
seeseekey.net
10K
SAS
600 GB
SRV01
NIC
PS1
PS2
PROCS
DIMMS
FANS
Selfhosting
Server aufsetzen und betreiben
Florian Bottke

seeseekey.net

Selfhosting

Server aufsetzen und betreiben

Florian Bottke

Selfhosting

Server aufsetzen und betreiben

Florian Bottke

seeseekey.net

Einleitung

Möchte man eine Webseite oder einen Blog ins Internet stellen, so benötigt man heute nicht einmal mehr einen Webhoster. Dienste wie Wordpress.com, Tumblr und Medium bieten jedem die Möglichkeit, schnell so etwas aufzusetzen. Nach der Registrierung dauert es nur ein paar Minuten und schon kann man den neuen Blog oder die Webseite mit Inhalten befüllen.

Natürlich begibt man sich dabei in eine Abhängigkeit. Denn, so bequem diese Dienste auch sein mögen, man sollte sich immer bewusst sein, dass sie doch nur ein riesiges Datensilo bilden, aus welchem man seine Daten unter Umständen nur schwer wieder heraus bekommt, und damit auf Gedeih und Verderben dem jeweiligen Anbieter ausgeliefert ist.

Eine andere Möglichkeit ist es, sich Webspace bei einem Webhoster zu mieten. Hier bekommt man meist ein sogenanntes LAMP-Paket, bestehend aus Linux, dem Webserver Apache, der Datenbank MySQL und der Skriptsprache PHP. Der Umfang der Angebote ist dabei von Webhoster zu Webhoster unterschiedlich. Bei dem einen bekommt man mehr Speicher, bei dem anderen statt dessen noch Python und Ruby als Skriptsprache dazu. Hier muss man als Nutzer abwägen, was den eigenen Anforderungen entspricht.

Die flexibelste aber auch arbeitsintensivste Methode ist es, seinen eigenen Server zu betreiben. Um das Aufsetzen und den Betrieb eines solchen soll es in diesem Buch gehen. Als Betriebssystem wird dabei ein Linux - in diesem Fall die Distribution Ubuntu in der Servervariante - genutzt.

Dieses Buch soll dem Leser dabei ein grundlegendes Verständnis über den Serverbetrieb und die damit verbundenen Aufgaben und Probleme vermitteln.

Aufbau

Das Buch beginnt mit den Grundlagen des verwendeten Betriebssystems, in unserem Fall ist das eine Linux-Distribution (*Ubuntu 16.04 LTS*). Nachdem die Grundlagen des Systems geklärt sind, geht es weiter mit der Installation und den notwendigen Einstellungen.

Danach folgen in den weiteren Kapiteln die unterschiedlichen Anwendungsgebiete eines Servers, wie E-Mail, Webserver und andere Dienste. Die Kapitel gegen Ende des Buches befassen sich mit allgemeinen und wichtigen Themengebieten aus dem Umfeld, wie dem Backup oder der Sicherheit.

Die Kapitel in diesem Buch können relativ unabhängig voneinander gelesen

werden, eine Lektüre Stück für Stück ist nicht notwendig.

Feedback

Der große Vorteil an einem eBook wie diesem ist, dass es im Gegensatz zu einem gedruckten Buch nicht statisch fest steht und Sie als Leser in den Genuss kostenloser Aktualisierungen kommen können. Dafür ist Ihr Feedback natürlich wichtig.

Wenn Sie finden, dass etwas zu einem Themenbereich fehlt bzw. nicht genügend abgedeckt wurde, Sie Fehler finden oder Kritik äußern möchten, so scheuen Sie sich nicht, mir eine E-Mail zu schreiben. Sie erreichen mich dabei unter *kontakt@seeseekey.net* oder über das Kontaktformular auf seeseekey.net.

Danksagung

Besonders bedanken möchte ich mich bei meinen Lektoren und ersten Lesern Dirk Pohlmann und Martin Harndt.

Des weiteren möchte ich mich bei meiner Frau Anne bedanken, welche mir immer tatkräftig zur Seite stand und mit ein Grund dafür ist, dass Sie dieses Buch in der Hand halten.

Florian Bottke

Beschaffung

Bevor man mit seinen Arbeiten am eigenen Server beginnt, benötigt man einen Anbieter und einen Server. Hier sollte man nicht den erstbesten nehmen, der einen über den Weg läuft, denn bei der Wahl des Anbieters und des Servers gibt es einige Dinge zu beachten.

Der passende Anbieter

Bevor man seinen eigenen Server einrichtet, benötigt man natürlich erst einmal einen solchen. Dabei gibt es zwei Modelle. Das erste ist das Mieten eines Servers für einen bestimmten Betrag pro Monat. Das andere Modell ist unter dem Namen *Colocation* bekannt. Hierbei schafft man die Hardware (also den Server) selber an und bezahlt den Anbieter dafür, den Server mit der entsprechenden Infrastruktur - Strom und Internetanbindung - zu versorgen.

Wer jetzt auf die Idee kommt, einen solchen Server dann doch lieber bei sich zu Hause zu betrieben, sollte es lieber lassen. Im Gegensatz zu einem Rechenzentrum ist es zu Hause schwierig, eine hohe Verfügbarkeit zu erreichen. So kann der Strom ausfallen oder der Netzanbieter hat Probleme mit der Anbindung, so dass der Server plötzlich nicht mehr zu erreichen ist. Außerdem ist die Anbindung für Hausanschlüsse in den meisten Fällen auch wesentlich schlechter als in einem Rechenzentrum. Dort sind die Internetanbindungen auch redundant, das bedeutet, dass ein Rechenzentrum mit mehr als einem Internet Service Provider verbunden ist.

Bei der Wahl zwischen dem Mieten eines Servers und der Colocation muss man auf die eigenen Anforderungen achten. Wenn man keine Spezialhardware, wie z. B. Teslakarten (eine Grafikkarte mit einer großen Anzahl von Shadereinheiten) für die Berechnung benötigt, kann man ruhigen Gewissens auf einen gemieteten Server setzen.

Bei der Suche nach einem entsprechenden Anbieter empfiehlt sich die Webseite webhostlist.de. Dort findet man eine große Auswahl an Anbietern mit den entsprechenden Bewertungen. Meine Empfehlung für den ersten Server wäre die *Hetzner Online GmbH*, welche unter hetzner.de zu finden ist. Dort bekommt sinnvoll konfigurierte Server für einen günstigen Preis. Ein weiterer empfehlenswerter Anbieter ist *Manitu* (zu finden unter manitu.de).

Der richtige Server

Nachdem der passende Anbieter ausgewählt wurde, muss ein passender Server aus dem Angebot gewählt werden. Je nach Anbieter findet man Server in den unterschiedlichsten Ausbaustufen. Grob kann man die angebotenen Serverprodukte in

zwei Kategorien einteilen. Zum einen gibt es die sogenannten virtuellen Server, zum anderen dedizierte Server.

Auf den ersten Blick lässt sich kein Unterschied erkennen, da man auf beiden Servern die volle Kontrolle über sein jeweiliges Betriebssystem hat. Man kann ein Betriebssystem seiner Wahl installieren und in diesem Schalten und Walten wie man möchte.

Während man bei einem dedizierten Server diesen wirklich sein eigen nennt und ihn mit niemandem teilen muss, sieht dies bei einem virtuellen Server anders aus. Hier setzt man seinen Server nicht direkt auf der Hardware auf, sondern in einer sogenannten virtuellen Maschine. Das bedeutet, dass man sich die Leistung des Server mit anderen Kunden teilt. Dafür sind solche Server wesentlich günstiger als dedizierte Server.

Hier muss man abwägen, ob ein virtueller Server für die eigenen Zwecke genügt. Möchte man nur einen Mail- oder XMPP-Server betreiben, reicht dieser in den meisten Fällen.

Anders sieht es aus, wenn man eine Webseite mit vielen Zugriffen oder einen Gameserver betreiben möchte - hier macht sich die Leistung eines dedizierten Servers bemerkbar.

Nachdem man sich entschieden hat, ob es ein virtueller oder dedizierter Server sein soll, geht es daran, das Angebot mit der passenden Leistungsfähigkeit zu finden. So beginnen günstige virtuelle Server schon ab ein paar Euro im Monat, während diese dedizierte Server bei knapp 50 Euro pro Monat beginnen.

Nach oben ist hierbei meist — wie bei so vielen Dingen im Leben — jede Grenzen offen. So ist es ohne weitere Probleme möglich, einen Server für 1400 Euro im Monat zu mieten, aber wer von uns benötigt wirklich 512 GB RAM und 24 TB an Festplattenspeicher?

Deshalb gilt es, bei der Wahl des Servers darauf zu achten, das dieser nicht zu überdimensioniert ist. Als Richtschnur sollte ein Server mit mindestens 16 GB Arbeitsspeicher mit einem Quadcore-Prozessor (und aufwärts) gelten. Damit ist man bei einem dedizierten Server für die meisten Aufgaben gerüstet. Je nach Verwendung, kann man auch kleinere Server wählen. Für einen Mailserver z.B. sollte ein Server mit 2 GB Arbeitsspeicher und einem Dualcore-Prozessor vollkommen ausreichen.

Bei den virtuellen Servern, gibt es keine *echte CPU*, sondern man bekommt sogenannte virtuelle CPUs zugewiesen. Je nach Anforderung können hier ein oder zwei CPU-Kerne genügen.

Bei der Wahl der Festplatte hat man mittlerweile die Wahl zwischen Solid State

Discs (kurz: SSD), welche durch kurze Zugriffszeiten und hohe Transferraten bestechen, und normale Festplatten, welche hohe Kapazität liefern. Wer viel Speicherplatz auf seinem Server benötigt, sollte zu den normalen Festplatten greifen. Hält sich der Speicherbedarf in Grenzen, kann auch bedenkenlos zu einer SSD gegriffen werden.

Bei umfangreichen Serverkonfigurationen sind auch Kombinationen aus einer SSD und gewöhnlichen Festplatten denkbar. In dieser Konfiguration könnte die SSD als Systemplatte dienen, während die Daten auf der Festplatte liegen.

Je nach Anbieter sollte man sich die Konditionen für den Datentransfer ansehen. Ein Server steht in den meisten Fällen nicht nur herum, sondern liefert Daten und empfängt diese. Bei den meisten Anbietern ist ein bestimmtes Datenvolumen inklusive z. B. 10 TB pro Monat - danach muss für jedes weitere Byte bezahlt werden. Andere Anbieter wie *Hetzner* handhaben das anders. Auch hier gibt es ein Inklusivvolumen. Nach dem Erreichen dieses Volumens, wird einfach die Geschwindigkeit der Anbindung (z. B. von 100 MBit/s auf 10 Mbit/s) gedrosselt. Benötigt man wieder eine Anbindung mit höherer Transferrate, muss auch hier wieder gezahlt werden.

Im Normalfall ist es so, dass die Inklusivvolumen in den meisten Servertarifen ausreichen. Selten übersteigt man mit seinem Haus- und Hofserver die in den Tarifen angegebenen Datenvolumen.

Linux-Grundlagen

Im Gegensatz zu einem Betriebssystem, das man auf dem Desktop oder dem Mobiltelefon fährt, kommt ein Serversystem meist ohne eine grafische Benutzeroberfläche aus (die Microsoft Windows Server außen vorgelassen). Das bietet einige Vorteile, so erzeugen die Systeme weniger Last, was der Performance des Servers zu Gute kommt.

Der Einsteiger sollte die Bedienung eines Linuxsystems erst auf einer virtuellen Maschine üben, so dass im Falle eines Falles nichts zu Bruch geht. Hierfür eignet sich die freie Software VirtualBox (virtualbox.org) von Oracle, mit deren Hilfe ein Rechner virtualisiert wird. VirtualBox ist dabei für Mac OS X, Linux und Windows verfügbar. Im Gegensatz zu einem echten Rechner kann man in einer solchen virtuellen Maschine ohne Probleme Schnappschüsse anlegen und zu diesen Punkten jederzeit zurückkehren. Damit kann man sein System auch im Falle eines katastrophalen Fehlers wieder retten.

Ein System ohne grafische Oberfläche hat dabei noch andere Vorteile. So werden die Einstellungen nicht abstrahiert und sind somit für den Administrator des Servers direkt zu erreichen. Im Laufe der Zeit wird man auch feststellen, dass viele Einstellungen, Konfigurationsschritte und Aufgaben auf der Konsole schneller von der Hand gehen und wesentlich unkomplizierter sind.

In diesem Buch wird die Linux-Distribution *Ubuntu 16.04 LTS* genutzt, da diese auch für Einsteiger geeignet ist. Linux selbst bezeichnet dabei nur den Kern (der sogenannte Kernel) des Betriebssystems. Die Zusammenstellung der Kernels mit der entsprechenden Software, wie einer Shell, den Anwendungspaketen und ähnlichem bezeichnet man dabei als Distribution.

Ubuntu ist dabei eine von Debian abgeleitete Distribution und blickt mittlerweile auf eine achtjährige Geschichte zurück. Sie wird von vielen großen Organisationen wie z. B. der MediaWiki Foundation für ihre Server genutzt.

Jedes halbe Jahr — immer im April und im Oktober — gibt es eine neue Ubuntu Version. Die Versionen werden dabei nach dem Schema *Jahr.Monat* versioniert. So trägt die Version vom Oktober 2013 die Versionsnummer 13.10 — die aktuelle LTS-Version vom April 2016 hingegen die Version 16.04.

Für normale Versionen gibt es neun Monate Support, was bedeutet, dass diese Versionen mit Aktualisierung und Sicherheitspatches versorgt werden. In zweijährigen Abständen gibt es dabei eine sogenannte *Long Term Support* Version, welche das Kürzel LTS trägt. Im Gegensatz zu normalen Versionen werden die LTS Versionen fünf Jahre unterstützt.

Das macht sie zu idealen Kandidaten für Server, die lange in Betrieb sein sollen und nicht unbedingt immer die neusten Softwareversionen benötigen. Es ist auch möglich, einen Server mit nicht LTS-Versionen zu betreiben, allerdings muss man in diesem Fall darauf achten, das Betriebssystem regelmäßig auf die neuste Version upzugraden.

Da die normalen Versionen alle sechs Monate erschienen, muss der Server spätestens drei Monate nach dem Erscheinen der neuen Version auf diese aktualisiert werden.

Installation von Software

Mit einem frisch installierten System kann man in der Regel nicht viel anfangen, da nur die notwendigste Software installiert ist. Im Gegensatz zur Windows und Mac OS X Welt verfügt Linux schon seit vielen Jahren über eine ausgeklügelte Paketverwaltung, mit deren Hilfe Software installiert und auch wieder deinstalliert werden kann - im Idealfall bleiben hierbei keine Spuren zurück.

Je nach verwendeter Linux Distribution gibt es dabei unterschiedlichsten Systeme wie *RPM*, *Pacman* oder den *Debian Package Manager*. In unserem Fall wird nur der *Debian Package Manager* behandelt, da dieser auch in Ubuntu Verwendung findet. Für die Verwaltung der Software wird *apt-get* genutzt, welches als Backend den *Debian Package Manager* kurz *dpkg* nutzt. Mit Hilfe von *apt-get* ist es möglich, die Software zu installieren, zu aktualisieren und sie wieder vom System zu entfernen.

Die Software in Linux-Distributionen, ist dabei in sogenannten Paketen organisiert. Im Normalfall gibt es für jede Software ein Paket. Für den *Midnight Commander*, einen grafischen Dateimanager für die Konsole, wäre das z. B. das Paket *mc*. Allerdings gibt es auch Pakete, in denen mehrere Programme zu finden sind, so z.B. bei vielen Utilities.

Eine Anwendung wie der Webserver *Nginx* besteht dabei nicht nur aus einem Paket, sondern auch aus bestimmten Abhängigkeiten, wie dem *OpenSSL*-Paket. Diese Abhängigkeiten werden vom Paketmanager automatisch bei der Installation aufgelöst, so dass nur das Paket *nginx* installiert werden muss — alle benötigen Bibliotheken und Anwendungspakete werden automatisch mit auf dem System installiert.

Für *apt-get* werden dabei root-Rechte benötigt. Ist man nicht als Nutzer *root* angemeldet, so stellt man den folgenden Kommandos den Befehl *sudo* voran. Dieser sorgt dafür, dass das folgende Kommando mit administrativen Rechten ausgeführt wird. Um ein Paket zu installieren gibt man auf der Kommandozeile:

```
apt-get install nginx
```

ein. In diesem Fall wird dann das Paket *nginx* installiert. Es ist auch möglich mehrere Pakete gleichzeitig zu installieren. Dazu gibt man die entsprechenden Pakete

einfach nacheinander ein:

```
apt-get install nginx sendmail mc
```

Mit dieser Zeile werden die Pakete *nginx*, *sendmail* und *mc* installiert. Möchte man ein bestimmtes Paket installieren, kennt aber nicht den genauen Namen so hilft die Ubuntu-eigene Paketsuche, welche unter packages.ubuntu.com zu finden ist. Dort kann man nicht nur für die aktuelle Ubuntu-Version nach Paketen suchen, sondern auch ältere Versionen einbeziehen.

Zum Entfernen eines Paketes bietet *apt-get* die Option *remove*. So wird das Paket *mc* mittels folgendem Befehl wieder deinstalliert:

```
apt-get remove mc
```

Auch bei dieser Option ist es möglich, mehrere Pakete auf einmal zu deinstallieren, indem man diese hintereinander angibt. Bei der Option *remove* muss man beachten, dass damit nur das entsprechende Paket entfernt wird. Daten wie Konfigurationsdateien bleiben in diesem Fall erhalten. Möchte man die entsprechenden Konfigurationsdateien auch entfernen, bietet sich die Option *purge* an:

```
apt-get purge mc
```

Purge entfernt dabei wie *remove* das entsprechende Paket und gleichzeitig die Konfigurationsdateien. Ein wichtiger Punkt bei einem Serverbetriebssystem ist natürlich die Aktualität der Softwarepakete. So ist es sehr problematisch, einen Server mit veralteten Versionen zu betreiben, welche bekannte Sicherheitslücken erhalten. Einem Angreifer wird es bei einem solchen Server unnötig einfach gemacht, den Server zu übernehmen.

Auch für das Aktualisieren der Pakete enthält *apt-get* die passenden Optionen bereit. Mit dem Kommando:

```
apt-get update
```

werden die Dateien mit den Paketquellen auf den aktuellsten Stand gebracht. In diesen Dateien sind die Informationen über alle verfügbaren Pakete enthalten, so das man diese Datei von Zeit zu Zeit aktualisieren sollte. Mit einem:

```
apt-get upgrade
```

wird das System dann auf den aktuellen Stand gebracht, wobei installierte Pakete wenn möglich auf eine neue Version gebracht werden. Eine Stufe weiter geht das Kommando:

```
apt-get dist-upgrade
```

bei dem wie bei *update* installierte Pakete aktualisiert werden, aber gleichzeitig die Möglichkeit besteht, neue Pakete während des Vorgangs zu installieren und bestehende zu deinstallieren.

