|  |
| --- |
| Informationstechnologie |
| Dokumentenart | Arbeitsunterlage |
| Titel | **Apache-Schulung: Lektion 3** |
| Nummer | 2 |
| Autor/-in | Christian Folini |
| Kontaktangaben | +41 79 220 23 76christian.folini@post.ch |
| Ausgabestelle | IT222 |
| Geltungsbereich | Internet / WWW |
| Klassifizierung | intern |
| Version | X01.00 |
| Ausgabedatum | 23. Oktober 2015 |
| Ersetzt Ausgabe vom | 1. Januar 1970 |

Apache Schulungszentrale: <https://bullnix.post.ch/foswiki/bin/view/BasicServices/ApacheSchulungen>

Quellen:
<https://github.com/Apache-Labor/labor/blob/master/labor-02/README.md>

<https://www.netnea.com/cms/apache_tutorial_3_applikationsserver_aufsetzen/>

## Aufsetzen eines Apache/PHP Applikationsservers

### **Was machen wir?**

Wir setzen einen Apache / PHP Applikationsserver mit den minimal nötigen Modulen auf

### **Warum tun wir das?**

Ein nackter Apache eignet sich, um statische Files auszuliefern. Komplexere Applikationen bauen auf dynamischem Content auf. Dies bedingt eine Erweiterung des Servers. Ein sehr schneller und sicherer Applikationsserver lässt sich mit Suexec, einem externen Fast-CGI-Daemon und PHP realisieren. Dies ist bei weitem nicht die einzige Variante und im Unternehmensumfeld vermutlich auch nicht die verbreitetste. Aber es ist eine sehr einfache Architektur, die sich hervorragend für ein Testsystem eignet.

### **Voraussetzungen**

* Ein Apache Webserver, idealerweise mit einem File-Layout wie bei [Anleitung 1 (Kompilieren eines Apache Servers)](http://www.netnea.com/cms/apache_tutorial_1_apache_compilieren/) erstellt.
* Verständnis der minimalen Konfiguration in [Anleitung 2 (Apache minimal konfigurieren)](http://www.netnea.com/cms/apache_tutorial_2_apache_minimal_konfigurieren/).

### **Schritt 1: Apache konfigurieren**

Wir konfigurieren den Webserver vorneweg; wohlwissend, dass er in dieser Konfiguration noch nicht lauffähig ist. Basierend auf dem in Lektion 2 beschriebenen minimalen Webserver konfigurieren wir einen sehr einfachen Applikationsserver in der Datei conf/httpd.conf.

ServerName localhost

ServerAdmin root@localhost

ServerRoot /apache

User www-data

Group www-data

ServerTokens Prod

UseCanonicalName On

TraceEnable Off

Timeout 5

MaxRequestWorkers 250

Listen 127.0.0.1:80

LoadModule mpm\_event\_module modules/mod\_mpm\_event.so

LoadModule unixd\_module modules/mod\_unixd.so

LoadModule log\_config\_module modules/mod\_log\_config.so

LoadModule mime\_module modules/mod\_mime.so

LoadModule authn\_core\_module modules/mod\_authn\_core.so

LoadModule authz\_core\_module modules/mod\_authz\_core.so

LoadModule suexec\_module modules/mod\_suexec.so

LoadModule fcgid\_module modules/mod\_fcgid.so

ErrorLogFormat "[%{cu}t] [%-m:%-l] %-a %-L %M"

LogFormat "%h %l %u [%{%Y-%m-%d %H:%M:%S}t.%{usec\_frac}t] \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined

LogLevel debug

ErrorLog logs/error.log

CustomLog logs/access.log combined

AddHandler fcgid-script .php

DocumentRoot /apache/htdocs

<Directory />

 Require all denied

 Options SymLinksIfOwnerMatch

 AllowOverride None

</Directory>

<VirtualHost 127.0.0.1:80>

 <Directory /apache/htdocs>

 Require all granted

 Options ExecCGI

 AllowOverride None

 FCGIWrapper /apache/bin/php-fcgi-starter/php-fcgi-starter .php

 </Directory>

</VirtualHost>

Ich gehe nicht mehr auf die gesamte Konfiguration ein, sondern nur noch auf die Abweichungen zur Anleitung 2. Hinzugekommen sind drei Module: Neben dem Suexec- und dem Fcgi-Modul ist es das Mime-Modul, welches uns erlaubt, die Dateiendung .php dem Fcgi-Daemon zuzuweisen. Diese Zuweisung erfolgt mittels der AddHandler-Direktive.

