fpm.sock";
  # TR: Sets SCRIPT_FILENAME as $document_root$fastcgi_script_name
  # TR: Meaning that $fastcgi_script_name MUST be correct now!
  include /etc/nginx/fastcgi.conf;
  # access_log off;
}
# Plesk's Directives for Web User Access are not needed on this domain!

# The "generic" handling of static resources
# Note that they are all formulated securely with "try_files".

charset utf-8;
location = /nginx.conf { deny all; }
location = /wordpress.conf { deny all; }

# PUT MOST FREQUENT FIRST
# try_files "uses the first found file for request processing; the processing is performed
in the current context"

# Frequent possibly modified content valid 2 weeks in Cache
location ~* \.(js|css|htm|html|xhtml|xml|json)$ {
 try_files $uri =404;
 expires 2w;
 add_header Pragma public;
 add_header Cache-Control public;
 access_log off;
 # add_header Debug Frequent;
}

# Pics and Fonts valid 90 Days in Cache
location ~* \.(png|jpg|jpeg|gif|ico|bmp|img|ttf|otf|eot|svg|svgz|woff)$ {
 try_files $uri =404;
 expires 90d;
 add_header Pragma public;
 add_header Cache-Control public;
 access_log off;
 # add_header Debug PicFont;
}

# Zips + PDF valid 2 weeks in Cache
location ~* \.(bz2|exe|gz|pdf|rar|tgz|zip)$ {
 try_files $uri =404;
 expires 2w;
 add_header Pragma public;
 add_header Cache-Control public;
```

```
 access_log off;
 # add_header Debug ZipPDF;
}

# Media files (large) valid 1 week in Cache
location ~* \.(ac3|avi|flv|iso|mp3|mp4|mpeg|mpg|ogg|qt|rm|swf|wav)$ {
 try_files $uri =404;
 expires 1w;
 add_header Pragma public;
 add_header Cache-Control public;
 access_log off;
 # add_header Debug Media;
}

# Infrequent possibly modified content valid 3 days in Cache
location ~* \.(dat|doc|docx|dts|ppt|pptx|tar|txt|xls|xlsx)$ {
 try_files $uri =404;
 expires 3d;
 add_header Pragma public;
 add_header Cache-Control public;
 access_log off;
 # add_header Debug Infrequent;
}

location ~ ^/$ {
 index index.php;
}

# Catchall: Catch permalinks and special slugs
# ?q=$uri&$args after /index.php is common but counterproductive: Cache-Plugins see
# GET parameters and may therefore not cache the page, fearing dynamic content.
# But WordPress is clever, it can infer Permalinks from the SERVER-Environment
# (QUERY_STRING, REQUEST_URI), so omit the GET parameters.
location ~ ^/.+$ {
 try_files $uri $uri/ /index.php;
}