Neben den Optionen *install*, *remove*, *update*, *upgrade* und *dist-upgrade* enthält *apt-get* noch eine Reihe von weiteren Optionen.

Eine dieser Optionen sind die *clean* und *autoremove*-Direktiven. Mit der Zeit wird der *apt-get*-Cache eine Reihe von Paketen enthalten, welche nicht mehr vom System genutzt werden. Für diese nicht mehr genutzten Pakete und Abhängigkeiten sind die *apt-get*-Optionen *autoremove*, *clean* und *autoclean* gedacht.

Verbindung mit dem Server aufnehmen

Wenn ein Server in einem Rechenzentrum steht, muss man diesen natürlich warten können. Schließlich will die Software aktualisiert und die Konfiguration vorgenommen werden. Hier wäre es äußert unpraktisch den weiten Weg zum Rechenzentrum auf sich zu nehmen. Außerdem würde man bei einem gemieteten Server nicht einmal in dessen Nähe kommen; schließlich sind Rechenzentren sensible Sicherheitsbereiche, in welchen nur wenigen Menschen der Zutritt gewährt wird.

Ein Ausweg aus diesem Dilemma ist die Fernwartung per SSH. SSH steht dabei für *Secure Shell*. Gemeint ist damit, dass man sich die Ausgabe des Servers auf den lokalen Rechner holt und die entsprechenden lokalen Eingaben wieder beim entfernten Server landen. Im Gegensatz zu den meisten anderen Fernwartungsverfahren bietet SSH eine hohe Sicherheit bei der Authentifizierung und Autorisierung. Die Verbindung mit dem Server ist durchgängig verschlüsselt, so dass keine sensiblen Daten nach außen dringen. Damit der Server per SSH erreichbar ist, muss ein SSH-Server installiert werden. Im Falle von Ubuntu ist dies der OpenSSH-Server welcher mittels

```
apt-get install openssh-server
```

installiert werden kann. Im Falle eines gemieteten Servers, ist der OpenSSH-Server in fast 100 Prozent der Fälle vorinstalliert. Der Server horcht standardmäßig auf dem Port 22. Er benötigt nach der Installation keine weitere Konfiguration. Um sich nun auf dem Server anzumelden wird ein SSH-Client benötigt. Unter Linux und Mac OS X ist dieser über das Terminal zu erreichen. Unter Windows benötigt man einen extra Client, es empfiehlt sich hierbei die Software *PuTTY*. Das Kommando zum Verbindungsaufbau sieht dabei wie folgt aus:

```
ssh root@server.example.org
```

Damit versucht der SSH-Client mit dem Rechner hinter der Subdomain *server.example.org* unter dem Namen *root* zu verbinden. Es gibt dabei zwei Möglichkeiten, sich per SSH anzumelden.

Die erste Möglichkeit, ist es sich mit einem Passwort anzumelden. Diese Methode wird als die unsichere von beiden betrachtet. Die zweite Variante ist es sich mit Hilfe eines Zertifikates anzumelden. Der Grund dafür ist unter anderen, das ein Passwort natürlich durchprobiert werden kann. Entscheidet man sich für diese Variante, sollte man ein sicheres und möglichst langes Passwort wählen. In unserem obigen Beispiel würde der entfernte Rechner nach dem Verbindungsversuch nach dem entsprechenden Passwort fragen.

Die zweite Variante ist die Anmeldung mittels eines öffentlichen Schlüssels (Public-Key-Verfahren). Für die Authentifizierung wird hierbei asymmetrische Verschlüsselung genutzt. Dazu wird ein privater und ein öffentlicher Schlüssel erzeugt. Der private Schlüssel verbleibt auf dem Clientrechner, während der öffentliche Schlüssel an den Server übertragen wird. Bei der Anmeldung wird nun mit Hilfe dieser beiden Schlüssel eine sichere Authentifizierung gewährleistet.

Damit dies funktioniert, muss im ersten Schritt ein solcher Schlüssel erstellt werden. Dies geschieht auf einem Linux oder Mac OS X Client mittels:

```
ssh-keygen -t rsa -C "seeseekey@example.org"
```

Unter Windows ist dies nicht ohne weiteres möglich. Hier wird das Tool *PuTTYgen* benötigt, welches ebenfalls auf der *PuTTY* Downloadseite zu finden ist. Ist das Schlüsselpaar erzeugt, muss es im nächsten Schritt auf den Server übertragen werden. Dazu gibt es unter Linux den Befehl *ssh-copy-id*. Um denn öffentlichen Schlüssel auf den Server zu übertragen, könnte der Befehl wie folgt angewendet werden:

```
ssh-copy-id -i ~/.ssh/id_rsa.pub root@example.org
```

Dadurch wird der öffentliche Schlüssel des Clients auf dem Server für den Nutzer *root* hinterlegt. Möchte man den *ssh-copy-id* unter Mac OS X nutzen, so muss dieser vorher nachgerüstet werden. Dazu sind einfach folgende Zeilen im Terminal nacheinander auszuführen:

```
curl -O
https://raw.githubusercontent.com/seeseekey/Selfhosting/master/Scripts/ss
copy-id.sh

sudo cp ssh-copy-id.sh /usr/bin/ssh-copy-id

sudo chmod a+x /usr/bin/ssh-copy-id
```

Anschließend kann der Befehl wie sein Linuxpendant genutzt werden. Bei der Übertragung des öffentlichen Schlüssels auf den entfernten Rechner wird man nach dem Passwort des ausgewählten Nutzers (in unserem Beispiel *root*) gefragt. Dies dient der Bestätigung - — damit weiß der Server, dass man autorisiert ist, einen Schlüssel für den entsprechenden Nutzer zu hinterlegen.

Wurde der Schlüssel hinterlegt, kann man sich wieder per SSH am entfernten Rechner anmelden. Wenn alles funktioniert, benötigt man kein Passwort für die Anmeldung — was die Wartung eines Servers nicht nur sicherer, sondern auch bequemer macht.

Das Kommando-ABC

Bei dem Betrieb eines Server mit einem Linuxsystem wird man immer wieder mit bestimmten Aufgaben und Problemen konfrontiert werden. Auf diesen Seiten sollen dabei die entsprechenden Befehle und Anwendungen in Kurzform vorgestellt werden, so dass man mit einem Grundstock an Befehlen und Anwendungen vertraut ist.

Change Directory (cd)

Alten DOS Veteranen sollte dieser Befehl bekannt vorkommen. Mittels *cd* ist es möglich in ein anderes Verzeichnis zu wechseln. Neben der normalen Anwendung

```
cd example
```

mit welchem in den Unterordner mit dem Namen *example* gewechselt wird, gibt es auch einige Spezialfälle. Mittels

```
cd ..
```

wechselt man im Verzeichnisbaum eine Ebene nach oben. Sehr praktisch ist es auch *cd* ohne Parameter anzugeben. Wenn dies geschieht, wechselt man in das Heimverzeichnis des angemeldeten Nutzers, im Falle des Nutzers *root* wäre dies */root/*.

Catalog (cat)

Das Kommando *cat* steht für Catalog und gibt den Inhalt von Dateien aus. Dies ist immer dann praktisch, wenn man ohne Umwege z. B. eine Konfigurationsdatei ausgeben möchte.

Copy (cp)

Der Befehl *cp* bezeichnet unter Linux die Kurzform des englischen Wortes *copy*. Mit diesem Befehl können Dateien von A nach B kopiert werden. Die einfachste Anwendung sieht dabei wie folgt aus:

```
cp dokument.txt dokument.txt.bak
```

Hierbei wird die Datei *dokument.txt* kopiert. Nach der Ausführung des Kommandos befindet sich im gleichen Ordner eine Datei mit dem Namen *dokument.txt.bak*, welche selbstverständlich den gleichen Inhalt wie die Originaldatei enthält.

grep

Mittels *grep* können bestimmte Zeichenketten gesucht werden. Besonders sinnvoll ist dieser Befehl in Verbindung mit Pipes, welche später behandelt werden. Um z. B. eine Datei nach dem Stichwort *options* zu durchsuchen könnte folgender Ausdruck genutzt werden:

```
cat file.txt | grep options
```

Hierbei werden sämtliche Vorkommnisse des Wortes *options* in der *file.txt* Datei auf dem Terminal ausgegeben. Mittels der Option *-r* kann *grep* Verzeichnisse rekursiv durchsuchen. Wenn wir uns im Verzeichnis */var/log/* befinden und dort den Befehl:

```
grep -r mail
```

eingeben, werden alle Vorkommnisse des Wortes *mail*, in allen Dateien welche sich unter */var/log/* befinden, angezeigt.

htop

In vielen Fällen möchte man wissen, was im Moment auf dem Server passiert. Hierbei hilft das Tool *htop*. Wenn dieses installiert ist, kann es mittels Eingabe seines Namens aufgerufen werden. *htop* zeigt anschließend viele Informationen über das System an. Zu diesen Informationen gehören unter anderem eine vollständige Liste der ausgeführten Prozesse, die Auslastung der CPU und des Speichers.

Manpage (man)

Mit dem *man* Befehl kann die sogenannte *Manpage* eines Kommandos aufgerufen werden. So würde:

```
man ls
```

die Hilfeseite des Befehls *ls* öffnen. Verlassen werden kann die *Manpage* in dem die Taste *q* gedrückt wird.

Midnight Commander (mc)

Beim Midnight Commander handelt es sich um einen Norton Commander Clone, den viele Anwender sicher noch aus MS-DOS Zeiten kennen werden. Der Commander ist dabei ein grafischer Dateimanager mit einer zweigeteilten Ansicht, so dass man Dateien von A nach B kopieren bzw. andere Operationen durchführen kann.

Move (mv)

Auch beim Befehl *mv* handelt es sich um ein Kürzel für ein englischsprachiges Wort, in diesem Fall *move*. Die Bedienung ist ähnlich dem *cp*-Befehl mit dem

Unterschied, das keine Kopie angelegt wird, sondern die Datei direkt verschoben wird.

Nano (nano)

Nano ist ein mitgelieferter Texteditor, der im Gegensatz zu vi (in vielen Linux-Distributionen der Standardeditor) auch für Einsteiger leicht zu bedienen ist. Eine Datei wird geöffnet (oder angelegt) indem der Dateiname als Parameter übergeben wird:

```
nano test.txt
```

Ist die Bearbeitung abgeschlossen kann die Datei mittels *Strg+O* gespeichert werden und der Editor mit *Strg+X* verlassen werden.

List (ls)

Mittels des Befehles *ls* kann der Verzeichnisinhalt angezeigt werden. *ls* unterstützt dabei eine Reihe von Optionen.

Röhren verbinden

Die Philosophie von Unix und auch Linux besteht darin, dass es für jedes Kommando eine Aufgabe gibt und diese von dem Kommando beherrscht wird (*Do One Thing and Do It Well*). Wenn man nun mehrere Kommandos miteinander kombinieren möchte, wird es auf den ersten Blick schwierig.

Wie leitet man die Ausgabe von Prozess A zu Prozess B? Die Antwort auf dieses Problem sind Pipes. Diese *Röhren* dienen dazu, Daten von einem Prozess zu einem anderen zu übermitteln. Ein Beispiel:

```
cat server.conf | grep port
```

Die Befehle in diesem Beispiel werden von links nach rechts ausgegeben. Das *cat* Kommando gibt zuerst die Datei *server.conf* aus. Allerdings geschieht die Ausgabe nicht wie sonst auf der Konsole, sondern wird mittels der Pipe, gekennzeichnet durch das | Symbol, umgeleitet. Der Empfänger der Ausgabe folgt nach der Pipe, in diesem Fall das Kommando *grep*, welches nach der Zeichenkette *port* sucht.

Im Endeffekt wird mit diesem Kommando also nach dem Begriff *port* in der Datei *server.conf* gesucht. Man ist hier nicht auf eine Pipe beschränkt, sondern kann diese beliebig anhängen.

Regelmäßige Aktionen

Auf vielen Servern — wenn nicht auf allen — gibt es bestimmte Aktionen, welche immer und immer wieder ausgeführt werden sollen. Für dieses Problem gibt es den

sogenannten Cron-Daemon. Dieser prüft in regelmäßigen Abständen den sogenannten *Crontab*. Dabei handelt es sich um eine Tabelle mit auszuführenden Skripten oder Anwendungen, welche mit einer Ausführungszeit oder einem Ausführungsintervall zusammen gespeichert werden.

Möchte man diese Tabelle bearbeiten, so gibt man auf dem Terminal

```
crontab -e
```

ein. Damit öffnet man die Crontab-Tabelle des aktuellen Nutzers. Eine Tabelle könnte dabei so aussehen:

```
# m h dom mon dow command

*/20 mono timelapse.exe
```

Im konkreten Beispiel wird hier die Mono-Anwendung *timelapse.exe* alle 20 Minuten gestartet. Jede Reihe ist dabei immer aus folgenden Spalten aufgebaut:

```
Minute Stunde Tag des Monats Monat Wochentag Bedeutung
```

Füllt man eine Spalte mit einem Sternchen, so wird der betreffende Wert ignoriert. Wichtig ist, dass eine Crontab-Datei immer mit einer Leerzeile abgeschlossen wird. Vergisst man die Leerzeile, kann dies zu Problemen führen.

Logführung

Einige der Vorteile an einem Server mit Linux entdeckt man erst dann, wenn es Probleme macht. Bei einem fehlerhaften System, will man natürlich die Ursache der Störungen finden. Hierzu gibt es unter Linux eine Reihe von Logdateien, welche die Suche beschleunigen und eingrenzen.

Grundsätzlich befinden sie die Logdateien unter Ubuntu im Verzeichnis */var/log/*. Neben den direkt in diesem Verzeichnis liegenden Logdateien, gibt es dort auch noch einige Unterorder, welche unter anderem von installierten Anwendungen stammen (z.B. Nginx). Einige dieser Dateien sollen hier vorgestellt werden.

Die Logdatei *auth.log* dokumentiert jegliche Versuche eines Logins. Dabei ist es unerheblich ob der Loginversuch erfolgreich war oder fehlgeschlagen ist. Systeme wie *Fail2ban* werten diese Datei aus um bestimmte IP-Adressen zu sperren, welche zu oft versuchen sich mit falschen Nutzerdaten auf dem Server einzuloggen.

Der Debian Package Manager führt mit der *dpkg.log*-Datei ebenfalls Buch über seine Aktionen. In dem Log finden sich detaillierte Aufstellungen der letzten Aktion des Paketmanagers.

In der *kern.log* sind Meldungen des Kernel zu finden. Es kann sich dabei um

Meldungen vom RAID System bis zu SSH und Meldungen des Dateisystemes handeln.

Ist auf dem System ein Mailserver installiert, so wird man im */var/log/* Verzeichnis die Dateien *mail.log* und *mail.err* finden. In der *.log* Dateien findet man allgemeine Meldungen und Warnungen des MTA, sowie des IMAP/POP3-Server.

Betreibt man auf seinem Server den SMB-Server Samba, so findet sich die gleichlautende Logdatei *samba* ebenfalls im */var/log/* Verzeichnis.

In vielen Linux-Systemen gibt es einen sogenannten Syslog-Dienst, welcher anderen Diensten eine Schnittstelle liefert, über welche Sie Meldungen an das Systemlog übergeben können. Dieses Log findet sich in der Datei *syslog* im */var/log/* Verzeichnis wieder.

Geschichte

In diesem Kapitel soll der Nutzer in die Geschichte von Linux eingeführt werden. Mit der Entwicklung von Linux begann dabei im Jahre 1992 der Finne Linus Torvalds, welcher einen Terminalemulator schreiben wollte, und die Architektur seines 386er PCs kennenlernen wollte.

Ursprünglich war Linux (damals noch unter dem Namen Freax) nicht dafür gedacht, auf andere Rechnerarchitekturen portiert zu werden. Dies war dadurch bedingt, das Torvalds einige sehr architekturspezifische Dinge benutze. Mittlerweile läuft Linux aber nicht nur auf PCs (x86) sondern auch auf so unterschiedlichen Geräten wie Waschmaschinen und Kühlschränken.

Damit gehört es wahrscheinlich zu dem Betriebssystem, das auf den meisten CPU- und Rechnerarchitekturen dieser Welt läuft. Bei Linux handelt es sich bei genauer Betrachtung um einen Unix-Klon. Linux gehört damit wie BSD zu den unixartigen oder auch unixoiden Systemen.

Die Anfänge von Linux liegen in den frühen 90er Jahren des letzten Jahrhunderts. Am 25. August veröffentlichte der damals 21-jährige Linus Torvalds einen Post im Usenet, in welchem er Linux ankündigt. Die erste öffentliche Version war knapp einen Monat später, am 17. September 1991, auf einem FTP-Server zu finden. Dabei wurde das Verzeichnis auf diesem FTP-Server vom Administrator des Servers von Freax auf Linux geändert, womit Linux seinen Namen bekam.

Da Linux genau betrachtet nur der Kernel ist und man mit einem solchem noch nicht wirklich viel anfangen konnte, wurden grundlegende Werkzeuge wie ein Editor, ein Compiler und andere benötigt. Hier spielte ein anderes Projekt eine große Rolle, ohne das Linux zum damaligen Zeitpunkt noch weit von einem halbwegs benutzbaren System entfernt gewesen wäre.

Die Rede ist vom GNU-Projekt. Dieses im Jahre 1983 von Richard Stallmann initiierte Projekt hatte das Ziel ein freies Betriebssystem zu schaffen. Allerdings ging Richard Stallmann genau entgegengesetzt an dieses Problem heran. Er fing an, die Anwendungen und Werkzeuge zu schreiben, welche für ein solches System nötig sind. Dies führte in den Jahren 1991 und 1992 dazu, das Richard Stallmann die passenden Werkzeuge und Linus Torwalds den Kernel lieferte.

Aus diesem Grund ist Richard Stallmann bis zum heutigen Tage der Meinung das Linux nicht Linux sondern GNU/Linux heißen sollte. Bei der freien Distribution Debian ist dies auch der Fall, dort wird Linux im Regelfall nur GNU/Linux genannt.

Richard Stallmann veröffentlichte während seiner Arbeit am GNU-Projekt auch eine Softwarelizenz, die sogenannte GNU Public Licence, kurz GPL. Unter diese Lizenz stelle Torwalds im Jahre 1992 seinen damaligen Kernel. Diese Lizenz führt dazu, dass der Kernel sogenannte freie Software ist. Freie Software garantiert dabei einige Freiheiten. Diese werden von der Free Software Foundation wie folgt definiert:

Freiheit 0: Das Programm zu jedem Zweck auszuführen.