Das Verzeichnis /apache/htdocs benötigt neu die zusätzliche Option ExecCGI. Schliesslich der FCGIWrapper. Das ist das Verbindungsstück zwischen dem Webserver und dem noch zu konfigurierenden FCGI-Daemon. Sobald die erste Anfrage mit der Endung .php an den Webserver gestellt wird, ruft der Server ein Wrapper-Script auf und startet damit den FCGI-Daemon, der sich ab diesem Moment um die PHP-Anfragen kümmert.

Fast-CGI\_ ist eine Technik um dynamischen Programmcode aus einem Webserver heraus aufzurufen. Es ist eine sehr schnelle Technik, welche den Server weitgehend unberührt lässt und die Applikation auf einem separaten Daemon ausführt. Um die Geschwindigkeit zu steigern stellt Fast-CGI mehrere Instanzen dieses Daemons bereit, so dass eine Anfrage ohne Wartepause verarbeitet werden kann. In der Praxis ist dies ein ansprechender Performance-Gewinn und vor allem eine Architektur, welche Speicher spart, wie unten genauer erklärt wird.

### **Schritt 2: Apache mir Suexec-Unterstützung kompilieren**

Wir müssen nun zwei fehlende Module kompilieren und weitere Komponenten für den FCGI-Daemon bereitstellen. Beginnen wir mit dem Suexec-Modul.

In Lektion 1 wurde ein Apache Webserver selbst kompiliert. Aber obschon dort die Option --enable-mods-shared=all verwendet wurde, ist suexec noch nicht kompiliert. Das Modul ist demnach so speziell, dass es nicht als Standard-Modul mitkompiliert werden kann, obschon es im Quelltext vorhanden ist.

Der in Lektion 2 konfigurierte Webserver läuft als Benutzer www-data oder je nach Konfiguration als beliebiger anderer dedizierter Benutzer. Unsere dynamische Applikation möchten wir noch weiter eingrenzen und den separaten Daemon unter einem weiteren separaten User drehen lassen. Dies wird uns durch das Modul suexec ermöglicht. Diese Massnahme ist nicht zwingend. Aber sie bringt ein Mehr an Sicherheit mit wenig zusätzlichem Aufwand.

Begeben wir uns ins Verzeichnis mit dem Apache Quelltext und kompilieren den Server neu.

$> cd /usr/src/apache/httpd-2.4.17

$> ./configure --prefix=/opt/apache-2.4.17 --enable-mods-shared=all --with-apr=/usr/local/apr/bin/apr-1-config --with-apr-util=/usr/local/apr/bin/apu-1-config --enable-mpms-shared="event worker" --enable-nonportable-atomics=yes --enable-suexec --with-suexec-caller=www-data --with-suexec-docroot=/opt/apache-2.4.17/bin && make && sudo make install

Zum bekannten configure sind nun drei Optionen hinzugekommen, die sich um das Suexec kümmern. Enable-suexec spricht für sich, with-suexec-caller teilen wir dem gewissenhaften Modul mit, dass ausschliesslich der User www-data die Erlaubnis erhalten soll, das Modul mit dem dahinter liegenden Programm aufzurufen. Schliesslich geben wir dem Modul noch bekannt, wo die aufzurufenden Skripte liegen sollen. Der Einfachheit halber nehmen wir das existierende bin-Verzeichnis. Allerdings ist suexec pingelig und wir können nicht mit dem Symlink arbeiten. Es muss also der voll qualifizierte Pfad sein.

Nach der erfolgreichen Konfiguration wird obenstehende Befehlszeile den Compiler aufrufen und nach dessen erfolgreichem Abschluss den neu-kompilierten Server installieren.