# Finally, switch gzipping on
gzip on;
gzip_proxied any;
gzip_min_length 100;
gzip_buffers 8 16k; # number size, default 32 4k|16 8k
gzip_types text/css text/plain text/javascript application/javascript application/json
application/x-javascript application/xml application/xml+rss application/xhtml+xml
application/x-font-ttf application/x-font-opentype application/vnd.ms-fontobject
image/svg+xml image/x-icon application/rss+xml application/atom_xml;
gzip_vary on;
gzip_comp_level 9;
gzip_http_version 1.0;
gzip_disable "MSIE [1-6]\.(?!.*SV1)";
```

PiWiK

Benutze die generischen Direktive oben. Sie bringen in PiWiK selbst wenig, weil PiWiK anscheinend andere Header erzeugt, die sie außer Kraft setzen; aber wichtig für die piwik.js, die von den Client-Websites includiert wird – dieses muss gezippt werden!

Vorbemerkungen ownCloud und eGroupware

Das „alte“ eGroupware 1.8 kann nicht selbst mit UTF-8 Strings umgehen und benötigt daher die php.ini Anweisung mbstring.func_overload = 7, was bei moderneren Anwendungen wie ownCloud wiederum unzulässig ist. Man kann die beiden also nicht bequem nebeneinander in Unterverzeichnissen einer Domain betreiben (obwohl jeder für sich kein Problem mit dem Betriebsmodus hat). Die neue Version 14.1 von eGW, die gerade im Anmarsch ist, behebt die Unschönheit mit mbstring.

Auch die Beispiel-Configs (Apache, Nginx) im Internet gehen eher davon aus dass das System direkt in der Domain lebt und nicht in einem Unterverzeichnis, so dass man dazu die Configs anpassen muss. Ferner hat man je nachdem, welche Anwendungen nebeneinander in der Domain laufen, womöglich Probleme deren unterschiedliche Bedürfnisse in der Webserver-Konfig abzubilden. Daher ist es einfacher (und sicherer – mit einer besseren Abschottung), **die Systeme jeweils in eigenen Subdomains** einzurichten, z.B. owncloud.-my.com.

ownCloud

ownCloud bietet nicht nur WebDAV-Zugang zu Dateien am Server, sondern auch mittels eines Clients die Möglichkeit, die Dateien auch lokal (auf dem eigenen PC) vorzuhalten und die Datenbestände in beiden Richtungen zu synchronisieren. Das ermöglicht ein flottes lokales Arbeiten sowie ohne Internet-Anschluss zu arbeiten (z.B. im Zug).

Was ownCloud verständlicherweise etwas in die Knie zwingt, ist riesige Mengen kleiner Dateien. Diese kommen vor allem dann vor, wenn man lokal Archive auspackt – Programmpakete wie ownCloud und eGroupware, oder Info-Archive wie SelfHtml. Daher Archive unbedingt auf einem Bereich der lokalen Festplatte auspacken, der **nicht** synchronisiert wird!

Folge der Anleitung hier; gute Tipps auch hier, hier.

Abgeschaut von .htaccess von ownCloud-Installation müssen auch in PHP folgende Ini-Werte gesetzt werden, in Plesk unter <Domain> | PHP Settings:

Feld memory_limit :: 512M

Feld max_execution_time :: 60

Feld post_max_size :: 513M

Feld upload_max_filesize 513M

open_basedir :: None

Additional configuration directives:

expose_php = off

cgi.fix_pathinfo = 0

date.timezone = Europe/Berlin

variables_order = "EGPCS" ???

Nginx Directive

Behandlung von .php Dateien: Auf dem ersten Blick sieht der ownCloud-Vorschlag um einiges anders aus als das, was Plesk sonst macht. Genauer betrachtet sind sie effektiv – wenn man die versch. Umgebungen berücksichtigt – gleich, nur anders formuliert; bis darauf dass in ownCloud ein „try_files $1 =404;“ zusätzlich vorkommt.

Insofern die Plesk-Konfig für php-Dateien auch die Behandlung von Plesk Web-User beinhaltet halte ich es für besser, auch hier...

- Serve static files directly by nginx: OHNE Häkchen
- Process PHP by nginx: MIT Häkchen (wie sonst immer)

Doch wir haben ein weiteres Problem:

- client_max_body_size – bei Plesk FEST im Template mit 128m, ownCloud Beispiel-Config sagt 10G – was mir übertrieben vorkommt. Ein Override ist nicht möglich, hier brauchen wir „Die letzte Rettung: Custom Configuration Template“ - aber mehr als 1G habe ich nicht übers Herz gebracht, da das Template für ALLE Domains gilt.

Die Distribution von ownCloud (Server) lieferte eine .htaccess Datei mit, aber nichts für Nginx. Im Folgendem gehe ich die .htaccess Element für Element durch, was es ist und wie man es unter Nginx / Plesk-Nginx abbilden kann:

ownCloud .htaccess Schritt für Schritt zu Nginx konvertieren

.htaccess	Erklärung	Nginx (allg.)	Nginx (Plesk 11.5)

`<IfModule mod_fcgid.c>`
`<IfModule mod_setenvif.c>`
`<IfModule mod_headers.c>`
`SetEnvIfNoCase ^Authorization$ "(.+)" XAUTHORIZATION=$1`
`RequestHeader set XAuthorization %{XAUTHORIZATION}e env=XAUTHORIZATION`
`</IfModule>`
`</IfModule>`