Freiheit 1: Das Programm zu untersuchen und zu verändern.

Freiheit 2: Das Programm zu verbreiten.

Freiheit 3: Das Programm zu verbessern und diese Verbesserungen zu verbreiten, um damit einen Nutzen für die Gemeinschaft zu erzeugen.

Bereits im Jahre 1993 arbeiteten über hundert Entwickler am damals neuen Linuxkernel. Ein Jahr später wurde die Version 1.0 des Kernels veröffentlicht. Diese Version war die erste netzwerkfähige Version des Linuxkernels. 1995 erschienen die ersten Portierungen des Linux-Kernels auf andere Prozessorarchitekturen (unter anderem DEC und Sun SPARC). Wiederum ein Jahr später erblickte die Version 2.0 des Kernels das Licht der Welt.

Zu dieser Zeit wurde Linux zunehmend ernster genommen. So erschienen 1997 bekanntere propietäre Applikationen wie der Netscape Navigator und StarOffice für Linux. 1998 gaben Hersteller wie IBM und Oracle ihre Linuxunterstüzung bekannt. Ein Jahr vor Ende des Jahrtausends, im Jahr 1999, erschien ein Meilenstein in der Kernelentwicklung: die Version 2.2. In dieser gab es große Verbesserungen am Netzwerkcode sowie der Unterstützung von mehr als einer CPU pro Maschine.

Im neuen Millennium gab es für Linux nur noch eine Richtung: Nach oben. Mittlerweile nutzen wir Linux nicht nur in unsichtbaren Bereichen wie unserer Waschmaschine oder in großen Rechenzentren, sondern auch auf unseren Mobiltelefonen in Form von Android.

sudo oder nicht sudo

Auf Ubuntu basierenden Distribution, zu denen auch Ubuntu Server gehört, ist es üblich, dass man keinen direkten Zugriff auf den *root*-Nutzer hat. Der *root*-Nutzer entspricht dem Systemverwalter, der auf dem System tun und lassen kann, was er will.

In vielen anderen Linux-Distributionen ist man nach der Installation als *root*-Nutzer angemeldet. Bei Ubuntu hingegen erstellt man sich einen normalen Nutzer, welcher über die Berechtigung verfügt, Kommandos auch mit *root*-Rechten auszuführen.

Damit der entsprechende Nutzer dies auch in der Praxis tun kann, gibt es das *sudo* Kommando. Wenn ein normaler Benutzer das Kommando *reboot* ausführt wird er folgende Meldung bekommen:

```
reboot: Administratorrechte (»root«-Rechte) benötigt
```

Mit Hilfe des *sudo*-Kommandos würde die Befehlszeile wie folgt aussehen:

```
sudo reboot
```

Bestätigt der Nutzer dieses Kommandos wird er nach seinem Nutzerpasswort gefragt. Anschließend wird der Befehl mit administrativen Rechten ausgeführt. Damit bietet *sudo* auch für den Nutzer eine erhöhte Sicherheit, da bestimmte Befehle, welche unter Umständen eine zerstörerische Wirkung auf das System haben, nicht ohne ein entsprechendes *sudo* ausgeführt werden können.

In diesem Buch ist die Verwendung von *sudo* nicht nötig, da wir während der Einrichtung des Systems den *root*-Account unter Ubuntu aktivieren werden. Während die Einschränkung auf Desktopsystemen sehr sinnvoll ist, kann man sich bei einer Serverinstallation darüber streiten.

Grundeinrichtung

Der Server ist beschafft und die Linux-Grundlagen wurden verinnerlicht. Eventuell wurde das eine oder andere Buch aus dem Kapitel *Weiterführendes* angeschaut. Nun muss der Server mit einem Betriebssystem versehen und eingerichtet werden. Dieses Kapitel fängt dabei mit der Installation an und geht dann Stück für Stück auf die grundlegenden Themen ein.

Installation

Nachdem der entsprechende Server bestellt ist und man die Zugangsdaten (meist per E-Mail) vom Anbieter erhalten hat, kann die Installation des passenden Betriebssystems beginnen. In diesem Buch wird dabei die Linux-Distribution *Ubuntu-Server 16.04 LTS (Long Term Support)* behandelt.

Bei den meisten Betreibern von dezidierten Servern, kann man eine große Auswahl an Betriebssystemen als Image auf den entsprechenden Rechner installieren lassen. In einem Großteil der Fälle handelt es sich dabei um eine Reihe von Linux-Distributionen. Daneben gibt es manchmal auch noch die Systeme *FreeBSD* und *NetBSD*. Auf vielen Servern ist es auch möglich, Windows Server als Betriebssystem auszuwählen. Dies ist allerdings meist mit zusätzlichen Lizenzgebühren verbunden und wird im Rahmen dieses Buches nicht behandelt.

Wenn man das entsprechende Image bzw. Abbild des Betriebssystems über die Verwaltungsoberfläche seines Anbieters aufspielt, so hat man nicht sehr viel mit der Installation zu tun. Meist kann man nur einige grundlegende Einstellungen wie Rechnername, verwendete Dateisysteme und die Aufteilung des Dateisystems bestimmen. Der Rest wird vom vorkonfektionierten Installer erledigt. Nach einigen Minuten erhält man seine Zugangsdaten in Form eines Nutzernamens und eines Passwortes und kann sich über diesen per SSH mit dem Server verbinden.

Anders sieht es aus, wenn man das System (aus welchen Gründen auch immer) selbst auf dem Server installiert, wie es z. B. bei der Colocation der Fall ist. Auch beim Betrieb von virtuellen Maschinen führt man die Installation der einzelnen virtuellen Maschinen selbst durch.

Für die eigene Installation benötigt man ein Image (Abbild) des Betriebssystems. Die aktuelle Ubuntu-Version kann dabei unter ubuntu.com/download/server bezogen werden. Die Version 16.04 ist dabei eine LTS-Version; das bedeutet, dass sie über einen längeren Zeitraum — bei Ubuntu Server 5 Jahre — mit regulären und Sicherheitsaktualisierungen versorgt wird.

Erste Schritte

Wenn der Server frisch installiert ist und man sich das erste Mal per SSH auf seinem Server einloggt, wird man vom Kommando-Prompt begrüßt:

```
nutzer@servername #
```

Er ist dabei nach einem Schema aufgebaut. Beim ersten Login wird der Nutzer der in der Installation angelegte Nutzer sein. Je nach Anbieter kann es sein, dass man direkt als *root*-Nutzer angemeldet ist. Die andere unter Ubuntu gebräuchliche Variante ist, dass man mit dem Nutzer angemeldet ist, welcher bei der Installation angelegt wurde. Dieser Nutzer bekommt administrative Rechte dabei nur über Befehle wie *sudo* oder *su*.

root-Nutzer aktivieren

Unter Ubuntu gibt es in einer Standardinstallation keinen aktivierten *root*-Nutzer. Das bedeutet allerdings nicht, dass der Nutzer nicht vorhanden ist. Stattdessen wird für den *root*-Nutzer kein Passwort vergeben, was dazu führt, dass man ihn nicht nutzen kann. Möchte man ihn doch nutzen, so vergibt man einfach ein Passwort für diesen Nutzer:

```
sudo passwd root
```

Anschließend verfügt man über einen *root*-Nutzer und kann sich mit diesem einloggen. Allerdings muss dem *root*-Nutzer nicht unbedingt ein Passwort zugewiesen werden. Stattdessen ist es auch möglich dem Nutzer ein Zertifikat zuzuweisen, mit welchem sich per SSH eingeloggt werden kann. Damit hat der *root*-Nutzer kein Passwort und kann trotzdem genutzt werden. Um das zu bewerkstelligen muss der öffentliche Schlüssel des SSH-Clients in der entsprechenden Datei des *root*-Nutzers angelegt werden:

```
sudo su
mkdir /root/.ssh/
nano /root/.ssh/authorized_keys_
```

In die Datei *authorizedkeys* wird nun der geschrieben und die Datei gespeichert. Damit kann sich nun per SSH direkt in den *root*-Nutzer eingeloggt werden.

Nun kann der Nutzer, der bei der Installation angelegt wurde, gelöscht werden. Dies geschieht mit dem Kommando *deluser*:

```
deluser nutzername
```

Damit wird der Nutzer aus der */etc/shadow*-Datei entfernt und zusätzlich sein Homeverzeichnis (*/home/nutzername/*) gelöscht. Nun ist es möglich, sich als *root*-Nutzer auf dem Server einzuloggen. Bei einigen Serveranbietern wie z. B. Hetzner sind die Betriebsystem-Images so gestaltet, dass man bereits zu Beginn einen *root*-Nutzer zur Verfügung hat.

Auf einem Serversystem spricht auch nichts dagegen, dass der administrative Nutzer entsprechende Rechte hat. Hier gilt die Prämisse, dass der Besitzer bzw. Betreuer eines Servers wissen sollte, was er tut.

Systeminfo beim Login aktivieren

Wenn man sich auf einem Ubuntu-Server einloggt, bekommt man meist folgende Zeilen zu sehen:

```
Welcome to Ubuntu 16.04 LTS (GNU/Linux 4.4.0-22-generic x86_64)


  • Documentation:  https://help.ubuntu.com/



  System information as of Wed Jun 29 22:23:32 CEST 2016

  System load:  0.18               Processes:           172
  Usage of /:   11.8% of 54.86GB   Users logged in:     0
  Memory usage: 32%                IP address for eth0: 192.168.1.1
  Swap usage:   2%

  Graph this data and manage this system at:
https://landscape.canonical.com/

1 Software-Paket kann aktualisiert werden.
0 Aktualisierungen sind Sicherheitsaktualisierungen.
```

Bei dieser Meldung handelt es sich um Systeminformationen, die beim Login auf dem Server angezeigt werden. Somit erhält der Nutzer einen Überblick über die wichtigsten Informationen, wenn er sich einloggt.

Unter Umständen kann es aber auch passieren das man diese Meldung nicht sieht. Den auszugebenen Text findet man dabei in der Datei */etc/motd*. Dort kann man das Verhalten beim Login allerdings nicht konfigurieren. Um die Meldung zu aktivieren, muss stattdessen die Datei */etc/pam.d/login* mittels eines Editors angepasst werden. Dort findet man die Zeilen:

```
Prints the message of the day upon succesful login.
(Replaces the `MOTD_FILE' option in login.defs)
This includes a dynamically generated part from /run/motd.dynamic
and a static (admin-editable) part from /etc/motd.
session    optional   pam_motd.so  motd=/run/motd.dynamic noupdate
session    optional   pam_motd.so
```

welche je nach Bedarf ein- oder auskommentiert werden können. Sind diese Zeilen bereits einkommentiert, so liegt das Problem am Fehlen des Kommandos *landscape-sysinfo*. Sobald das passende Paket mittels:

```
apt-get install landscape-common
```

nachinstalliert wurde, funktioniert das Anzeigen der Systeminformationen nach einem Login wieder. Je nach Bedarf und Wunsch kann man die Funktion zur Anzeige von Systeminformationen beim Login auch deaktivieren.

Virtuelle Maschinen

In diesem Buch wird eine virtualisierte Serverkonfiguration beschrieben. Das bedeutet, dass wir auf einem Server mehrere sogenannte virtuelle Maschinen betreiben.

Eine virtuelle Maschine ist dabei ein Computer, der nicht direkt auf einer physischen Hardware ausgeführt wird. Stattdessen läuft die virtuelle Maschine auf einem sogenannten Hypervisor. Der Vorteil ist, dass man damit mehrere virtuelle Maschinen auf einem physikalischen Server betreiben kann (wie es unter anderem bei sogenannten V-Servern gemacht wird).

Die Nutzung von virtuellen Maschinen hat dabei einige Vorzüge. So kann man unabhängige Systeme konfigurieren und diese auch auf jeweils anderen IP-Adressen betreiben. Der virtuellen Maschine können bei Bedarf problemlos mehr Ressourcen wie Arbeitsspeicher und CPUs zugewiesen werden (wenn das Hostsystem ein entsprechendes mehr an Ressourcen bereitstellt). Auch ein Serverumzug ist dank virtueller Maschinen einfacher zu bewerkstelligen. So müssen nur die Festplattenimages der entsprechenden Maschinen auf den neuen Server kopiert und in den neuen Hypervisor eingebunden werden.

Natürlich hat die Nutzung virtueller Maschinen auch Nachteile. So entsteht ein gewisser Mehraufwand (Overhead) bei deren Nutzung. Das System läuft langsamer, da eine zusätzliche Abstraktionsschicht zwischen physikalischer Maschine und Betriebssystem eingefügt wird. Allerdings verfügen x86- und x64-kompatible CPUs mittlerweile seit einigen Jahren über Hardwareunterstützung für Virtualisierung, die diese Nachteile stark minimieren.

Aufsetzen eines KVM-Hostes

Um eine Virtualisierung durchzuführen, wird ein sogenannter Hypervisor benötigt. Linux bietet hierfür standardmäßig die *Kernel Virtual Machine* kurz KVM an, mit der eine Virtualisierung schnell eingerichtet ist. Dazu installiert man auf dem Server eine Version von Ubuntu Server (mit dem *OpenSSH*-Paket). Dabei sollte man darauf achten, dass der Nutzer kein verschlüsseltes *home*-Verzeichnis besitzt, sonst könnte es später Probleme mit der Verwendung von SSH-Schlüsseln geben. Anschließend überprüft man auf der Konsole mittels:

```
cat /proc/cpuinfo
```

ob die CPU über die entsprechende Hardwareunterstützung für Virtualisierung verfügt. Diese erkennt man in der Sektion *flags* der Ausgabe. Dort muss für Intel-CPUs das Flag *vmx* und für AMD-CPUs das Flag *svn* vorhanden sein. Ist dies der Fall, so kann KVM mittels:

```
apt-get install qemu-kvm libvirt-bin virtinst
```

installiert werden. Ein anschließendes:

kvm-ok

überprüft dann nochmal, ob die CPU wirklich für KVM geeignet ist. Dabei ist zu beachten, dass es das Kommando *kvm-ok* nur unter Ubuntu gibt, andere Distribution enthalten es aller Wahrscheinlichkeit nach nicht. Nun muss der Nutzer noch der Gruppe *libvirtd* hinzugefügt werden. Auf der Konsole ist dazu ein:

```
adduser root libvirtd
```

nötig. Danach sollte der KVM-Host neu gestartet bzw. sich ab- und wieder angemeldet werden. Zur Verwaltung der Maschinen wird der *Virtual Machine Manager* benutzt. Dabei handelt es sich um eine für Linux verfügbare grafische Oberfläche zur Verwaltung des KVM-Hosts und seiner Maschinen. Dieser wird auf der entsprechenden Zielmaschine (welche nicht identisch mit dem KVM-Host sein muss) mittels:

```
apt-get install virt-manager
```

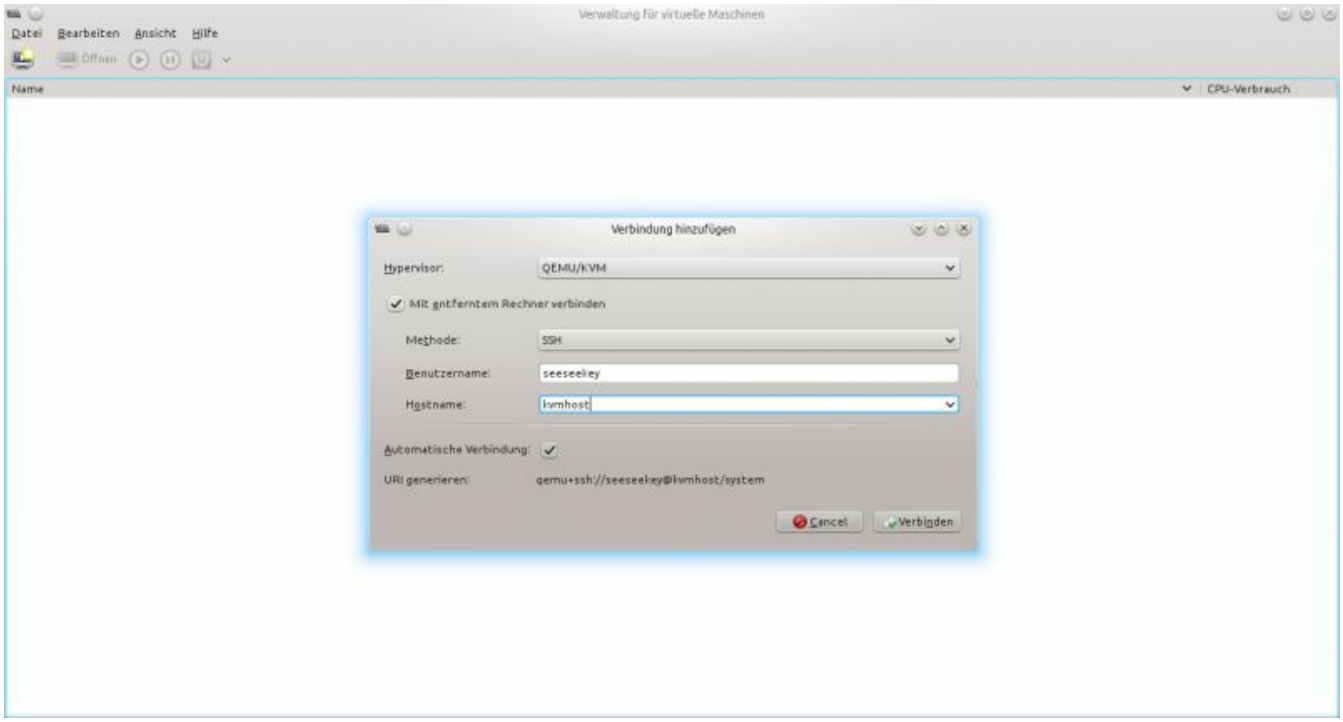
installiert. Auf der Maschine, die die Verwaltung übernimmt, sollte ein SSH-Schlüssel erzeugt werden. Dies geschieht auf der Konsole:

```
ssh-keygen -t rsa -C "user@example.org"
```

Nun übertragen wir den Schlüssel auf den KVM-Host, damit wir uns mit diesem verbinden können, was mittels des Kommandos *ssh-copy-id* bewerkstelligt wird:

```
ssh-copy-id -i /.ssh/idrsa.pub root@kvmhost
```

Danach sollte der *Virtual Machine Manager* gestartet werden. Über *Datei -> Verbindung hinzufügen* wird im darauf folgenden Dialog der KVM Host hinzugefügt.



Eine Verbindung wird hinzugefügt

Nun muss noch eine Netzwerkbrücke eingerichtet werden. Diese dient dazu, dass die virtuellen Maschinen von außen angesprochen werden können. Ohne diese Brücke befinden sich die Maschinen hinter einem NAT (Network Address Translation) und können nur mit dem KVM Host kommunizieren - was natürlich subobtimal ist.