Übrigens: Wenn man bei der obenstehenden Konfiguration einen Fehler macht und Suexec andere Compilerkonstanten mitgeben möchte, dann muss man vor dem erneuten Compilieren zuerst die Compiler-Umgebung reinigen. Ansonsten werden die neuen Optionen ignoriert. Dies können wir über den Befehl make clean vor dem configure erreichen, oder durch manuelles Löschen der Dateien support/suexec, support/suexec.lo und support/suexec.o, was schneller geht, weil danach nicht mehr der gesamte Webserver neu gebaut werden muss.

### **Schritt 3: Fast-CGI Modul herunterladen und kompilieren**

Das Fast-CGI-Modul wird von Apache verwaltet. Es ist aber nicht Teil des normalen Quellcodes des Webservers. Laden wir den Quelltext für das zusätzliche Modul also herunter und prüfen wir die über eine verschlüsselte Verbindung geladene Checksumme.

$> cd /usr/src/apache

$> wget http://www.apache.org/dist/httpd/mod\_fcgid/mod\_fcgid-2.3.9.tar.gz

$> wget https://www.apache.org/dist/httpd/mod\_fcgid/mod\_fcgid-2.3.9.tar.gz.sha1

$> sha1sum --check mod\_fcgid-2.3.9.tar.gz.sha1

Wir erwarten wieder ein OK. Wenn dies korrekt retourniert wurde, ist es Zeit für das Entpacken, Kompilieren und Installieren.

$> tar xvzf mod\_fcgid-2.3.9.tar.gz

$> cd mod\_fcgid-2.3.9

$> APXS=/apache/bin/apxs ./configure.apxs

$> make

$> sudo make install

Der Configure-Befehl hat hier ein etwas anderes Format, da es sich beim Fcgi-Modul um ein von Apache abhängiges Modul handelt. Wir verwenden deshalb APXS, das Apache Expansion Tool, das Teil des in Lektion 1 kompilierten Servers ist. Leider zerstört das make install teilweise die von uns gesetzten Besitzverhältnisse. Dies muss also nachjustiert werden.

$> sudo chown `whoami` /apache/conf/httpd.conf

Weiter benötigt der Apache-Benutzer Zugriff auf ein Verzeichnis, in das er Sockets anlegt, um mit dem FCGI-Daemon kommunizieren zu können. Dieses erstellen wir und übergeben es ihm direkt.

$> sudo mkdir /apache/logs/fcgidsock

$> sudo chown www-data:www-data /apache/logs/fcgidsock

### **Schritt 4: PHP installieren und vorkonfigurieren**

Bislang haben wir die ganze Software Stück für Stück selbst kompiliert. Beim ganzen PHP-Stack ist aber eine Grenze erreicht. Es soll niemandem verwehrt werden, PHP selbst zu kompilieren, hier konzentrieren wir uns aber auf den Webserver und übernehmen dieses Stück Software deshalb aus der Linux-Distribution. In Debian/Ubuntu heisst das entsprechende Paket php5-cgi und es zieht php5-common nach sich.

PHP richtig zu konfigurieren ist ein weites Feld und ich empfehle die einschlägigen Seiten zu konsultieren, denn ein falsch konfiguriertes PHP kann ein grosses Sicherheitsproblem darstellen. Hier möchte ich nicht mehr Informationen dazu geben, da es von unserem eigentlichen Thema, dem einfachen Applikationsserver, wegführen würde. Für den Betrieb im Internet, also nicht mehr im heimischen Laborbereich, ist es aber deutlich angezeigt, sich mit den relevanten PHP-Sicherheitseinstellungen vertraut zu machen.

### **Schritt 5: CGI User erstellen**

Oben wurde bereits beschrieben, dass wir planen, einen separaten Daemon zur Bearbeitung der PHP-Anfragen zu starten. Dieser Daemon soll mittels suexec gestartet werden und als eigenständiger Benutzer laufen. Wir erstellen diesen Benutzer folgendermassen:

$> sudo groupadd fcgi-php

$> sudo useradd -s /bin/false -d /apache/htdocs -m -g fcgi-php fcgi-php

Es ist zu erwarten, dass eine Warnung betreffend des vorhandenen Verzeichnisses /apache/htdocs erscheint. Diese können wir aber ignorieren.