`</IfModule>`
	FALLS mod_fggid (also FastCGI) im Einsatz ist und Module setenvif und headers verfügbar sind, dann (1) falls Authorization request-header vorhanden, dann setze die Umgebungsvariable XAUTHORIZATION gleich ihrem Wert; (2) fügt einen Xauthorization request-header mit gleichem Wert hinzu, aber nur falls (1) passiert ist.	`fastcgi_param XAUTHORIZATION "$http_authorization" if_not_empty;`
		
		`Übrigens: Im Code von ownCloud wird nur 'HTTP_XAUTHORIZATION' abgefragt; daher zusätzlich:`
		
		`fastcgi_param HTTP_XAUTHORIZATION "$http_authorization" if_not_empty;`
			Directive müssen in ein Custom Template aufgenommen werden!
			
			`custom/domain/service/fpm.php`

`ErrorDocument 403 /core/templates/403.php` `ErrorDocument 404 /core/templates/404.php`		`error_page 403 /core/templates/403.php;` `error_page 404 /core/templates/404.php;`	In eigener nginx.conf
php_value...	Überlagern php.ini Werte		Plesk (s. oben): Domain | PHP Settings
`SetEnv htaccessWorking true`	ownCloud stellt die Wirksamkeit von .htaccess anhand dieser Umgebungsvariable fest. Für Nginx irrelevant.	-	-
`RewriteRule .* - [env=HTTP_AUTHORIZATION:%{HTTP:Authorization}]`	Stellt sicher dass wenigstens HTTP_AUTHORIZATION in der Umgebung gesetzt ist	`fastcgi_param HTTP_AUTHORIZATION "$http_authorization";`	
`RewriteRule ^.well-known/host-meta /public.php?service=host-meta [QSA,L]` `RewriteRule ^.well-known/host-`	Webfinger, d.h. nicht lebenswichtig Achtung: QSA bewirkt dass ein Query-String im Request angehängt	`rewrite ^/.well-known/host-meta /public.php?service=host-meta last;` `rewrite ^/.well-known/host-`	In eigener nginx.conf

.htaccess	Erklärung	Nginx (allg.)	Nginx (Plesk 11.5)
`meta.json /public.php?service=host-meta-json [QSA,L]` `[QSA,L]: Siehe unten!`	wird (sonst nicht da Replacement schon ein Query String enthält).	`meta.json /public.php?service=host-meta-json last;`	
`RewriteRule ^.well-known/carddav /remote.php/carddav/ [R]` `RewriteRule ^.well-known/caldav /remote.php/caldav/ [R]`	[R] = Sendet Redirect an den Browser mit Default-Code 302; ebenso „redirect“ in Nginx.	`rewrite ^/.well-known/carddav /remote.php/carddav/ redirect;` `rewrite ^/.well-known/caldav /remote.php/caldav/ redirect;`	In eigener nginx.conf

-	Diese Directive entstammen dem Tutorial von ownCloud und machen verkürzte URLs möglich.	`rewrite ^/caldav(.*)$ /remote.php/caldav$1 redirect;`
		`rewrite ^/carddav(.*)$ /remote.php/carddav$1 redirect;`
		`rewrite ^/webdav(.*)$ /remote.php/webdav$1 redirect;`
			In eigener nginx.conf

`RewriteRule ^apps/calendar/caldav.php remote.php/caldav/ [QSA,L]` `RewriteRule ^apps/contacts/carddav.php remote.php/carddav/ [QSA,L]`	Ähnlich wie oben, Alternativ-URLs. Was 'rauskommen muss ähnelt "remote.php/caldav/principals/username/" - d.h. **kein** query string sondern PATH_INFO, daher "$1".	`rewrite ^/apps/calendar/caldav(.*)$ /remote.php/caldav$1 last;` `rewrite ^/apps/contacts/carddav(.*)$ /remote.php/carddav$1 last;`	In eigener nginx.conf
`RewriteRule ^apps/([^/]*)/(.*\.(php))$ index.php?app=$1&getfile=$2 [QSA,L]`	Direktaufrufe von Apps werden über index.php umgelenkt. Subpattern (php) scheint nutzlos, $3 nicht benutzt – entfernt!	`rewrite ^/apps/([^/]*)/(.*\.php)$ index.php?app=$1&getfile=$2 last;`	In eigener nginx.conf
`RewriteRule ^remote/(.*) remote.php [QSA,L]`	Aufrufe an „/remote/...“ werden auf remote.php umgelenkt – aber $1 = (.*) geht verloren (Query-String aber nicht). Verbessert...	`rewrite ^/remote/(.+) /remote.php/$1 last;`	In eigener nginx.conf
`AddType image/svg+xml svg svgz` `AddEncoding gzip svgz`	Make SVGZ fonts work on iPad. AddEncoding bewirkt dass die Datei	Type bereits enthalten in /etc/nginx/mime.types	-

.htaccess	Erklärung	Nginx (allg.)	Nginx (Plesk 11.5)
	als gzip-enkodiert markiert wird.	Ein Pendant zu AddEncoding konnte ich nicht finden!	
`AddDefaultCharset utf-8`		`charset utf-8;`	In eigener nginx.conf

Nginx akzeptiert fastcgi_param nicht innerhalb von einem „if“. If is evil (innerhalb einer location). Aber: „location“ wird auch nicht innerhalb eines „if“ akzeptiert! Gott sei Dank gibt es „if_not_empty“. Und hier liest man, Warum try_files!

[QSA,L] (Apache)

RewriteRule ^remote/(.*) remote.php [QSA,L]   # remote/123?abc => remote.php?abc

By default, the query string is passed through unchanged. When the replacement URI contains a query string, the default behavior of RewriteRule is to discard the existing query string, and replace it with the newly generated one. Using the [QSA] flag causes the query strings to be combined. Consider the following rule:

```
RewriteRule /pages/(.+) /page.php?page=$1 [QSA]
```

With the [QSA] flag, a request for /pages/123?one=two will be mapped to /page.php?page=123&one=two.

Without the [QSA] flag, that same request will be mapped to /page.php?page=123 - that is, the existing query string will be discarded.

Nginx:

rewrite: By default, the query string is passed through unchanged.

Putting a question mark at the end of a replacement string avoids having it appended

rewrite ... redirect: returns a temporary redirect with the 302 code

Das Ergebnis (für Plesk) ist dann:

```
# THIS Example file (ownCloud) assumes Case A2!!!
# You MUST insert your domain name at line 102

# A) IF the Plesk Domain "Web Server Setting" "Process PHP by nginx" is OFF then...
# 1) all ".php" script calls will be passed to Apache; use here as is.
# 2) BUT if you have manually activated PHP-FPM by cloning the domains config
#    in /etc/php5/fpm/pool.d, then you can leave it OFF but yourself pass all the
#    script requests to php-fpm - uncomment that block below, set domain in fastcgi_pass

# B) IF the Plesk Domain "Web Server Setting" "Process PHP by nginx" is ON then...
# Plesk sets a "location ~ \.php(/.*)?$ {" which passes the scripts to PHP-FPM.
# The problem is, that Plesk's directive block does NOT include a "try_files",
# which leaves the setup insecure.

# After each modification: /etc/init.d/nginx restart
# Server Error log: /var/log/nginx/error.log
# Domain Error log: /var/www/vhosts/system/<domain>/logs/error_log
# http://wiki.nginx.org/Pitfalls
# http://www.php.net/manual/de/regexp.introduction.php

# Plesk's "location /" sends the resource request to Apache via Ports 7080/7081
# Plesk's "location /" is the ONLY prefix string location in the Plesk config,
# which means it's always the winning prefix location, and gets noted as such.
# - UNLESS we add any longer prefix string locations...
# Implying that if we want to use Nginx for everything, then we must add
# "location" statements matching ALL resource types we expect, with "try_files".
# Security via e.g. "try_files" is NOT an issue at "location /", that's then Apache's job.

# error_log /var/log/nginx/error.log debug;

fastcgi_buffers 32 8K; # nginx default 8 4k|8k / ownCloud says 64 4K

index index.php index.html;

error_page 403 /core/templates/403.php;
error_page 404 /core/templates/404.php;

rewrite ^/.well-known/host-meta /public.php?service=host-meta last;
rewrite ^/.well-known/host-meta.json /public.php?service=host-meta-json last;

rewrite ^/.well-known/carddav /remote.php/carddav/ redirect;
rewrite ^/.well-known/caldav /remote.php/caldav/ redirect;

# Makes abbreviated URLs work
# What we NEED to come is like "remote.php/caldav/principals/username/"
# i.e. this is NOT a query string but PATH_INFO, so "$1" is needed!
rewrite ^/caldav(.*)$ /remote.php/caldav$1 redirect;
rewrite ^/carddav(.*)$ /remote.php/carddav$1 redirect;
rewrite ^/webdav(.*)$ /remote.php/webdav$1 redirect;