Um die Bridge zu erstellen, wird die Datei */etc/network/interfaces* geändert. Auf einem normalen System sollte diese wie folgt aussehen:

```
This file describes the network interfaces available on your system
and how to activate them. For more information, see interfaces(5).

The loopback network interface
auto lo
iface lo inet loopback

The primary network interface
auto eth0
iface eth0 inet dhcp
```

Nun werden *auto eth0* in *auto br0* und *iface eth0 inet dhcp* in *iface br0 inet dhcp* geändert. Anschließend fehlt nur noch die Zeile:

```
bridge_ports eth0
```

welche am Ende hinzugefügt wird. Damit sieht die neue */etc/network/interfaces* dann wie folgt aus:

```
This file describes the network interfaces available on your system
and how to activate them. For more information, see interfaces(5).
```

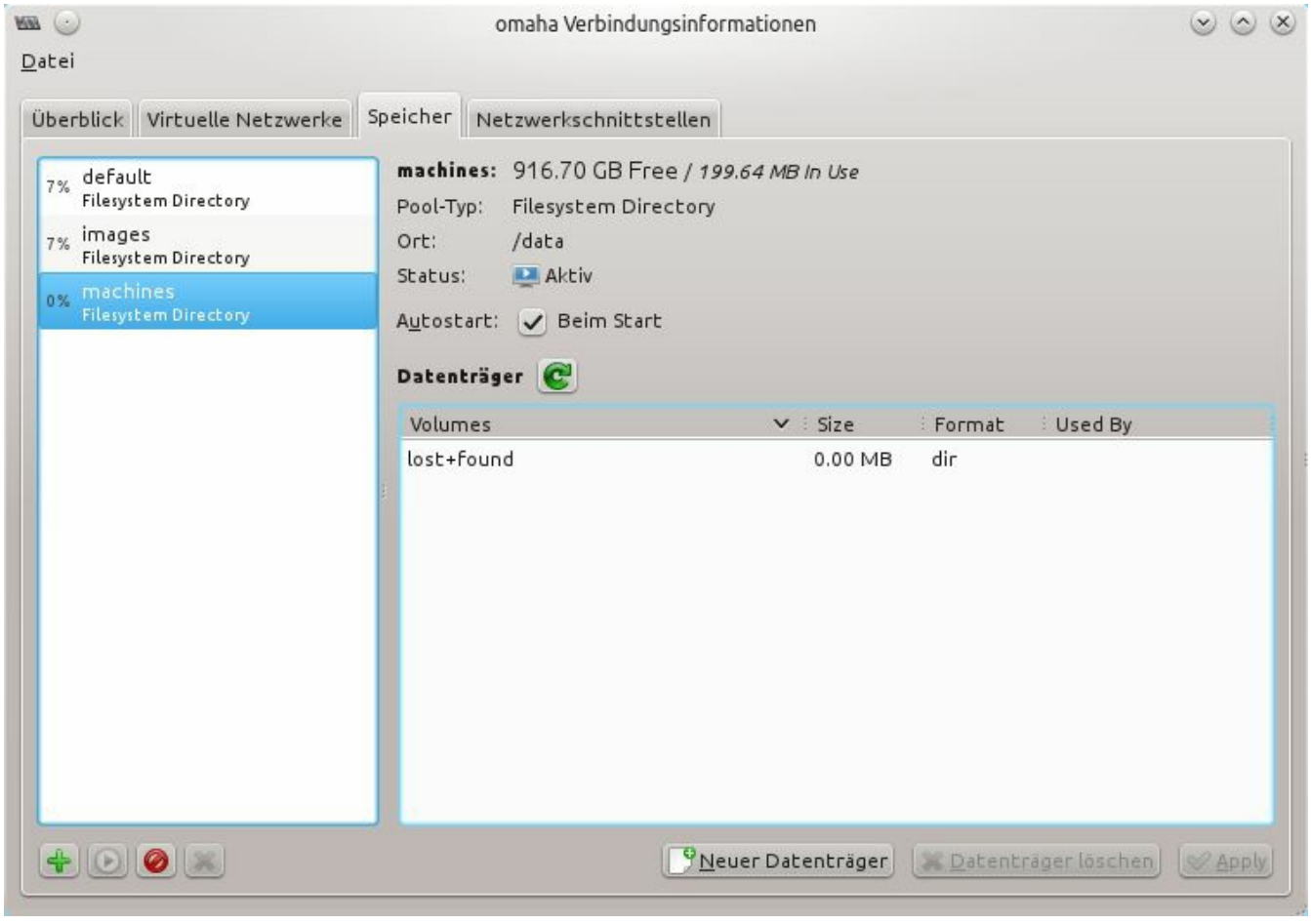
```
The loopback network interface
auto lo
iface lo inet loopback

The primary network interface
auto br0
iface br0 inet dhcp
bridge_ports eth0
```

Um die Änderungen zu übernehmen, muss im Terminal anschließend:

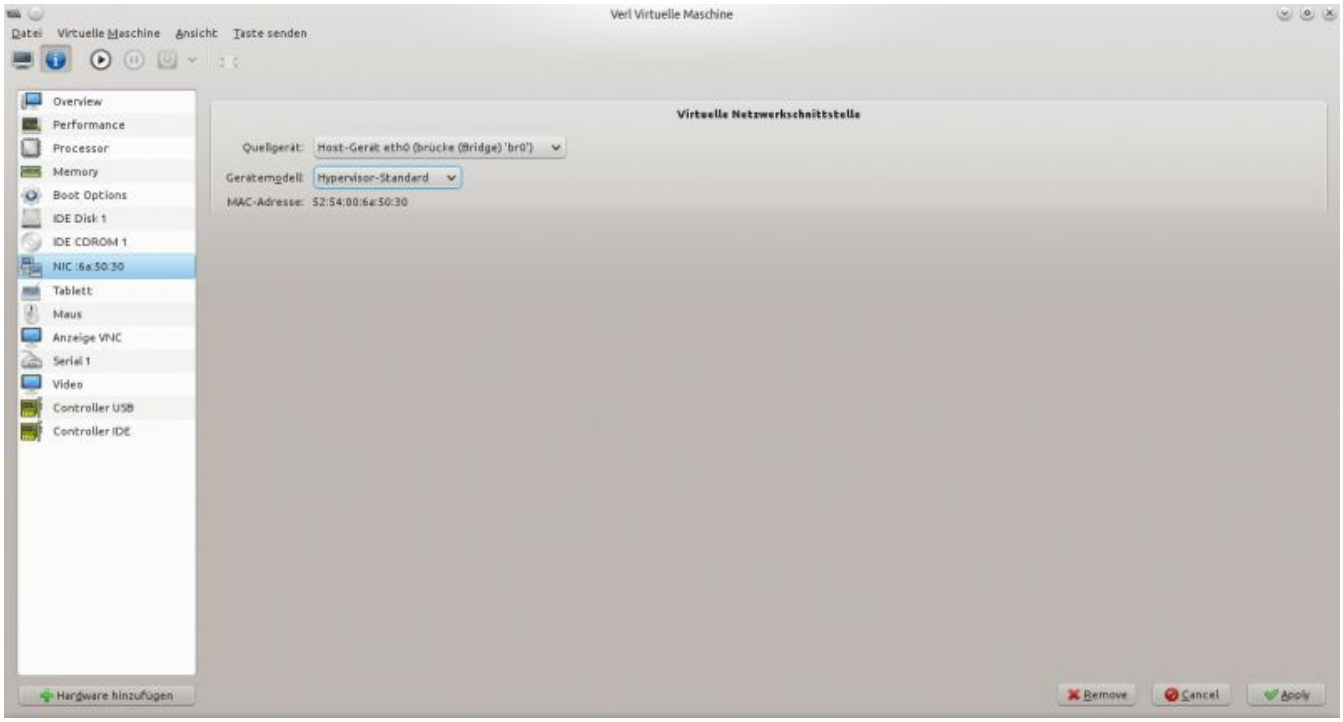
```
/etc/init.d/networking restart
```

mit *root*-Rechten eingegeben werden und schon ist die entsprechende Konfiguration wirksam.



Die Storage Pools des KVM Host

Nachdem dies geschehen ist, kann man eine neue virtuelle Maschine anlegen. Dies geschieht mittels des *Virtual Machine Manager*. Bei den Storage Pools, in welchem die Daten für die virtuellen Maschinen liegen, empfiehlt es sich, den *default*-Pool außen vor zu lassen. Stattdessen legt man sich einen Pool *images* und einen Pool *machines* an. Im *images*-Pool lagert man dann alle Betriebssystemimages für die Installation neuer Maschinen. Im *machines*-Pool hingegen sollten sich die installierten Maschinen befinden.



Das Quellgerät muss auf die Netzwerkbrücke eingestellt werden

In jeder virtuellen Maschine muss dabei das Quellgerät in der Netzwerkkonfiguration auf die Netzwerkbrücke (*br0*) eingestellt werden. Damit ist die Maschine ein Teil des Netzwerkes, in dem sich auch der KVM-Host befindet. Bei den virtuellen Maschinen empfiehlt es sich, in der entsprechenden Konfiguration unter *Video* das Modell *vmvga* auszuwählen.

Nachdem die Maschine konfiguriert wurde, kann sie gestartet werden. Dazu sollte vorher ein Installationsmedium eingelegt werden. Im Falle einer virtuellen Maschine geschieht das z. B. dadurch, dass man eine Imagedatei im virtuellen CD-Laufwerk hinterlässt. Auch die Boot-Reihenfolge sollte kontrolliert werden. Nachdem diese Schritte abgeschlossen sind, kann die Maschine gestartet werden. Nach dem Start sieht man die Ausgabe der virtuellen Maschine und kann die Installation vornehmen.

Name des Servers ändern

Unter Umständen kann es gewünscht sein den Name des Servers zu ändern. Wenn man sich per SSH auf dem Server einloggt wird man von einem Prompt begrüßt.

```
root@rechnername:~#
```

Dieser Prompt enthält dabei unter anderem den Name des Servers. Möchte man diesen Namen ändern, müssen zwei Dateien angepasst werden. Die erste Datei ist dabei */etc/hostname* welche z.B. mit *nano* geöffnet werden kann:

```
nano /etc/hostname
```

Wenn der Name in dieser Datei geändert wurde, muss die Änderung noch fixiert

werden. Dies geschieht mit dem Befehl:

```
hostname -F /etc/hostname
```

Die zweite Datei in welcher der Name geändert werden muss ist die Datei */etc/hosts*. Nach der Änderung und einem Neustart des Servers ist der neue Name aktiv.

Wartung und Verwaltung

Unabhängig von der installierten Serversoftware wie einem Web- oder Mailserver gibt es eine Reihe von Aufgaben, die man auf diesen Systemen findet. Neben der Wartung eines Servers sollte man auch deren Verwaltung nicht unterschätzen. Ein Server ist (leider) kein System, welches man einmal installiert und dann sich selbst überlassen kann. Stattdessen ist ein gewisser Aufwand an Wartung und Pflege unerlässlich. Einige dieser Wartungs- und Pflegeaufgaben werden in diesem Kapitel beschrieben.

Verwaltung des Server

Die Verwaltung eines Server fängt bei den kleinen Dingen wie den Domains an. So kann man seine Server natürlich über ihre IP-Adressen verwalten, allerdings wird dies auf Dauer wahrscheinlich etwas mühselig. Sinnvoller ist es sich für die Verwaltung eine Domain zu registrieren.

Unter dieser Verwaltungsdomain, kann jeder einzelne Server eine Subdomain bekommen. Im Beispiel könnte das Ganze dann so aussehen:

```
kallisti.server.example.org

game.kallisti.server.example.org

web.kallisti.server.example.org
```

Ich persönlich gestalte die Subdomains dabei so, dass virtuelle Maschinen Subdomains zu ihren physikalischen Host-Servern bilden. So handelt es sich bei dem Server *kallisti* um einen physikalischen Server, bei den Servern *game* und *web* hingegen um virtuelle Maschinen, welche auf dem Host-Server *kallisti* laufen.

Upgrade des Servers

Im Lauf eines Serverlebens wird man nicht darum herum kommen, das Betriebssystem des Servers auf eine aktuelle Version zu aktualisieren. Dies ist spätestens dann nötig, wenn die genutzte Version nicht mehr mit Sicherheitsaktualisierungen versorgt wird.

Je nach verwendetem Betriebssystem gibt es dabei unterschiedliche Upgradepfade. So kann man z.B. bei *CentOS* nicht von einer Hauptversion auf die nächste Upgraden. Stattdessen wird empfohlen das System mit der neuen Hauptversion neu aufzusetzen. Andere Systeme wie *Arch Linux* nutzen sogenannte Rolling Releases, was bedeutet, dass es streng genommen keine Hauptversionen mehr gibt. Stattdessen installiert man ein solches System mit einer Snapshot genannten Installations-CD und aktualisiert es dann auf den aktuellen Stand.

Bei Ubuntu ist es so, dass man von der vorherigen Version immer auf die nächste Version upgraden kann. Eine Besonderheit gibt es bei den LTS-Versionen von Ubuntu, welche im Dreijahresrhythmus erscheinen. Eine LTS-Version kann nicht nur auf die nächste Version sondern auch auf die nächste LTS-Version aktualisiert werden.

In der Datei */etc/update-manager/release-upgrades* ist dabei aufgeführt, wie das System aktualisiert wird:

```
[DEFAULT]
default prompting behavior, valid options:
 never  - never prompt for a new distribution version
 normal - prompt if a new version of the distribution is available
 lts    - prompt only if a LTS version of the distribution is available
Prompt=normal
```

Wenn dort *normal* steht, wird das System von einer Halbjahresversion auf die nächste aktualisiert. Ist dort stattdessen *lts* eingetragen, so wird nur auf eine neue LTS-Version aktualisiert.

In diesem Fall soll exemplarisch das Upgrade von der Ubuntu-Version 13.04 auf 13.10 beschrieben werden. Im Normalfall unterscheiden sich die dafür erforderlichen Schritte nur minimal von Version zu Version. Vor einer Aktualisierung sollte man zuerst ein komplettes Backup des Server erstellen. Wie das funktioniert wird im entsprechenden Backup-Kapitel erläutert.

Im ersten Schritt sollte das Betriebssystem auf den neusten Stand gebracht werden:

```
apt-get update
```

```
apt-get dist-upgrade
```

Nachdem dies geschehen ist, kann der Update Manager installiert werden:

```
apt-get install update-manager-core
```

Mit dem Befehl:

```
do-release-upgrade
```

beginnt das Systemupdate. Nach einigen Überprüfungen werden die Paketdatenbanken auf den neusten Stand gebracht. Führt man die Aktualisierung per SSH auf einem entfernten Rechner durch, so wird ein zweiter SSH-Dienst auf dem Port 1022 gestartet. Dieser dient dazu, das System beim Fehlschlagen des Updates zu reparieren.

Das Upgrade nimmt auf aktuellen Systemen knapp 30 Minuten in Anspruch, bei größeren Installation kann sich die Zeit entsprechend verlängern. In der ersten Phase des Upgrades werden einige Dinge wie Abhängigkeiten, Speicherplatz und ähnliches

überprüft. Nachdem dies geschehen ist, wird man auf der Konsole darüber informiert, wie viele neue Pakete installiert werden sollen. Nach der Bestätigung beginnt der Update-Manager mit dem Upgrade auf die neue Distributionsversion.

Je nach verwendeter Serverhardware und der Größe der Installation kann das Upgrade dabei von 15 Minuten bis zu einigen Stunden dauern. Erfahrungsgemäß ist das Upgrade meist nach 30 bis 45 Minuten abgeschlossen.

Am Ende des Upgrade wird der Nutzer gefragt, ob nicht mehr benötigte Pakete entfernt werden können. Diese Frage kann im Normalfall immer mit *Ja* beantwortet werden. Damit ist das Upgrade beendet — der Server benötigt nun einen Neustart.

Mailserver

Wenn man sich das Internet betrachtet, dann ist E-Mail sicherlich eine seiner Killerapplikationen. Um Mails selber empfangen zu können, muss ein Mailserver eingerichtet werden. Die Einrichtung eines solchen Servers soll in diesem Kapitel beschrieben werden.

Aufsetzen eines Mailservers

Möchte man einen Mailserver auf seinem eigenen Server aufsetzen, kann man sich durch hunderte Seiten Dokumentationen wühlen. Allerdings geht das ganze auch einfacher. Für einen Mailserver benötigen wir dabei einen MTA (Mail Transfer Agent) und einen IMAP/POP3-Server. Als MTA wird in diesem Fall *Postfix* und als IMAP/POP3-Server *Dovecot* genutzt. Dankenswerterweise liefert Ubuntu bereits das Paket *dovecot-postfix* mit, welches nur installiert werden muss:

```
apt-get install dovecot-postfix
```

Bei der anschließenden Installation werden einige Fragen gestellt. So sollte man bei einem dezidierten Server die Konfigurationsart *Internet-Site* wählen. In der abschließenden Frage sollte der Domainname des Servers eingetragen werden. Dieser muss dabei nicht mit dem Domainnamen der späteren Mailadressen übereinstimmen. So könnte der Domainname *mail.example.com* sein, während die spätere Mailadresse *test@example.org* lautet.

Mailserver schauen nach, ob der Name der IP-Adresse (Reverse DNS) auch zum Mailserver passt. Nutzt man zum Beispiel die IP-Adresse *192.168.10.1* und deren Reverse DNS Name lautet *mail.example.com*, so muss auch der Hostname *mail.example.com* lauten. Den Hostnamen kann man in der */etc/hosts* Datei konfigurieren. Auch ein Blick in die */etc/postfix/main.cf*-Datei lohnt sich. Der Hostname (*myhostname*) sollte hier der gleiche sein. Auch das Überprüfen der Option *mydestination* wird empfohlen.