### **Schritt 6: PHP Wrapper Skript erstellen**

Es ist üblich, PHP und FCGI mittels eines Wrapper-Skripts zusammenarbeiten zu lassen. Wir haben das oben auch schon in der Apache Konfiguration so vorgesehen. Der Webserver wird ausschliesslich dieses Skript aufrufen, während sich das Skript um alles weitere kümmert. Wir legen das Skript wie vorgesehen in /apache/bin/php-fcgi-starter/php-fcgi-starter ab. Das Unterverzeichnis ist nötig, denn suexec setzt voraus, dass das Verzeichnis dem vorkonfigurierten Benutzer gehört und wir möchten ./bin nicht komplett dem neuen Benutzer übergeben. Legen wir das Unterverzeichnis also an:

$> cd /apache/bin

$> sudo mkdir php-fcgi-starter

$> sudo chown fcgi-php:fcgi-php php-fcgi-starter

Wir müssen nun in diesem Verzeichnis ein Starter-Skript platzieren. Da wir das Verzeichnis bereits dem Benutzer fcgi-php übergeben haben, muss das Skript durch den Root-Benutzer erstellt werden. Oder durch in an diesen Ort kopiert werden. Das Erstellen eines Skripts in einem Verzeichnis, das uns nicht mehr gehört, ist bisweilen schwierig. Wir lösen das mit einem Trick mittels cat und einer Sub-Shell. Hier der Trick und daran anschliessend das Skript (Die Eingabe in cat wird mittels STRG-D abgeschlossen.).

$> sudo sh -c "cat > php-fcgi-starter/php-fcgi-starter"

#!/bin/sh

export PHPRC=/etc/php5/cgi/

export PHP\_FCGI\_MAX\_REQUESTS=5000

export PHP\_FCGI\_CHILDREN=5

exec /usr/lib/cgi-bin/php

Was legen wir hier fest? Wir geben PHP bekannt, wo sich seine Konfiguration befindet, legen die maximale Zahl der Requests eines Fcgi-Daemons auf 5'000 fest (danach wird er durch einen frischen Prozess ersetzt), wir bestimmen die Zahl der Prozess-Kinder auf 5 und rufen zu guter Letzt PHP selbst auf.

Nun nicht vergessen, das Skript auch dem FCGI-Benutzer zu übergeben und ausführbar machen:

$> sudo chown fcgi-php:fcgi-php php-fcgi-starter/php-fcgi-starter

$> sudo chmod +x php-fcgi-starter/php-fcgi-starter

### **Schritt 7: PHP-Testseite erstellen**

Zum Schluss sollten wir noch eine einfache php-basierte Testseite erstellen: /apache/htdocs/info.php.

<?php

phpinfo();

?>

### **Schritt 8: Ausprobieren**

Das war alles. Nun können wir den Webserver starten und ausprobieren.

$> sudo ./httpd -k start

Erreichbar ist unser Testskript unter der URL <http://localhost/info.php>.


Im Browser zeigt phpinfo einen umfassenden Statusbericht.

Ergibt der Start des Servers oder der Aufruf der URL eine Fehlermeldung, dann weiss das Fehler-Log des Servers oder das separate Suexec-Log unter logs/suexec\_log Abhilfe. Typische Fehler betreffen Besitz und Zugriffsrechte auf Verzeichnisse und Files.

Hier zur Zusammenfassung nochmals die relevanten Files und ihre Besitzer:

2107985 4 drwxr-xr-x 2 fcgi-php fcgi-php 4096 Jul 2 11:15 bin/php-fcgi-starter/

2107987 4 -rwxr-xr-x 1 fcgi-php fcgi-php 125 Jul 2 11:15 bin/php-fcgi-starter/php-fcgi-starter

2107977 32 -rwsr-xr-x 1 root root 32146 Jul 2 10:44 bin/suexec

2633547 4 drwxr-xr-x 2 myuser root 4096 Jul 2 11:16 htdocs/

2633758 4 -rw-r--r-- 1 myuser myuser 20 Jul 2 11:15 htdocs/info.php

2762281 4 drwx------ 2 www-data www-data 4096 Jul 2 10:46 /apache/logs/fcgidsock/

Auffällig ist das Suid-Bit auf dem Suexec-Binary.