# Makes alternative URLs work
```

```
rewrite ^/apps/calendar/caldav(.*)$ /remote.php/caldav$1 last;
rewrite ^/apps/contacts/carddav(.*)$ /remote.php/carddav$1 last;

rewrite ^/apps/([^/]*)/(.*\.php)$ index.php?app=$1&getfile=$2 last;

rewrite ^/remote/(.+) /remote.php/$1 last;

rewrite ^(/core/doc/[^\/]+/)$ $1/index.html;

# Plesk's "location /" sends the resource request to Apache via Ports 7080/7081
# Plesk's "location /" is the ONLY prefix string location in the Plesk config,
# which means it's always the winning prefix location, and gets noted as such.
# - UNLESS we add any longer prefix string locations...
# Implying that if we want to use Nginx for everything, then we must add
# "location" statements matching ALL resource types we expect, with "try_files".
# Security via e.g. "try_files" is NOT an issue at "location /", that's then Apache's job.

# This prefix string location gets tested in phase 1, pre-empts "location /" since longer
location = /robots.txt {
 allow all;
 log_not_found off;
 access_log off;
}

# Then comes regex processing, *** which terminates on the first match ***, and the
# corresponding location configuration is used. If no match with a regular expression
# is found, then the configuration of the prefix location remembered earlier is used.
# SO the order in which regex locations appear is IMPORTANT!

# Hide sensitive areas not needing direct user access
# *First* regex match attempt ensures deny all really happens
location ~ ^/(data|config|\.ht|db_structure\.xml|README) { deny all; }

# PHP-Handling: Best of Plesk and ownCloud and Web-Forum articles
# A match here pre-empts Plesk's "location /", sends the script to PHP-FPM
# "(?U)" sets internal "ungreedy" option, i.e. to FIRST ".php" - COOL
location ~ ^((?U).+?\.php)(/.*)?$ {
  # TR: See http://wiki.nginx.org/HttpFastcgiModule#fastcgi_split_path_info
  # TR: Directive populates $fastcgi_script_name + $fastcgi_path_info
  # TR: BUT there's a problem: http://trac.nginx.org/nginx/ticket/321
  fastcgi_split_path_info ^((?U).+\.php)(/?.+)$;
  set $path_info $fastcgi_path_info;
  # Requested PHP Scripts MUST exist exactly as named
  try_files $fastcgi_script_name =404;
  # try_files has emptied $fastcgi_path_info
  fastcgi_param PATH_INFO $path_info;
  # You can add any application-special headers to $_SERVER here
  fastcgi_param XAUTHORIZATION "$http_authorization" if_not_empty;
  fastcgi_param HTTP_XAUTHORIZATION "$http_authorization" if_not_empty;
  fastcgi_pass "unix:/var/www/vhosts/system/<domain>/php-fpm.sock";
  # TR: Sets SCRIPT_FILENAME as $document_root$fastcgi_script_name
  # TR: Meaning that $fastcgi_script_name MUST be correct now!
  include /etc/nginx/fastcgi.conf;
  # access_log off;
```

```
}
# Plesk's Directives for Web User Access are not needed on this domain!

# The "generic" handling of static resources
# Note that they are all formulated securely with "try_files".

charset utf-8;
location = /nginx.conf { deny all; }
location = /owncloud.conf { deny all; }

# PUT MOST FREQUENT FIRST
# try_files "uses the first found file for request processing; the processing is performed
in the current context"

# Frequent possibly modified content valid 2 weeks in Cache
location ~* \.(js|css|htm|html|xhtml|xml|json)$ {
 try_files $uri =404;
 expires 2w;
 add_header Pragma public;
 add_header Cache-Control public;
 access_log off;
}

# Pics and Fonts valid 90 Days in Cache
location ~* \.(png|jpg|jpeg|gif|ico|bmp|img|ttf|otf|eot|svg|svgz|woff)$ {
 try_files $uri =404;
 expires 90d;
 add_header Pragma public;
 add_header Cache-Control public;
 access_log off;
}

# Zips + PDF valid 2 weeks in Cache
location ~* \.(bz2|exe|gz|pdf|rar|tgz|zip)$ {
 try_files $uri =404;
 expires 2w;
 add_header Pragma public;
 add_header Cache-Control public;
 access_log off;
}

# Media files (large) valid 1 week in Cache
location ~* \.(ac3|avi|flv|iso|mp3|mp4|mpeg|mpg|ogg|qt|rm|swf|wav)$ {
 try_files $uri =404;
 expires 1w;
 add_header Pragma public;
 add_header Cache-Control public;
 access_log off;
}