Der hier konfigurierte Mailserver soll für verschiedene Domains zuständig sein. Außerdem sollen bei diesem Setup keine Unix-Accounts für die Nutzer angelegt werden, die Verwaltung der Nutzer erfolgt rein virtuell. Auch auf Datenbanken zur Speicherung der Nutzer wird verzichtet, da sich dies erst ab zirka 100 Domains lohnt. Im ersten Schritt wird die Datei */etc/postfix/main.cf* um folgende Einträge erweitert:

```
virtual_mailbox_domains = /etc/postfix/virtual_domains

virtual_mailbox_base = /var/mail/vhosts

virtual_mailbox_maps = hash:/etc/postfix/vmailbox
```

```
virtual_alias_maps = hash:/etc/postfix/virtual_alias

virtual_minimum_uid = 100

virtual_uid_maps = static:5000

virtual_gid_maps = static:5000

virtual_transport = dovecot

mailbox_size_limit = 0
```

Anschließend wird der Nutzer erzeugt, welcher Zugriff auf alle lokalen Mailboxen hat:

```
groupadd -g 5000 vmail

useradd -s /usr/sbin/nologin -u 5000 -g 5000 vmail

usermod -aG vmail postfix

mkdir -p /var/mail/vhosts

chown -R vmail:vmail /var/mail/vhosts
```

Postfix wird bei der Gelegenheit in Zeile 3 auch der Gruppe *vmail* hinzugefügt. In den letzten beiden Zeilen wird der Ordner für die lokalen Mailboxen erzeugt und dem Nutzer *vmail* zugewiesen. Danach geht es an die Konfiguration der virtuellen Mailboxen:

```
nano /etc/postfix/virtual_domains
```

In der Datei *virtualdomains* werden die Domains festgelegt, für welche das System zuständig ist. Dabei wird jede Domain zeilenweise eingetragen:

```
example.com

example.org
```

Das Mapping der jeweiligen Mailadressen zu den Mailboxen findet in der Datei */etc/postfix/vmailbox* statt:

```
nano /etc/postfix/vmailbox
```

Dort wird auch wieder zeilenweise die jeweilige Mailadresse zur Mailbox zugeordnet.

```
webmaster@example.com example.com/webmaster

seeseekey@example.com example.com/seeseekey
```

Die jeweiligen Pfade der Mailbox sind dabei nicht absolut, sondern setzen sich aus dem in *virtualmailboxbase* definierten Pfad und dem jeweiligen Mailboxpfad zusammen. Der Slash am Ende der Mailbox-Definition führt dazu, dass Postfix als Speicherverfahren *maildir* anstatt *mbox* benutzt, wobei *maildir* vorgezogen werden sollte. Weiterleitungen von einer Mailadresse an eine andere, werden in der Datei */etc/postfix/virtualalias* definiert:

```
abuse@example.com seeseekey@example.com

abuse@example.org seeseekey@example.com
```

Postfix benötigt einige binäre Lookup-Tabellen, welche nach jeder Änderung mittels:

```
postmap /etc/postfix/vmailbox

postmap /etc/postfix/virtual*alias*
```

erzeugt werden müssen. Geschieht dies nicht, verwendet Postfix die alten Lookup-Tabellen, bis dies geschehen ist. Nun muss der LDA (Local Delivery Agent) in der */etc/postfix/master.cf*-Datei konfiguriert werden. Dieser nimmt die Mails vom MTA entgegen und speichert sie in den lokalen Mailboxen. Dazu fügen wir am Ende der Datei die Zeilen hinzu:

```
dovecot unix - n n - - pipe

flags=DRhu user=vmail:vmail argv=/usr/lib/dovecot/deliver -f $sender
-d $recipient
```

Nachdem die Mailboxen über Postfix fertig konfiguriert worden sind, geht es an die Detailkonfiguration von Dovecot. In der Datei */etc/dovecot/conf.d/10-auth.conf* wird die Option *#disableplaintextauth = yes* auskommentiert, damit keine Klartextauthentifikation möglich ist. Am Ende der Datei kommentieren wir die Zeile:

```
!include auth-system.conf.ext
```

aus und stattdessen die Zeile:

```
!include auth-passwdfile.conf.ext
```

ein. Anschließend wird die Datei *auth-passwdfile.conf.ext* bearbeitet. Die Datei sollte nach der Bearbeitung in etwa so aussehen:

```
passdb {
  driver = passwd-file
  args = scheme=sha512-crypt username_format=%u /var/mail/vhosts/passwd
}

userdb {
```

```
  driver = passwd-file
  args = username_format=%u /var/mail/vhosts/passwd
}
```

Im nächsten Schritt wird der LDA in *Dovecot* konfiguriert. In der Datei */etc/dovecot/conf.d/15-lda.conf* ergänzen wir dem *lda*-Block, so dass er anschließend wie folgt aussieht:

```
protocol lda {
Space separated list of plugins to load (default is global mail_plugins).
mail_plugins = $mail_plugins

  hostname = mail.example.com
  postmaster_address = postmaster@example.com
  auth_socket_path = /var/run/dovecot/auth-master
  mail_plugins = cmusieve
}
```

Nun muss noch der Speicherort der Mailboxen bekanntgegeben werden. Dazu wird in der Datei */etc/dovecot/conf.d/10-mail.conf* die *maillocation* von:

```
mail location = mbox:/mail:INBOX=/var/mail/%u
```

in

```
mail location = maildir:/var/mail/vhosts/%d/%n
```

geändert. Um Probleme mit der Authentifikation von Postfix zu vermeiden, muss die Datei */etc/dovecot/conf.d//10-master.conf* angepasst werden. Hier wird der Block:

```
unix_listener auth-userdb {
mode = 0666
user =
group =
}
```

in

```
unix_listener auth-master {
mode = 0666
user =
group =
}
```

geändert. Nach einem Neustart der Dienste mittels:

```
service dovecot restart

service postfix restart
```

sollte das Mailsystem funktionieren. Ist das wider Erwarten nicht der Fall, empfiehlt sich ein Blick in die Logdateien */var/log/mail.err* und */var/log/mail.log*.

Wenn die Informationen in diesen Dateien zu ungenau sind, hilft es, in der Konfigurationdatei */etc/dovecot/conf.d/10-auth.conf* die Optionen:

```
authverbose = yes

authdebug = yes
```

zu setzen. Dadurch ist die Ausgabe wesentlich ausführlicher, was die Fehlersuche enorm vereinfachen kann. Möchte man einen neuen Nutzer anlegen, so erzeugt man zuerst die passende Zeile für die *passwd*-Datei im *vhosts*-Verzeichnis:

```
echo info@example.com:$(doveadm pw -p "geheim123" -s sha512-crypt):5000:5000/var/mail/vhosts/example.com/info/
```

Nun muss der entsprechende Eintrag in der */etc/postfix/vmailbox* angelegt und einige Befehle ausgeführt werden:

```
postmap /etc/postfix/vmailbox

mkdir -p /var/mail/vhosts/example.com/info

chown -R vmail:vmail /var/mail/vhosts

sudo service dovecot restart

sudo service postfix restart
```

Natürlich kann man sich für das Anlegen neuer Nutzer auch ein entsprechendes Skript schreiben. Nach dem Neustart der Services ist das neue Postfach eingerichtet und kann genutzt werden.

Postgrey gegen Spam

Nachdem der Mailserver eingerichtet ist, wird man schon nach kurzer Zeit mit einem Problem konfrontiert: Spam. Der installierte Server ist in der Lage, Mails zu senden und zu empfangen. Allerdings enthält er keinerlei Maßnahmen zum Schutz vor unerwünschten Mails. An dieser Stelle setzt Postgrey, eine Greylisting Implementierung für Postfix, an.

Beim Greylisting werden Mails von einem unbekannten Sender erst einmal mit einem temporären Fehler beantwortet. RFC konforme Mailserver senden die Mail nach einer Verzögerung noch einmal. Bei Spamversendern ist dies meist nicht der Fall, womit ein Großteil des gewöhnlichen Spams nicht im System auftaucht. Die Installation ist dabei relativ einfach:

```
apt-get install postgrey
```

Nach der Installation muss die */etc/postfix/main.cf* Datei angepasst werden. Der

Zeile *smtpdrecipientrestrictions* wird dabei der Wert *checkpolicyservice inet:127.0.0.1:10023* hinzugefügt. Nachdem Postfix mittels:

```
service postfix restart
```

neu gestartet wurde, ist das Greylisting aktiv. In der Logdatei */var/log/mail.log* findet man, sobald man eine Mail von einem unbekanntem Sender bekommt, folgende Zeile:

```
Oct 10 10:32:39 service postfix/smtpd22287: NOQUEUE: reject: RCPT
from mail.example.org178.19.71.5: 450 4.2.0 <mailinglists@example.com>:
Recipient address rejected: Greylisted, see
http://postgrey.schweikert.ch/help/example.com.html; from=
<mail.example.org> to=<mailinglists@example.com> proto=ESMTP helo=
<mail.example.org>
```

Wenn die Mail erneut empfangen wird und die Greylistingzeit vorbei ist, wird Postgrey die Mail annehmen. Möchte man die Greylistingzeit ändern, so muss die Datei */etc/default/postgrey* bearbeitet werden. Für eine Verzögerung von 60 Sekunden könnte das Ganze dann wie folgt aussehen:

```
POSTGREY_OPTS="--inet=127.0.0.1:10023 --delay=60"
```

Natürlich muss der Service nach dieser Änderung neu gestartet werden.

Mails von anderen Postfächern einsammeln

Mit der Umstellung auf den eigenen Server möchte man eventuell auch andere Maildienste hinter sich lassen. Oder aber man möchte verschiedene Mailadressen in einem Postfach sammeln und diese in unterschiedlichen IMAP-Ordnern sammeln. Ein solches Feature gibt es z. B. bei Google Mail.

Für diesen Fall habe ich ein PHP-Skript mit dem Namen *collectmails.php* entwickelt, welches über die in der Einleitung erwähnte Seite zum Buch bezogen werden kann.

Dovecot und Ordnernamen

Wenn man bei einem normal konfigurierten Dovecot einen IMAP-Ordner mit dem Namen *test@example.org* anlegt, wird man im Mailprogramm folgende Ordnerstruktur zu sehen bekommen:

```
test@example

org
```

Der Grund ist darin zu finden, dass der Punkt im Falle einer Maildir-Konfiguration standardmäßig als Trennzeichen eingerichtet ist. Möchte man trotzdem IMAP-Ordner

mit einem Punkt anlegen, so hilft das Dovecot-Plugin *Listescape*. Die Handhabung ist dabei denkbar einfach. In der *conf.d/20-imap.conf* wird dazu die auskommentierte Zeile:

```
#mail_plugins =
```

durch

```
mail_plugins = listescape
```

ersetzt. Nun muss in der Datei *10-mail.conf* im vordefinierten Namespace *inbox* der Separator neu definiert werden:

```
namespace inbox {
Namespace type: private, shared or public
type = private

  separator = /

  ...
}
```

Er darf hier nicht auf einen Punkt gesetzt werden, da das ganze sonst nicht funktioniert. Nach einem anschließenden:

```
service dovecot restart
```

können dann auch neue Ordner mit einem Punkt im Namen angelegt werden. Auf bestehende Ordner wirkt sich das ganze nicht aus. Diese müssen bei Bedarf neu angelegt werden.

Bestimmte Mailadressen blockieren

Unter Umständen möchte man seine eigene Blacklist direkt auf dem Mailserver führen. Für diesen Zweck gibt es unter Postfix die Einstellung *checksenderaccess*. Dazu bearbeitet man die Datei */etc/postfix/main.cf* mittels des gewünschten Editors:

```
nano /etc/postfix/main.cf
```

In der Abteilung *smtpdrecipientrestrictions* wird der Eintrag:

```
check_sender_access hash:/etc/postfix/sender_access
```

ergänzt. In die neu anzulegende Datei *senderaccess* werden nun Einträge nach folgendem Schema definiert.

```
webmaster@example.org REJECT

spam@example.org REJECT
```

```
spam@example.com REJECT
```

Mittels *postmap* wird anschließend die binäre Repräsentation erzeugt und mittels *reload* die Konfiguration neu geladen:

```
postmap /etc/postfix/sender_access

service postfix reload
```

Damit werden Mails von den definierten Adressen immer abgewiesen. Möchte man ganze Domains ausschließen, muss die ganze Domain eingetragen werden:

```
example.org REJECT

example.com REJECT
```

Hinter dem REJECT kann zusätzlich noch eine Begründung für die Abweisung angegeben werden. Das könnte dann so aussehen:

```
example.org REJECT No spammers!

example.com REJECT No spammers!
```

Versionsnummer von Postfix ermitteln

In vielen Fällen ist es nützlich die installierte Version des Mail Transfer Agent *Postfix* zu kennen. So kann man schnell einschätzen, ob eine aktuelle Sicherheitslücke den eigenen Server betrifft. Auf der Konsole gibt man dazu:

```
postconf -d mail_version
```

ein. Anschließend erhält man eine Ausgabe nach dem Schema:

```
mail_version = 2.11.0
```

Da die Versionsnummer aus der *main.cf* ausgelesen wird, ist es wichtig den Parameter *–d* mit anzugeben. Somit wird nicht der Wert aus der Konfigurationsdatei, sondern die korrekte Versionnummer zurückgegeben.

Zertifikat für Dovecot erneuern

Von Zeit zu Zeit muss das Zertifikat, welches Dovecot nutzt erneuert werden. Die Gründe dafür können unterschiedlich sein, so ist es möglich das dass Zertifikat schlicht abgelaufen ist, oder es durch eine Sicherheitslücke wie *Heartbleed* gezwungener Maßen erneuert werden muss.

Zur Erzeugung eines neuen Zertifikates wird die Kommandozeile von *OpenSSL* genutzt. Um ein neues Zertifikat zu erzeugen, gibt man in der Konsole folgendes ein:

```
   openssl req -new -x509 -days 3650 -nodes -out
/etc/dovecot/dovecot.pem -keyout /etc/dovecot/private/dovecot.pem
```

Die zeitliche Gültigkeit des Zertifikats sollte man dabei je nach seinen Bedürfnissen über den Parameter *days* anpassen. Anschließend muss Dovecot mittels:

```
   service dovecot restart
```

neugestartet werden. Danach wird das neue Zertifikat genutzt.

Gameserver

Eine der beliebtesten Serverdienste sind Gameserver für Multiplayer-Spiele. Auf diesem Markt gibt es auch eine Reihe von spezialisierten Anbietern, welche solche Server anbieten. So kann man sich ohne Probleme einen Server für ein Online-First-Person-Shooter bestellen und hat mit diesem keinen Administrationsaufwand.

Aber darum soll es in diesem Kapitel nicht gehen. Stattdessen wird exemplarisch ein Minecraft-Server aufgesetzt.

Minecraft Server aufsetzen

Das Spielprinzip von Minecraft, welches mittlerweile von Microsoft aufgekauft wurde, lässt sich am besten als LEGO für Erwachsene beschreiben. In einer Welt, die nur aus Blöcken besteht, kann man sich ausleben und seiner Kreativität freien Lauf lassen.

Um einen Minecraft-Server aufzusetzen, muss im ersten Schritt Java mittels:

```
apt-get install openjdk-6-jre-headless
```

installiert werden. Nun kann die Version des installierten Javas (*java–version*) bestimmt werden. Diese sollte 1.6 oder höher sein:

```
OpenJDK Runtime Environment (IcedTea6 1.9.9) (6b20-1.9.9-0ubuntu1~10.04.2
OpenJDK Server VM (build 19.0-b09, mixed mode)
```

Nachdem damit die Grundvoraussetzungen erfüllt sind, wird mittels:

```
adduser minecraft
```

```
su minecraft
```

ein Nutzer für den Server angelegt und in diesen Nutzer gewechselt. Nun schreiben wir uns ein kleines Startskript mit dem Namen *start-server.sh*:

```
!/bin/sh
wget -N http://www.minecraft.net/download/minecraft_server.jar
java -Xmx1024M -Xms1024M -jar minecraft_server.jar nogui &
```

Dieses Skript lädt die aktuelle Version herunter und führt sie im Hintergrund aus. Nun fehlt nur noch ein Stoppskript, welches auf den Namen *stop-server.sh* hört:

```
!/bin/bash
pkill -SIGTERM java
sleep 5
```

```
pkill -SIGKILL java
```

Nach dem ersten Start des Servers werden einige Konfigurationsdateien angelegt. In die Datei *ops.txt* tragen wir den oder die Operatoren für den Server ein. In der Datei *server.properties* sind die grundlegenden Einstellungen für den Server zu finden. In diesem Fall wird die Whitelist aktiviert, indem die Eigenschaft *white-list* auf *true* gesetzt wird. In der *whitelist.txt* kann man dann die entsprechenden Nutzer eintragen, welche sich in die Welt einloggen dürfen.

Eine Karte für Minecraft

Eine Welt in Minecraft hat die Angewohnheit, mit der Zeit immer größer zu werden. In einem solchen Fall ist eine Karte natürlich sehr praktisch. Mit Hilfe des Tools *Minecraft Overviewer* kann eine solche Karte erstellt werden. Der Overviewer erzeugt dabei neben den Kartenkacheln auch eine Javascript-Anwendung, mit der diese betrachtet werden können.

Im ersten Schritt sollte man sich den Minecraft Overviewer mittels:

```
git clone https://github.com/overviewer/Minecraft-Overviewer.git
```

auf den Server holen. Mit dem Befehl wird das Git-Repository, in welchem sich der Quelltext befindet, auf den Server geklont. Im Git-Kontext bedeutet dies, dass das gesamte Repository herunter geladen wird. Falls Git nicht installiert ist, muss das entsprechende Paket mittels:

```
apt-get install git
```

installiert werden. Anschließend installieren wir einige Abhängigkeiten, um das *coverviewer* Modul zu kompilieren. Dies geschieht mittels:

```
apt-get install build-essential python2.6 python2.6-dev python-imaging py
cd Minecraft-Overviewer
python setup.py build
```

Der Minecraft Overviewer benötigt eine terrain.png-Datei. Diese kann aus der *minecraft.jar* extrahiert (z.B. mittels 7-Zip) werden und wird in den Minecraft-Overviewer Ordner kopiert. Nun kann man eine Karte erzeugen:

```
./overviewer.py ../world/ ../mcmap/
```

Damit wird die Karte für die Minecraft-Welt gerechnet. Ist die Welt größer, sollte man ein kaufmännisches Und (&) an die Befehlszeile anhängen. Damit wird der Prozess im Hintergrund ausgeführt. Dies ist praktisch, da er so nicht beendet wird, wenn man die SSH-Sitzung verlässt.

Webserver

Ein Webserver stellt Webseiten über Protokolle wie HTTP dem Nutzer zur Verfügung. In diesem Kapitel geht es um die Installation und Konfiguration eines solchen Servers. Hierbei wird speziell auf den Webserver *Nginx* eingangen; der frühere Platzhirsch *Apache* wird nicht behandelt, da er meiner Meinung nach vor allem für Einsteiger unnötig kompliziert ist.