### **Schritt 9 (Bonus): Ein kleiner Lasttest**

Der hier gebaute Applikations-Server ist im Vergleich zu einem Apache mit eingebautem PHP sehr leistungsfähig. Ein kleiner Lasttest kann dies illustrieren. Wir starten unseren Webserver im Daemon-Mode und benutzen Apache-Bench um ihm mit 5 Usern auf den Zahn zu fühlen. Neu ist die Option -l, welche das Werkzeug instruiert, auf Abweichungen in der Länge der Antworten nicht weiter einzugehen. Denn die Seite ist ja dynamisch generiert und der Inhalt, sowie dessen Länge, weichen natürlicherweise immer wieder ein wenig ab. Nach dem Lasttest beenden wir den Server wieder.

$> sudo ./bin/httpd -k start

$> ./bin/ab -c 5 -n 1000 -l http://localhost/info.php

...

$> sudo ./bin/httpd -k stop

In meinem Fall lieferte ab folgenden Output:

This is ApacheBench, Version 2.3 <$Revision: 1663405 $>

Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/

Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)

Completed 100 requests

Completed 200 requests

Completed 300 requests

Completed 400 requests

Completed 500 requests

Completed 600 requests

Completed 700 requests

Completed 800 requests

Completed 900 requests

Completed 1000 requests

Finished 1000 requests

Server Software: Apache

Server Hostname: localhost

Server Port: 80

Document Path: /info.php

Document Length: Variable

Concurrency Level: 5

Time taken for tests: 2.567 seconds

Complete requests: 1000

Failed requests: 0

Total transferred: 66892443 bytes

HTML transferred: 66739443 bytes

Requests per second: 389.63 [#/sec] (mean)

Time per request: 12.833 [ms] (mean)

Time per request: 2.567 [ms] (mean, across all concurrent requests)

Transfer rate: 25452.57 [Kbytes/sec] received

Connection Times (ms)

 min mean[+/-sd] median max

Connect: 0 0 0.1 0 1

Processing: 4 13 70.9 7 1147

Waiting: 2 12 70.8 6 1143

Total: 4 13 70.9 7 1147

Percentage of the requests served within a certain time (ms)

 50% 7

 66% 9

 75% 10

 80% 11

 90% 14

 95% 17

 98% 24

 99% 28

 100% 1147 (longest request)

Das sind 389 dynamische Requests pro Sekunde. Das sind viele. Besonders weil das Resultat von einem kleinen Testrechner stammt. Auf einem modernen Server in Produktionsgrösse lässt sich ein Vielfaches davon realisieren.

Bemerkenswert ist aber weniger die Geschwingigkeit des Systems als der Speicherverbrauch. Im Gegensatz zu einem Applikationsserver mit integriertem PHP-Modul haben wir den PHP-Stack hier ausgelagert. Das erlaubt es uns einen Apache Webserver mit Event-MPM zu benutzen. In einem integrierten Setup müssten wir den Prefork-MPM verwenden, der nicht mit Serverthreads, sondern mit speicherhungrigen Serverprozessen arbeitet. Und jeder dieser Prozesse würde dann auch noch das PHP-Modul laden, unbesehen davon, dass die meisten Requests in der Regel auf statische Applikationsteile wie Bilder, CSS, Javascripts etc. entfallen. Auf meinem Testsystem schlägt jeder Prefork-Apache-Prozess inklusive PHP mit 6 MB Resident Size zu Buche. Ein Event-Prozess mit lediglich 4 MB und die Zahl der externen FCGI-Prozesse bleibt deutlich kleiner.

### **Verweise**

* Apache: <http://httpd.apache.org>
* Apache FCGI: <http://httpd.apache.org/mod_fcgid>
* How2Forge PHP/FCGI: <http://www.howtoforge.com/how-to-set-up-apache2-with-mod_fcgid-and-php5-on-ubuntu-10.04>