# Infrequent possibly modified content valid 3 days in Cache
location ~* \.(dat|doc|docx|dts|ppt|pptx|tar|txt|xls|xlsx)$ {
 try_files $uri =404;
 expires 3d;
```

```
 add_header Pragma public;
 add_header Cache-Control public;
 access_log off;
}

# Virtually nothing should arrive here: We've already (hopefully) done any
# special protected / allowed locations, all ".php" scripts, all static assets.

location ~* ^/.* {
  # Allow the unrecognised type to be a file or directory, reject all else
  try_files $uri $uri/ =404;
  # Let this be logged – we want to know of anything untoward!
}

# NOTE: The gzipping block is omitted here - causes fails by ownCloud Client!
# This is because Nginx suppresses the E-Tag when gzipping, see:
# https://github.com/owncloud/mirall/issues/1291
# With gzipping the client activity log shows errors like this:
# Kein E-Tag vom Server empfangen, bitte Proxy / Gateway überprüfen
```

eGroupware 1.8

Siehe hier, hier, hier; auf einer eigenen Umsetzung habe ich verzichtet, da eGW 14.1 kurz vor der Freigabe steht (bzw. stand).

Mantis (Bugtracker)

Mantis verwendet keine .htaccess – keine Nginx-Direktive nötig.

Plesk Web-Stats und Nginx

Plesk bietet generell die Möglichkeit, auf Statistische Auswertung von awStats basiered auf den Server-Logs zuzugreifen. Der Weg ist etwas steinig.

Zunächst: Was ist Wo bei Plesk 12.

Das Statistics-Verzeichnis wurde Umgezogen (für bessere Sicherheit wohl):

/<VHOST>/statistics => /system/<VHOST>/statistics

Damit erfordert der Zugriff darauf etwas Klimzüge in der Webserver-Config.

Ferner: Das Verzeichnis ist passwort-geschützt. Gibt man das URL dafür ein:

https://<domain>/plesk-stat/webstat/

https://<domain>/plesk-stat/webstat-ssl/

wird man um ein HTTP-Authentifizierung „Domain statistics“ gebeten, das sind die Daten vom Kunden bzw. WorkSpace (FTP). Man kann sie auch in Plesk selber setzen:

Domain Settings | Password-Protected Directories | /plesk-stat

Siehe Verzeichnis: /var/www/vhosts/system/<VHOST>/pd

PROBLEM: Plesk 11.5 erzeugt die Alias-Anweisungen für Statistics nur in der Apache-Config!

LÖSUNG: Auf Plesk 12 aufsteigen, dort gibt es in der Plesk-Config einen Abschnitt, der diese Anfragen zu Apache weiterleitet, der sieht so aus:

```
location ~ ^/(plesk-stat|webstat|webstat-ssl|ftpstat|anon_ftpstat|awstats-icon) {
    proxy_pass https://##YOUR-IP##:7081;
    proxy_set_header Host             $host;
    proxy_set_header X-Real-IP        $remote_addr;
    proxy_set_header X-Forwarded-For  $proxy_add_x_forwarded_for;
    proxy_set_header X-Accel-Internal /internal-nginx-static-location;
    access_log off;
}
```

Diese Anweisungen kann man dem eigenen Nginx-Config auch hinzufügen (nötig wenn „Process PHP by nginx“ kein Häkchen hat), oder wenn man nicht auf Plesk 12 aufsteigen will kann man sie in Plesk 11.5 nachrüsten (siehe Die letzte Rettung: Custom Configuration Template). Das geht dann so (hier zur Beseitigung von PoodleBleed):

```
mkdir /usr/local/psa/admin/conf/templates/custom
mkdir /usr/local/psa/admin/conf/templates/custom/domain

cp /usr/local/psa/admin/conf/templates/default/domain/nginxDomainVirtualHost.php
/usr/local/psa/admin/conf/templates/custom/domain

EDIT these lines:

ssl_protocols TLSv1.2 TLSv1.1 TLSv1;
ssl_ciphers
ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:ECDH+3DES:DH+3DES:RSA+AESGC
M:RSA+AES:RSA+3DES:!aNULL:!MD5:!DSS;

/usr/local/psa/admin/bin/httpdmng --reconfigure-all
```

ACHTUNG Forking a custom template means that one does NOT get any future Plesk improvements to it!