Grundlegende Installation

Normalerweise nennt man einen Webserver auch LAMP-Server. Diese Abkürzung steht dabei für *Linux*, *Apache*, *MySQL* und *PHP* (in manchen Fällen auch *Python* oder *Perl*). In unserem Fall firmiert das ganze unter dem Kürzel LNMP, da wir anstatt Apache den Webserver *Nginx* einsetzten. Mittels *Nginx* ist ein Webserver mit *PHP* und *MySQL* in ein paar Minuten eingerichtet. Dazu installiert man unter Ubuntu folgende Pakete:

```
   apt-get install mysql-server nginx php5-fpm php5-curl php5-gd php5-
imap php5-mysql php5-curl php5-gd
```

Nach der Installation der nötigen Pakete geht es an die grundlegende Konfiguration des Webserver:

```
mkdir -p /var/www/example.org/root
mkdir -p /var/www/example.org/test
cd /var/www
chown -R www-data:www-data .
usermod -a -G www-data nutzername
```

In diesem Beispiel soll die Domain *example.org* sowie die Subdomain *test.example.org* eingerichtet werden. Deshalb legen wir zwei Ordner an. Anschließend teilen wir Nginx mit, welche Domains und Subdomains der Webserver verwalten soll. Dazu legen wir die Datei */etc/nginx/sites-available/example* an und füllen diese mit folgendem Inhalt:

```
server {
listen   80;
listen [::]:80;

root /var/www/example.org/root;
index index.php index.html index.htm;

server_name .example.org;

location ~ \.php$ {
fastcgi_pass unix:/var/run/php5-fpm.sock;
fastcgi_index index.php;
include fastcgi_params;
}
}

server {
listen   80;
listen [::]:80;

root /var/www/example.org/test;
index index.php index.html index.htm index.php;
```

```
server_name test.example.org;

location / {
auth_basic "Access denied";
auth_basic_user_file /var/www/example.org/root/.htpasswd;

autoindex on;
}

location ~ \.php$ {
fastcgi_pass unix:/var/run/php5-fpm.sock;
fastcgi_index index.php;
include fastcgi_params;
}
}
```

Im ersten Block wird die Domain *example.org* definiert und konfiguriert. Durch die Notation *.example.org* wird *Nginx* angewiesen, jede Subdomain, welche nicht definiert ist, auf die Hauptseite umzuleiten. Für die Domain *test.example.org* wurde außerdem ein Passwortschutz angelegt und die Verzeichnisansicht aktiviert. Damit die Konfiguration auch aktiv werden kann, muss eine symbolische Verknüpfung in das Verzeichnis */etc/nginx/sites-enabled* angelegt werden:

```
ln -s /etc/nginx/sites-available/example /etc/nginx/sites-enabled/example
```

Die bestehende *default*-Datei im *sites-enabled* Ordner enthält eine Beispielkonfiguration. Diese kann entweder entfernt oder konfiguriert werden:

```
server {
listen   80 default_server;
listen [::]:80 default_server ipv6only=on;

root /var/www/default/root;
index index.php index.html index.htm;

server_name _;
}
```

Mit dieser Konfiguration wird jede Seite auf Default umgeleitet, welche nicht in anderen Konfigurationen auftaucht. Nun muss nur noch der Webserver und der PHP-Dienst neugestartet werden:

```
service nginx restart

service php5-fpm restart
```

Ist der Server bereits online, reicht es auch, die Konfiguration neuzuladen:

```
service nginx reload
```

Sollten während der Konfiguration Fehler auftreten, hilft ein Blick in das entsprechende Log mittels:

```
cat /var/log/nginx/error.log
```

Ist alles richtig konfiguriert worden, läuft der Webserver auf Port 80 und wartet auf erste Anfragen.

HTTPS mit Nginx

Für verschlüsselte HTTP-Verbindungen (HTTPS) benötigt man ein Zertifikat. Dieses kann man sich von einer Zertifizierungsstelle (Certificate Authority, CA) ausstellen lassen. Der Haken an der Sache ist, dass dies Geld kostet (CACert und StartCom mal außen vor gelassen). Eine Alternative hierzu wäre es, das Zertifikat selbst zu erstellen. Bei Diensten, die man nur für einen kleinen Nutzerkreis - z.B. für die Familie - hostet, ist es auch vertretbar, die Zertifikatswarnung im Browser über sich ergehen zu lassen. Für die Zertifikate wird ein Ordner erstellt und in diesen gewechselt:

```
mkdir /etc/nginx/ssl

cd /etc/nginx/ssl
```

Nun werden das Zertifikat und der *Certificate Signing Request* erstellt:

```
openssl genrsa -out example.key 2048

openssl req -new -key example.key -out example.csr
```

Bei der Erstellung des *Certificate Signing Request* müssen einige Daten angegeben werden:

```
Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:Mecklenburg-Vorpommern
Locality Name (eg, city) []:Neubrandenburg
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Example Inc
Organizational Unit Name (eg, section) []:Skunk works
Common Name (e.g. server FQDN or YOUR name) []:example.org
Email Address []:webmaster@example.org

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Nun muss das Zertifikat noch signiert werden, bevor es verwendet werden kann:

```
openssl x509 -req -days 730 -in example.csr -signkey example.key -out example.crt
```

In diesem Fall ist das Zertifikat 730 Tage, also zwei Jahre gültig. Da die Signie-

rung nun abgeschlossen ist, kann das Zertifikat in *Nginx* eingebunden werden. Dazu wird die Datei */etc/nginx/sites-available/example* geöffnet, wobei *example* hier natürlich für die entsprechende Konfigurationsdatei steht. Dort sollte die SSL-Konfiguration vorgenommen werden:

```
server {
listen 443 ssl;

root /var/www/example/root;
index index.html index.htm;

server_name .example.org;

ssl_certificate /etc/nginx/ssl/example.crt;
ssl_certificate_key /etc/nginx/ssl/example.key;
}
```

Nach dem Aktualisieren der Konfiguration mittels:

```
service nginx restart
```

ist die verschlüsselte Verbindung für die eingerichtete Seite aktiv und kann genutzt werden.

Basic Authentification

Auch im Webserver *Nginx* lässt sich *Basic Authentification* einrichten. Darunter versteht man das Fenster, welches sich auf manchen Webseiten öffnet und nach einem Nutzernamen sowie einem Passwort fragt. Um diese Funktionalität zu nutzen, fügt man in der entsprechenden Konfigurationsdatei im Server folgenden Block hinzu:

```
location / {
auth_basic           "Access denied";
auth_basic_user_file /var/www/example.org/.htpasswd;
}
```

In dem definierten Ordner wird nun eine Datei mit dem Namen *.htpasswd* angelegt. Dieser Datei funktioniert dabei nach dem Schema *Nutzername:Passwort*. Mittels:

```
openssl passwd
```

kann der Passworthash für die Datei erzeugt werden. Alternativ kann das ganze auch über das *htpasswd*-Tool, welches im *apache2-utils*-Paket enthalten ist, erstellt werden:

```
htpasswd .htpasswd seeseekey
```

Die *.htpasswd*-Datei könnte dann so aussehen:

```
seeseekey:ppWAv1Wkrq/jg
```

Nach dem Neustart des Servers mittels:

```
service nginx restart
```

sollte die *Basic Authentification* funktionieren.

Directory Listing

Unter Directory Listing versteht man die Funktionalität des Webservers, die Verzeichnisstruktur als Webseite darzustellen. In Apache wird das Directory Listing in der *.htaccess*-Datei mit der Direktive:

```
Options +Indexes
```

aktiviert. Damit wird beim Aufruf eines Verzeichnisses ohne Indexdatei die Verzeichnisstruktur angezeigt. Auch der freie Webserver *Nginx* unterstützt dieses Verfahren. Bei ihm nennt sich die passende Direktive *autoindex* und wird in der Seitenkonfiguration eingetragen:

```
server {
location / {
   autoindex on;
}
}
```

Damit werden dann Verzeichnisse und Dateien im Browser angezeigt. Das Feature sollte natürlich nur bei Seiten eingeschaltet werden, wo dieses gewünscht ist.

Upload Limit

Standardmäßig liegt das Uploadlimit für Dateien einer *Nginx* Installation mit der entsprechenden PHP-Installation bei 2 MB. Damit stößt man natürlich bei vielen Anwendungen sehr schnell an die Grenze, so z. B. beim Upload von größeren Bildern. Möchte man dies ändern, müssen sowohl die *Nginx*-Konfiguration als auch die *PHP*-Konfiguration angepasst werden. Im ersten Schritt wird die *Nginx*-Konfuration bearbeitet:

```
nano /etc/nginx/nginx.conf
```

Dort wird im *http*-Block die Zeile:

```
client_max_body_size 1024m;
```

hinzugefügt. Unter Umständen muss hier auch die *clientbodytimeout*-Option mit einem höheren Timeout versehen werden. Dies sollte man allerdings durch eigene Tests herausfinden. Nachdem *Nginx* konfiguriert ist, geht es an die *php.ini*-Datei:

```
nano /etc/php5/fpm/php.ini
```

Dort müssen folgende Parameter gesetzt werden:

```
post_max_size = 1024M
upload_max_filesize = 1024M
```

Nachdem das erledigt ist, können *Nginx* und der PHP Service neugestartet werden:

```
service nginx restart
service php5-fpm restart
```

Anschließend verfügt man in diesem Fall über ein neues Upload Limit von 1024 MB.

MySQL Dump einspielen

Wenn man seinen Webserver installiert, ist es wahrscheinlich, dass Daten migriert werden müssen. So müssen unter Umständen bestehende MySQL-Datenbanken auf den Server umgezogen werden. Natürlich kann man in diesem Fall auf Softwarepakete wie *phpMyAdmin* zurückgreifen. Problematisch wird das ganze immer dann, wenn die einzuspielende Datenbank größer oder auf dem Server kein Webserver mit installiertem PHP vorhanden ist.

Für solche Fälle gibt es die MySQL-Kommandozeile (*mysql*). Dazu wird aus der bestehenden Datenbank ein Dump, ein kompletter Auszug einer Datenbank, erzeugt. Anschließend kann dieser in die neue Datenbank eingespielt werden. Existiert die Datenbank im Zielsystem noch nicht, so sollte im ersten Schritt mittels *mysql* eine Datenbank und ein Nutzer erstellt werden:

```
mysql> CREATE DATABASE datenbankname;
mysql> GRANT ALL PRIVILEGES ON datenbankname.* TO nutzername@localhost ID
mysql> quit
```

Anschließend kann der Dump in die neu angelegte Datenbank eingespielt werden:

```
mysql -u root -p datenbankname < dump.sql
```

Alternativ kann die Datenbank anstatt mit *root* auch mit dem neu angelegten Nutzer importiert werden.

HTTPS mit Let’s Encrypt

Neben der Möglichkeit selbstsignierte Zertifikate zu nutzen und solche teuer einzukaufen, gibt es mit der Zertifizierungsstelle *Let’s Encrypt* nun die Möglichkeit Zertifikate schnell, sicher und kostenlos zu erstellen.

Die Zertifizierungsstelle wird dabei unter anderem von der EFF und Mozilla unterstützt. Im Gegensatz zu anderen Lösungen ist der Prozess bei *Let’s Encrypt* hochgradig automatisiert, so das die Einrichtung schnell von statten geht.

Der offizielle Client hört auf den Namen Certbot (früher *Let's Encrypt Client*) und implementiert das ACME-Protokoll (*Automated Certificate Management Environment*) über welches der Prozess der Erstellung und Auslieferung abgewickelt wird. Neben dem offiziellen Client gibt es viele weitere Clients welche das ACME-Protokoll implementieren. Um *Let's Encrypt* unter Ubuntu zu nutzen, muss im ersten Schritt der Client installiert werden:

```
apt-get install letsencrypt
```

Nachdem der Client installiert wurde, kann mit der Erzeugung der Zertifikate begonnen werden. Im Gegensatz zum Webserver *Apache* wird unter *Nginx* die automatische Einrichtung der generierten Zertifikate noch nicht unterstützt. Aus diesem Grund werden nur die Zertifikate mit dem Client erzeugt. Dies geschieht mit dem Befehl:

```
letsencrypt certonly
```

Damit wird der interaktive Modus gestartet in welchem das Zertifikat erzeugt werden kann. Zuerst wird nach einer Mailadresse gefragt, mit welcher die Wiederherstellung in Notfällen möglich ist. Anschließend müssen die allgemeinen Geschäftsbedingungen akzeptiert werden.

Nun wird nach den Domains gefragt, für welche ein Zertifikat erstellt werden soll. Hier kann man mehrere Domains per Komma bzw. Leerzeichen getrennt angeben – allerdings scheint dies in der aktuellen Version nicht zu funktionieren. Stattdessen wird nur für die erste angegebene Domain ein Zertifikat erzeugt. Standardmäßig benötigt *Certbot* während der Generierung der Zertifikate Zugriff auf den Port 80. Hintergrund für dieses Verhalten ist das der Client kurz einen Webserver aufsetzt um die Kommunikation mit der CA durchzuführen.

Abgelegt werden die erzeugten Zertifikate dabei im Ordner */etc/letsencrypt/*. In diesem Ordner liegen neben den Stammzertifikaten auch die eigentlichen Zertifikate für die einzelnen Domains. Nun kann dieses Zertifikat in *Nginx* eingebunden werden. Dazu muss die Konfigurationsdatei der jeweiligen Seite (z.B. */etc/nginx/sites-available/example*) geöffnet werden. Im ersten Schritt wurde dazu in der Konfiguration eine Weiterleitung eingerichtet:

```
server {
listen 80;
listen [::]:80;

server_name .example.org;

return 301 https://$host$request_uri$is_args$args;
}
```

Diese Weiterleitung sorgt dafür das eine Verbindung über unverschlüsseltes HTTP

automatisch auf die verschlüsselte Variante umgeleitet wird. Weiter geht es mit der Konfiguration der verschlüsselten Verbindung:

```
server {
listen   443 ssl;
listen [::]:443 ssl;

ssl_certificate /etc/letsencrypt/live/example.org/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/example.org/privkey.pem;

...
```

Jedes erzeugte Zertifikat von *Let's Encrypt* ist 90 Tage lang gültig, so das ein automatischer Prozess eingerichtet werden sollte um die Zertifikate automatisch zu erneuern. Mit dem Befehl:

```
letsencrypt renew --agree-tos
```

kann dabei der Erneuerungsprozess angestoßen werden. Möchte man das ganze ohne Risiko testen, so sollte der Parameter *dry-run* angefügt werden:

```
letsencrypt renew --agree-tos --dry-run
```

Bei der Erneuerung der Zertifikate kann es nun vorkommen das man den *Nginx*-Server vorher beenden muss. Das ganze kann man in einem Skript erledigen:

```
!/bin/sh
service nginx stop
letsencrypt renew --agree-tos
service nginx start
```

Dieses Skript kann man nun zum Beispiel einmal in der Nacht per Cronjob ausführen. Der Client überprüft dabei ob eine Erneuerung notwendig ist und führt diese dann automatisch durch.

Webapplikationen

Webapplikationen wie CM-Systeme (WordPress, Drupal, Joomla und Co.) gibt es eine Menge. In diesem Kapitel soll es um konkrete Beispiele gehen, bestimmte Systeme zu installieren. Neben der Installation sollen auch einige Probleme mit den Systemen und deren Lösungen angesprochen werden.

Wordpress

Früher war WordPress das Blogsystem. Mittlerweile ist es schon seit vielen Jahren aus diesem Anspruch herausgewachsen und zu einem vollwertigen und einfach zu bedienenden CMS herangereift. Neben dem Kern bietet WordPress über sein Plugin

Repository jede nur erdenkliche Erweiterung.

Wordpress installieren

Für WordPress wird ein Webserver mit PHP und MySQL benötigt. Sind diese Vorrausetzungen gegeben, kann das Paket unter wordpress.org heruntergeladen werden. Auf dem Server kann das ganze auch mittels:

```
wget https://de.wordpress.org/latest-de_DE.zip

unzip latest-de_DE.zip
```

herunter geladen und entpackt werden. Nachdem das Paket entpackt wurde, muss es in den entsprechenden Ordner kopiert werden. Für die Domain *example.org* könnte dies der Ordner */var/www/example/main/* sein, wobei der genaue Ordner natürlich von der entsprechenden Nginx-Konfiguration abhängt.

Wenn sich die Dateien im entsprechenden Ordner befinden, muss nur noch die Webseite im Browser aufgerufen werden. Der WordPress-Installer führt den Nutzer dann durch die verbleibenden Schritte bis zur erfolgreichen Installation.

Lastprobleme unter Wordpress

Unter Umständen kann es bei älteren WordPress-Installationen passieren, dass das System unter Lastproblemen leidet. Diese äußerten sich darin, dass die Webseite bei jedem Aufruf hohe Last erzeugt. Im Laufe der Zeit kann diese Last dabei immer größer werden, bis der Server nur noch einen HTTP-500-Error von sich gibt. Im Backend von WordPress kann sich das ganze mit einer weißen Seite äußern, was im Normalfall auf zu wenig Speicher für PHP hinweist.

Bei derartigen Lastproblemen sollte man einen Blick in die WordPress-Datenbank werfen. Meist stößt man dabei auf die Tabelle *wpoptions*, welche im Normalfall nur ein paar MB groß ist. Allerdings kann es Fälle geben, wo diese Tabelle auf mehrere hundert Megabyte an Größe anwächst.

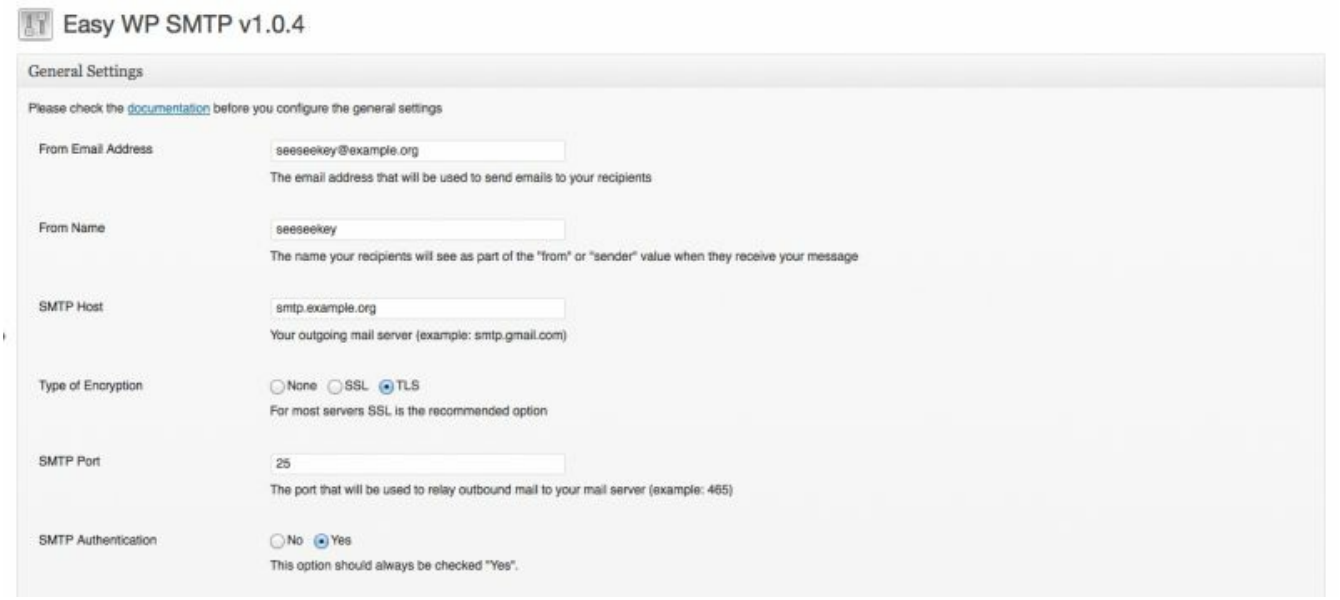
Bei diesen Dimensionen wird klar, wohin die Performance verschwindet. Im konkreten Fall gibt es meist viele *transient* und *displayedgalleries* Einträge. Letzte Einträge werden vom Plugin *NextGEN Gallery* erzeugt. Die *transient*-Einträge entstehen durch Caches (unter anderem für den RSS-Feed). Möchte man die Datenbank bereinigen helfen folgende SQL Befehle:

```
DELETE FROM wp_options WHERE option_name LIKE ('_transient_%')
DELETE FROM wp_options WHERE option_name LIKE ('displayed_galleries%')
```

Anschließend sollte die Tabelle optimiert werden, damit der von den gelöschten Einträgen belegte Speicherplatz wieder freigegeben wird.

Mails, Wordpress und SMTP

Unter Umständen kann es vorkommen, dass man sein WordPress auf einem Server betreibt, welcher keine PHP *mail()* Funktion unterstützt bzw. diese nicht konfiguriert ist. In diesem Fall sollten die Mails per SMTP versendet werden. Möglich wird dies mit dem WordPress-Plugin *Easy WP SMTP*. Damit werden die Mails nicht mehr über die PHP-Funktion sondern über einen SMTP Account versendet.



Die Einstellungen des Plugins Easy WP SMTP

Bei der Konfiguration des Plugins werden SSL und TLS unterstützt, so dass hier die Zugangsdaten nicht im Klartext durch das Netz gesendet werden.

ownCloud

Nachdem es gefühlte Jahrhunderte gedauert hatte, gibt es mittlerweile eine Lösung für das Problem, Dateien im Internet einfach von A nach B zu bewegen. Die beliebteste Lösung ist sicherlich der Dienst Dropbox. Allerdings ist es mit Nachteilen verbunden, seine Dateien bei einem fremden Dienstleister zu parken.

Befreien kann man sich aus dieser Bredouille mit dem freien Cloud-Dienst ownCloud, welcher unter anderem Funktionen zum Synchronisieren von Dateien sowie für die Kalender-, Kontakte- und Aufgabenverwaltung mitbringt. Daneben lässt sich die Funktionalität von ownCloud mittels Plugins erweitern.

ownCloud aufsetzen

Der erste Schritt zur Installation von ownCloud ist es sich die Installationsdateien von ownCloud unter owncloud.org herunterzuladen. Es empfiehlt sich dabei die Installation auf einer Domain zu installieren, welche über ein SSL-Zertifikat verfügt. Damit ist die spätere Verbindung verschlüsselt und kann nicht ganz so einfach abgehört wer-

den.

Nachdem das Installationsarchiv entpackt wurde, kann es in den entsprechenden Order unter */var/www/* gelegt werden. Nun muss die Domain im Webbrowser aufgerufen werden.

Der ownCloud Installer

Beim Ausführen des Installers kann es passieren das derselbe fehlende Dateirechte bemängelt:

```
Can't write into apps directory

This can usually be fixed by giving the webserver write access to the app
```

Wenn die Dateirechte korrigiert wurden sollte der Installer anschließend starten.

Nach dem Ausfüllen der Felder für die Datenbank und dem administrativen Account kann der Installer abgeschlossen werden. Mit dem vergebenden Nutzernamen und dem entsprechenden Passwort kann sich dann eingeloggt werden. Verfügt man über ein SSL-Zertifikat sollte man in den Einstellungen den Punkt *Erzwinge HTTPS* aktivieren.

Damit ist ownCloud auf dem Server installiert und kann genutzt werden. Für die

lokale Synchronisierung muss der entsprechende Client installiert werden. Nach dem Start des Clients wird man aufgefordert, den Pfad zur ownCloud Installation, sowie seinen Nutzernamen und das Passwort einzugeben. In den erweiterten Einstellungen kann auch der lokale Pfad verändert werden.

Die Servereinstellungen von ownCloud

Mit einem Klick auf den Button *Abschließen* beginnt die Synchronisation.

Password zurücksetzen

Wenn man das ownCloud-Passwort ändern möchte, so kann man dies über die *Passwort vergessen?*-Funktion zurücksetzen. Wenn dies nicht mehr möglich ist, so kann das Passwort in der Datenbank geändert werden:

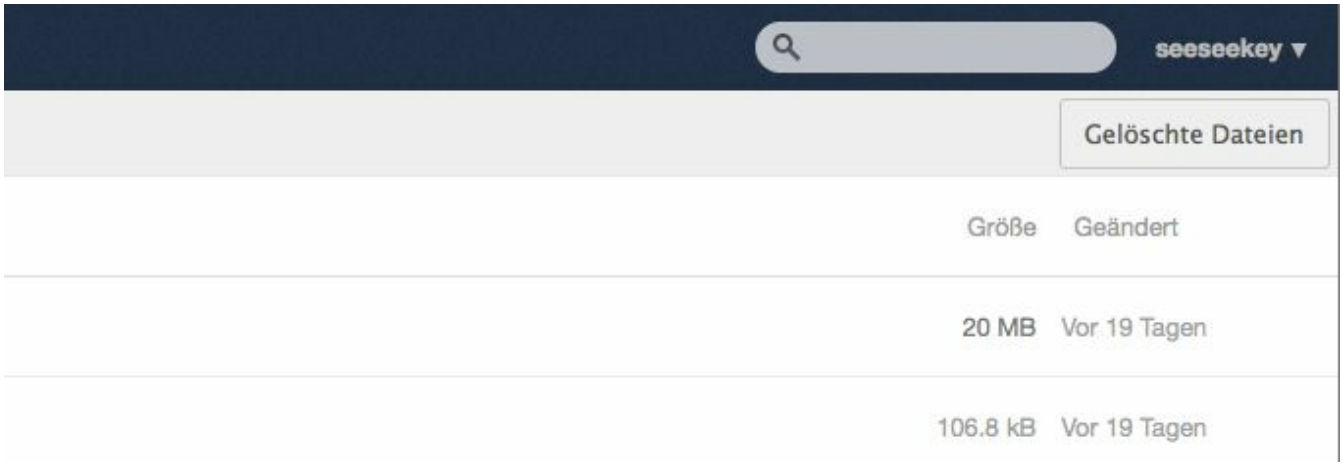
```
UPDATE oc_users SET password=SHA1('geheim') WHERE uid='nutzername'
```

Da die Passwörter mit einem Salt und mittels SHA1 gehasht gespeichert werden, muss das neue Passwort auch gehasht werden. Dies erledigt die Datenbank-Funktion *SHA1* für uns. Nachdem das Passwort zurückgesetzt wurde, sollte das Passwort nach dem Login über die *Persönlich*-Seite erneut geändert werden, damit wieder ein Salt für selbiges generiert wird.

Gelöschte Dateien entfernen

Bei einer ownCloud-Instanz, welche schon einige Betriebsstunden auf dem Buckel hat, kann es zu einem unschönen Effekt kommen. Beim Versuch, den Papierkorb über die Funktion *Gelöschte Dateien* zu leeren, versucht ownCloud, alle Dateien

aufzulisten, was allerdings nicht gelingt. Der Browser friert ein und das Leeren des Papierkorbes ist nicht möglich.



Der Button, um die gelöschten Dateien aufzurufen

Auch wenn das Leeren des Papierkorbes nach Meinung der ownCloud-Entwickler nicht notwendig ist, da die Dateien nach einer Weile weggeworfen werden, sollte es trotzdem eine Lösung geben, um den Papierkorb manuell zu leeren. Ein Workarround ist es, den Papierkorb direkt zu löschen, in dem man das Verzeichnis *owncloud/data/username/filestrashbin/* löscht. Anschließend müssen noch zwei Tabellen in der Datenbank bereinigt werden:

```
TRUNCATE TABLE oc_files_trashsize;

TRUNCATE TABLE oc_files_trash;
```

Eine weitere Möglichkeit, diese Problematik zu entschärfen, ist es, die Vorhaltezeit von gelöschten Dateien von 180 Tagen auf 30 Tage zu reduzieren. Dazu öffnet man die *config.php-Datei*, welche im *config*-Ordner zu finden ist und trägt folgenden Wert ein:

```
'trashbin_retention_obligation' => 30,
```

Damit wird verhindert, dass sich zu viele Dateien im Papierkorb ansammeln und das Problem wird deutlich entschärft.

Weitere Dienste

Neben den grundlegenden Diensten wie Mail und Web, welche man im ersten Moment mit einem Server in Verbindung bringt, gibt es auch andere wie Git, OpenVPN oder XMPP. Diese Dienste sollen in diesem Kapitel behandelt werden.

Git

Im Lauf der letzten Jahre hat sich das Versionverwaltungssystem Git zu einem der beliebtesten Systeme gemausert. Das schlägt sich nicht nur in den dem Nutzer sichtbaren Vorteilen wie Dezentralität nieder sondern auch auf der Serverseite. Ein Git-Server ist dabei innerhalb weniger Minuten aufgesetzt.

Hierbei werden in diesem Abschnitt zwei Arten von Git-Servern behandelt: zum einen ein Git-Server, welcher nur von einer Person, und zum anderen ein Git-Server, welcher von mehreren Nutzern genutzt wird.

Git-Server für einen Nutzer

Möchte man auf einem Ubuntu-System einen Git-Server aufsetzen, so ist dies schnell erledigt. Zuerst muss dafür *git* mittels:

```
apt-get install git
```

installiert werden. Danach wird der passende Nutzer für die Git-Repositories angelegt:

```
adduser git
```

Nun kann man ein bestehendes Repository zu diesem Server hochladen. Auf dem Server wird in den Kontext des Nutzers *git* gewechselt und dort ein passender Ordner sowie ein *rohes* Git-Repository angelegt:

```
su git
mkdir testproject.git
cd testproject.git
git init --bare
```

Dem lokalen Git-Repository wird mittels:

```
git remote add origin git@example.org:testproject.git
```

ein neuer Remote zugewiesen. Sollte bereits ein *remote* für *origin* existieren, so wird dieser mit:

```
git remote rm origin
```

entfernt. Anschließend kann das lokale Repository an den Server übertragen und auf Updates überprüft werden:

```
git push origin master

git pull origin master
```

Wenn gewünscht, kann man nun noch verhindern, dass man sich mittels des *git*-Accounts auf dem Server anmelden kann. Dazu muss die Datei */etc/passwd* editiert werden. Für den Nutzer *git* wird die Shell dabei von */bin/bash* in */usr/bin/git-shell* geändert. Anschließend kann man sich mit dem Account nicht mehr an der Shell anmelden.

Git-Server für mehrere Nutzer

Bei der vorherigen Methode wurde ein Git-Server aufgesetzt, welcher sich effektiv nur für einen Nutzer eignet. Natürlich kann man mit dieser Methode auch mehrere Nutzer zum Repositories verbinden, hat damit aber keine Möglichkeit mehr, Zugriffsberechtigungen für die Repositories zu setzen. Als Lösung für das Problem wird *Gitolite* für die Nutzer- und Rechteverwaltung genutzt. Im ersten Schritt werden auf dem Server die notwendigen Pakete installiert:

```
apt-get install git openssh-server perl
```

Als nächster Schritt wird der Nutzer angelegt, in welchem *Gitolite* läuft, und in diesen gewechselt:

```
useradd -m git
su git
```

Danach geht es an die Installation von *Gitolite*:

```
cd ~
git clone git://github.com/sitaramc/gitolite
mkdir bin
cd gitolite
./install -ln
```

Anschließend muss der öffentliche SSH-Schlüssel von dem Rechner, mit dem auf das System zugegriffen werden soll, in den Home-Ordner des *git*-Nutzers kopiert werden. Nun kann das Setup abgeschlossen werden:

```
cd ~/bin
./gitolite setup -pk $HOME/seeseekey.pub
```

Damit ist das Setup komplett und es kann mit der Konfiguration begonnen werden. Dazu wird vom Rechner, dessen Public Key beim Setup benutzt wurde, das entspre-

chende administrative Repository geklont:

```
git clone git@192.168.1.128:gitolite-admin
```

Die Dateistruktur des Repositories sieht dabei wie folgt aus:

```
conf
  gitolite.conf
keydir
  seeseekey.pub
```

In dem Verzeichnis *keydir* sind die SSH-Schlüssel enthalten. Um einen Nutzer hinzuzufügen, reicht es, einfach einen neuen öffentlichen Schlüssel in das Verzeichnis zu legen und das ganze ins Git-Repository einzubringen. Die eigentliche Konfiguration der Repositories erfolgt in der *gitolite.conf*-Datei. Diese sieht nach der Erzeugung so aus:

```
repo gitolite-admin
RW+     =   seeseekey

repo testing
RW+     =       @all
```

Das bedeutet, dass es zwei Repositores gibt; eines trägt den Namen *gitolite-admin* und dient der Verwaltung. Das zweite Repository ist *testing*, auf das alle Nutzer zugreifen dürfen. Benötigt man nun ein neues Repository, so fügt man einen neuen *repo* Abschnitt mit dem Namen und den entsprechenden Rechten hinzu. Sobald das ganze commitet und gepusht wurde, legt *Gitolite* das neue Repository an. Wenn man bei den Schlüsseln mehrere SSH-Schlüssel pro Nutzer wünscht, so legt man dafür am besten eine Verzeichnisstruktur an:

```
keydir
  seeseekey
rechner1
  seeseekey.pub
rechner2
  seeseekey.pub
```

Möchte man ein Repository löschen, so entfernt man es aus der *gitolite.conf* und löscht es anschließend auch vom Server. Damit hat man eine Lösung für Git- Server mit mehren Nutzern und entsprechender Verwaltung.

OpenVPN

Wenn man sich in einem unsicheren Netz bewegt - z.B. in einem offenen WLAN - empfiehlt es sich, ein VPN zu nutzen. Dafür gibt es die unterschiedlichsten VPN-Dienste, welche dies gegen einen monatlichen Obolus anbieten. Besitzt man einen eigenen Server, so kann man sich mittels OpenVPN einen solchen Dienst selber auf-

setzen.

OpenVPN einrichten

Um OpenVPN zu nutzen, muss im ersten Schritt das passende Paket installiert werden:

```
apt-get install openvpn
```

Nach der Installation wird aus den Beispieldateien das Skript *easy-rsa2* zum Erstellen der Zertifikate an den entsprechenden Ort kopiert:

```
cp -r /usr/share/doc/openvpn/examples/easy-rsa/2.0 /etc/openvpn/easy-rsa2
```

Für die Erzeugung der Zertifikate müssen einige Parameter angepasst werden. Dazu wird die entsprechende Datei im Editor geöffnet:

```
cd /etc/openvpn/easy-rsa2

nano vars
```

Der Wert Keysize *KEYSIZE* wird von 1024 auf 2048 erhöht. Am Ende der Datei befinden sich dann folgende Einträge:

```
export KEY_COUNTRY="US"
export KEY_PROVINCE="CA"
export KEY_CITY="SanFrancisco"
export KEY_ORG="Fort-Funston"
export KEY_EMAIL="me@myhost.mydomain"
export KEY_CN=changeme
export KEY_NAME=changeme
export KEY_OU=changeme
export PKCS11_MODULE_PATH=changeme
export PKCS11_PIN=1234
```

Nach der Konfiguration sollte das ganze dann in etwa so aussehen:

```
export KEY_COUNTRY="DE"
export KEY_PROVINCE="MV"
export KEY_CITY="Neubrandenburg"
export KEY_ORG="Example Inc."
export KEY_EMAIL="webmaster@example.org"
export KEY_CN="vpn.example.org"
export KEY_NAME="Example VPN"
export KEY_OU="VPN"
export PKCS11_MODULE_PATH=changeme
export PKCS11_PIN=1234
```

Die *vars*-Datei wird den Umgebungsvariablen hinzugefügt und anschließend das Masterzertifikat erstellt:

```
source ./vars
mkdir /etc/openvpn/easy-rsa2/keys
./clean-all
./build-ca
```

Damit ist die *Certificate Authority* (CA) erstellt. Nun geht es an die Erstellung der Schlüssel für den Server:

```
./build-key-server server
```

Beim Common Name gibt man die Domain ein, unter welcher der VPN-Server später erreichbar sein soll (z.B. vpn.example.org). Nach der Erstellung wird man im *keys*-Verzeichnis die Dateien *server.crt, server.csr* und *server.key* finden. Für den Diffie-Hellman-Schlüsselaustausch müssen entsprechende Parameter erzeugt werden:

```
./build-dh
```

Nun fehlen nur noch die Zertifikate für die entsprechenden Nutzer. Ein Nutzer wird dabei mittels:

```
./build-key nutzerOderClientName
```

erstellt. Damit sind alle benötigten Zertifikate und Schlüssel erstellt, so dass nun die restliche Konfiguration des Servers vorgenommen werden kann. Als Basis wird dabei die mitgelieferte Beispieldatei genutzt:

```
cp /usr/share/doc/openvpn/examples/sample-config-files/server.conf.gz /et
gunzip /etc/openvpn/server.conf.gz
```

Nach dem Kopiervorgang wird die Datei */etc/openvpn/server.conf* mittels *nano* geöffnet. Folgende Einstellungen werden dabei angepasst:

```
ca ca.crt -> ca ./easy-rsa2/keys/ca.crt
cert server.crt -> cert ./easy-rsa2/keys/server.crt
key server.key -> key ./easy-rsa2/keys/server.key

dh 1024.pem -> dh ./easy-rsa2/keys/dh2048.pem

;user nobody -> user nobody
;group nogroup -> group nogroup

;push "redirect-gateway def1 bypass-dhcp" -> push "redirect-gateway def1
```

Damit jegliche Kommunikation vom Client über den Server läuft, muss die Datei */etc/rc.local* auf dem Server um einige Einträge ergänzt werden:

```
iptables -t nat -F POSTROUTING
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -t nat -A POSTROUTING -s 10.8.0.0/24 -j SNAT --to-source IP_DES_
```

Nun ist der OpenVPN-Service konfiguriert und kann mittels:

```
service openvpn restart
```

neu gestartet werden.

OpenVPN und Logging

Wenn man einen OpenVPN-Server in der Standardkonfiguration betreibt, wird man sich unter Umständen wundern, an welcher Stelle ist das OpenVPN Log zu finden ist. Im Ordner */var/log/* ist keine nach OpenVPN benannte Datei zu finden.

Ursache hierfür ist, das der OpenVPN-Server die Logmeldungen im Normalfall an den Syslog-Deamon übergibt. Die Meldungen welche OpenVPN betreffen, kann man dabei mit dem Befehl *grep* aus der *syslog*-Datei filtern. Dazu muss auf der Konsole:

```
grep VPN /var/log/syslog
```

eingeben werden. Alternativ kann die OpenVPN-Konfiguration angepasst werden. Dazu muss die Datei */etc/openvpn/server.conf* bearbeitet werden. Dort gibt es eine Option *log-append*, welche wie folgt angepasst wird:

```
log-append /var/log/openvpn.log
```

Nach einem Neustart des Service mittels:

```
service openvpn restart
```

wird die definierte Logdatei genutzt.

XMPP

Neben dem Senden von Mails hat sich im letzten Jahrzehnt auch eine andere Art der schriftlichen Kommunikation über das Internet etabliert. Die Rede ist in diesem Fall vom Instant Messaging, welches man frei als *sofortige Nachrichtenübermittlung* übersetzen könnte.

In der Frühzeit gab es dabei Dienste wie ICQ, AOL Instant Messenger und einige andere. Das Problem an diesen Diensten ist, dass sie untereinander inkompatibel sind. Hier springt Jabber ein, bzw. XMPP, wie das Protokoll nun heißt. XMPP steht dabei für Extensible Messaging and Presence Protocol, was man grob mit *erweiterbares Nachrichten- und Anwesenheitsprotokoll* übersetzen könnte.

Dieses auf XML basierende Protokoll ist dabei seit 2004 von der IETF standardisiert. Im Gegensatz zu anderen Instant Messaging Systemen kann jeder bei XMPP einen Server aufsetzen und trotzdem mit anderen Nutzern kommunizieren, welche ihren Account auf anderen Servern haben.

Bei einem eigenen Server bietet es sich natürlich an, auch einen XMPP-Server zu betreiben. In diesem Buch wird dabei eine Lösung anhand des XMPP-Servers *ejabberd* beschrieben, welcher in der Programmiersprache Erlang geschrieben ist. *ejabberd* wird dabei von vielen namenhaften Firmen wie Facebook und GMX für ihre eigenen XMPP-Dienste genutzt.

ejabberd aufsetzen

Im ersten Schritt muss das passende Paket für *ejabberd* installiert werden:

```
apt-get install ejabberd
```

Der Server *ejabberd* unterstützt dabei die Verwaltung mehrerer Domains, so dass eine Serverinstanz die Adressen user1@example.org und user2@example.com bereitstellen kann. Um dies umzusetzen, geht es im ersten Schritt an die Konfiguration. Dazu wird die Datei */etc/ejabberd/ejabberd.cfg* mittels *nano* geöffnet:

```
nano /etc/ejabberd/ejabberd.cfg
```

Die Syntax der Datei ist dabei im Gegensatz zu anderen Konfigurationsdateien etwas gewöhnungsbedürftig, da sie auf der Programmiersprache Erlang basiert. Kommentare z.B. werden in dieser Datei mit einem Prozentzeichen eingeleitet. Im ersten Schritt werden die Domain respektive die Hostnamen definiert, welche *ejabberd* verwalten soll. Dazu wird der Punkt *hosts* gesucht und mit den entsprechenden Domains gefüllt.

```
{hosts, ["xmpp.example.com", "xmpp.example.org"]}.
```

Die Versprache kann für einen deutschsprachigen Server geändert werden:

```
{language, "de"}.
```

Damit werden Nachrichten vom Server in Deutsch anstatt in Englisch ausgegeben. Standardmäßig ist der Server so eingestellt, dass niemand außer dem Administrator Nutzer anlegen kann. Natürlich wäre es auch denkbar, einen Server zu betreiben, bei welchem sich jeder registrieren kann. Damit man Nutzer anlegen kann, muss ein Nutzer mit den entsprechenden Rechten definiert werden:

```
{acl, admin, {user, "seeseekey", "xmpp.example.com"}}.
```

Damit ist der Nutzer *seeseekey@xmpp.example.com* ein Nutzer mit administrativen Rechten. Allerdings muss dieser Nutzer noch erstellt werden:

```
service ejabberd restart

ejabberdctl register seeseekey xmpp.example.com geheimespasswort
```

Weitere Nutzer können anschließend über die Weboberfläche angelegt werden. Zu

erreichen ist diese unter:

```
http://xmpp.example.com:5280/admin
```

Die Zugangsdaten entsprechen dabei denen des administrativen Nutzers.

DNS SRV und XMPP

Wenn man einen XMPP-Server unter einer Subdomain wie *xmpp.example.org* betreibt, so sehen die Kontakte dementsprechend aus - in diesem Fall *kontakt@xmpp.example.org*. Möchte man Kontakte nach dem Schema *kontakt@example.org* anlegen, so muss der DNS-Eintrag der Hauptdomain auf den XMPP-Server zeigen. Dies ist aber nicht immer möglich. So kann es vorkommen, dass der Webserver auf der Hauptdomain seinen Dienst verrichtet, während der XMPP-Server auf einem anderen Server läuft. In einem solchen Fall kann man sich das ganze mit *SRV*-Einträgen im DNS zurechtzubiegen.

```
_jabber._tcp             IN SRV     0 0 5269 xmpp.example.org.
_xmpp-client._tcp        IN SRV     0 0 5222 xmpp.example.org.
_xmpp-server._tcp        IN SRV     0 0 5269 xmpp.example.org.
```

Mit diesen Einträgen ist es dann möglich, den XMPP-Server unter *example.org* zu betreiben, obwohl der A-Record auf einen anderen Server zeigt, da der anfragende Client umgeleitet wird.

Backup

Das schönste Backup ist eines, welches man nie benötigt. Aber leider ist dies meist ein Wunschdenken, denn irgendetwas passiert immer — Murphys Gesetz lässt grüßen: sei es ein Ausfall der Hardware, eine übereifrige Löschung im Dateisystem oder ein Angriff auf den Server, bei welchem dieser kompromittiert wurde. In solchen Fällen sollte ein vernünftiges Backup vorhanden sein.

Die meisten Anbieter von Mietservern bieten dabei meist dezidierte Backupserver an, welche mit einer entsprechend großen Leitung an die Server angebunden sind. Natürlich kann man seinen Server auch nach Hause sichern, allerdings kann dies je nach Leitung ein unpraktikabler Vorgang sein.

Grundlagen

Ein Backup weiß man immer erst dann zu schätzen, wenn es zu spät ist. Backups sind ein wichtiger Bestandteil der Datensicherheit. Dabei ist Backup nicht gleich Backup. Das ganze folgt einer gewissen Unterteilung.

Vollbackup

Ein Vollbackup ist eine komplette Sicherung aller Daten auf einen anderen Datenträger. Natürlich könnte man jeden Tag ein Vollbackup von seinem Server machen, allerdings würde dies in der Praxis einige Probleme bereiten. So ist ein Vollbackup eine Frage des Speichers, der Bandbreite und der Zeit pro Backup. Aus diesem Grund gibt es noch andere Verfahren.

Inkrementelles Backup

Bei einem inkrementellem Backup wird im ersten Schritt ein klassisches Vollbackup erstellt. Anschließend werden nur noch die Änderungen zum bestehenden Update gesichert. Hier muss man zwischen Verfahren unterscheiden, welche Dateibasiert arbeiten und solche welche Byteweise bzw. Blockweise arbeiten.

Verfahren, welche auf Dateibasis arbeiten, kopieren die geänderte Datei. Dies kann natürlich bei einem großen Image, welches z. B. eine virtuelle Maschine enthält, zu einem Problem führen. Schließlich möchte man nur den geänderten Bereich sichern. Das Problem bei einem reinen inkrementellen Backup ist die Tatsache, dass man für die komplette Wiederherstellung der Daten das Vollbackup und alle inkrementellen Backups benötigt.

Differentielles Backup

Ein differentielles Backup kombiniert die Vorteile eines Vollbackups mit dem des inkrementellen Backups. Dabei werden von Zeit zu Zeit Vollbackups erstellt. Die nachfolgenden inkrementellen Backups beziehen sich anschließend immer auf das letzte Vollbackup.

Den Server sichern

Nachdem ich einige Varianten für ein Server-Backup ausprobiert habe, bin ich schließlich bei einer Variante geblieben, welche auf dem Tool *rsync* basiert. Dieses Tool dient zur Synchronisation von Daten über ein Netzwerk und beschreibt auch das verwendete Protokoll.

Mittels *rsync* ein Backup zu erstellen, ist für den normalen Nutzer unter Umständen etwas schwierig. Wesentlich vereinfacht wird der Prozess von dem Skript *rsync-time-backup*. Dieses unter einer MIT-Lizenz stehende Skript funktioniert dabei unter Linux und Mac OS X. Die Installation besteht dabei aus dem Klonen des entsprechenden Git-Repositories auf den eigenen Rechner oder den Server.

```
git clone https://github.com/laurent22/rsync-time-backup
```

Genutzt wird das ganze nach dem Schema:

```
./rsync_tmbackup.sh <quelle> <ziel> [ausgeschlossene Dateien als Pattern]
```

Das Skript legt dabei für jedes Backup ein Verzeichnis mit dem Datum als Namen an. Dateien, welche bereits in einem vorherigen Backup vorhanden und identisch sind, werden dabei per Hardlink verknüpft. Damit belegen sie keinen zusätzlichen Speicherplatz. Das Wiederherstellen von Daten ist bei diesem Backup sehr einfach. Es reicht, den entsprechenden Ordner wieder zum Quellverzeichnis zu kopieren. Hierfür werden keine zusätzlichen Tools benötigt.

Möchte man vom entfernten Server auf ein lokales Medium sichern, muss man das Dateisystem des Servers lokal einbinden. Dies geschieht unter Linux und Mac OS X mittels *sshfs*. So könnte ein Backup unter Zuhilfenahme des Skriptes wie folgt aussehen:

```
sshfs server.example.org:/ /mnt/server
./rsync_backup /mnt/server /media/backup/server
umount /mnt/server
```

Wohin mit den Daten?

Neben der Frage, wie die Daten gesichert werden, stellt sich auch die Frage, wohin die Daten gesichert werden. Hier wären einige Möglichkeiten denkbar. Die erste Variante ist die Sicherung der Daten auf einen zweiten Server (hier Backup-Server genannt). Die zweite Variante ist die Sicherung der Daten auf einen anderen

Datenträger wie eine an den Server angeschlossene externe Festplatte. Bei der dritten Variante werden die Daten lokal bei Ihnen gesichert.

Bei der ersten Variante nutzt man einen zweiten Server, welcher Speicherplatz für ein Backup bereitstellt. Viele Anbieter von Servern bieten meist auch Backup-Speicherplatz an. Dieser Speicherplatz verfügt dann meist auch über eine entsprechende Anbindung, was sich positiv auf die Geschwindigkeit des Backups auswirkt. Nachteilig ist, dass man den Speicher meist extra bezahlen und dies bei mehreren 100 Gigabyte unter Umständen teurer als der Server werden kann.

Die zweite Variante ist die Sicherung auf einen externen Datenträger, welcher an den Server angeschlossen wird. Je nach Anbieter werden gegen Aufpreis solche Dienste angeboten. Vorteil bei dieser Variante ist die schnelle und direkte Anbindung, was sich vor allem im Falle einer Wiederherstellung bemerkbar macht. Nachteilig ist die örtliche Nähe zum Server. Sollte das Rechenzentrum abbrennen oder geflutet werden, ist auch die externe Festplatte nicht mehr zu gebrauchen. Auch wenn dieses Szenario weit hergeholt ist, hinterlässt es einen faden Beigeschmack bei dieser Variante in Punkto Datensicherheit.

Die dritte Variante ist es die Serverdaten auf dem eigenen Rechner oder einer lokalen Festplatte zu speichern. Der Vorteil dieser Variante ist die räumliche Trennung zwischen Serverstandort und dem der Datensicherung. Nachteilig wirkt sich die Dauer der Wiederherstellung aus, die von der verfügbaren Anbindung abhängt. Bei einer Leitung mit 16 MBit/s macht es keinen Spaß, die Daten wieder zurück zu sichern.

Für welche der Varianten man sich schlussendlich entscheidet, hängt von unterschiedlichen Anforderungen wie der maximalen Zeit bis zur Wiederherstellung, der verfügbaren Bandbreite, der Preis für den Backupspeichers und ähnlichem ab. Natürlich kann man die unterschiedlichen Varianten auch miteinander kombinieren.

MySQL-Datenbanken und deren Sicherung

Bei der Sicherung eines Servers mit einer MySQL-Datenbank kann es zu einem Problem kommen. Wenn man die Datenbank direkt sichert, kann es passieren, dass man am Ende einen inkonsistenten Datensatz sichert.

Damit dies nicht passiert, könnte man per Cronjob jeden Tag einen Dump der Datenbanken anlegen. Diese können dann konsistent gesichert werden.

Möchte man mittels *mysqldump* die einzelnen Datenbanken einer MySQL-Installation sichern, so sähe das so für eine Datenbank in etwa so aus:

```
mysqldump -u root -p<passwort> --result-file=example.sql --databases example
```

Packt man das nun in ein Skript, muss jede Tabelle von Hand in dieses Skript ein-

tragen werden. Allerdings könnte man das ganze auch über den SQL-Befehl *SHOW DATABASES* lösen und das ganze in ein automatisches Skript gießen. Auf der Webseite mit den Extras zum Buch kann ein solches Skript bezogen werden.

Sicherheit

Mit großer Macht geht große Verantwortung einher. Beim eigenen Server ist dies nicht anders. Schließlich handelt es sich dabei um einen Rechner, welcher mit einer guten bis sehr guten Anbindung an das Internet angeschlossen ist.

Mit Hilfe eines solchen Rechners können Kriminelle allerlei Schindluder treiben; deshalb sollte man bei seinem Server ein großes Augenmerk auf dessen Sicherheit legen. Missbrauch kann sich dabei in vielen Formen äußern. Ihr Server könnte als Spam-Schleuder fungieren oder für DDoS-Attacken genutzt werden. Natürlich kann ein Angreifer auch einfach Ihren Server derart verunstalten, dass Sie ihn neu aufsetzen dürfen.

Bei der Serversicherheit helfen schon wenige Maßnahmen, einen Server für Angreifer unattraktiv werden zu lassen. Die wenigsten Angreifer brechen in einen hoch gesicherten Server ein — stattdessen werden sich leichtere Ziele gesucht.

Allgemeine Vorsichtsmaßnahmen

Neben den speziellen Maßnahmen, welche im folgenden erläutert werden, gibt es auch allgemeine Maßnahmen, welche die Sicherheit eines Servers erhöhen.

Dies fängt mit den Nutzerkonten an. Jedes Nutzerkonto auf dem System sollte über ein ausreichend sicheres Passwort verfügen. Die Passwörter können dabei ruhig 32 oder 64 Zeichen lang sein. Oft kommt es vor, dass ein Angreifer sich Zugriff auf ein schlecht gesichertes Konto auf dem Server verschafft und anschließend versucht, mit Hilfe des gekaperten Kontos mehr Rechte auf dem Server zu erlangen.

Die zweite größere Maßnahme hängt mit geöffneten Ports zusammen. Jeder Server stellt eine Reihe von Diensten wie z.B. E-Mail über verschiedene Protokolle bereit. Je nach Protokoll kann es vorkommen, dass bestimmte Ports nicht benötigt werden. Diese sollten dann mit Hilfe einer Firewall-Regel nach außen hin abgeschirmt werden. In der Standardinstallation von Ubuntu sind im übrigen alle Ports geschlossen, nur dann wenn eine Serveranwendung installiert wird, werden von dieser die benötigten Ports geöffnet.

Ein weiterer Angriffspunkt ist die Fernwartung über SSH. SSH läuft im Standardfall auf dem Port 22. Ändert man diesen Port, hat man schon viel gewonnen, da somit viele Angriffsversuche ins Leere laufen. Wie die Umstellung des Ports genau läuft, wird im weiteren Verlauf dieses Kapitels erklärt.

Passwort-Authentifikation per SSH deaktivieren

Wenn man sich für seinen SSH-Zugang nur noch mit einem entsprechenden Schlüsselpaar anmeldet, kann die Authentifikation per Passwort deaktiviert werden. Dazu wird die */etc/ssh/sshdconfig*-Datei in einem Editor geöffnet:

```
nano /etc/ssh/sshd_config
```

Dort wird die Option:

```
Change to no to disable tunnelled clear text passwords
PasswordAuthentication yes
```

gesucht und in:

```
Change to no to disable tunnelled clear text passwords
PasswordAuthentication no
```

geändert. Anschließend muss der SSH-Server mittels:

```
service ssh restart
```

neu gestartet werden. Damit ist die Anmeldung per Passwort nicht mehr möglich und der Server ist wieder ein Stück sicherer.

Updates und Paketverwaltung

Ein wichtiger Punkt für einen sicheren Server ist das zeitnahe Einspielen von Updates. Der abgesicherte Server nützt nichts, wenn in ihm eine Sicherheitslücke klafft, welche bereits seit 4 Wochen mit einem Update zu schließen wäre.

Ein häufiger Fehler, welcher bei einem Sicherheitsupdate gemacht wird, ist es, die entsprechenden Dienste nicht neu zu starten. Nach einem Update wird der betroffene Dienst nicht automatisch neugestartet, dies muss der Nutzer selber erledigen. Um zu sehen, welche Dienste einen Neustart benötigen, kann man unter Ubuntu und Debian das Kommando:

```
checkrestart
```

nutzen. Dafür muss allerdings vorher das Paket *debian-goodies* installiert werden. Das Tool gibt dabei aus, welche Dienste aktualisiert wurden, aber im Moment noch in einer älteren Version laufen.

Auch gehört es zum guten Ton, in der Paketverwaltung von Zeit zu Zeit aufzuräumen. Dazu werden unter anderem die Kommandos *apt-get clean* und *apt-get purge* genutzt.

Server gegen unbefugten Login sichern

Beim Betrieb eines Servers wird man schnell feststellen, dass man nicht der

einzige ist, der gern Zugriff auf den Server hätte. Um zu häufige Loginversuche abzublocken, gibt es *Fail2ban*. Dieses Programmpaket durchsucht die entsprechenden Logs und blockiert böswillige Versuche, in das System einzubrechen. Damit gehört Fail2ban zu den Intrusion Prevention Systemen. Die Installation auf einem Ubuntu-Server geht dabei leicht von der Hand:

```
apt-get install fail2ban
```

Anschließend kann mit der Konfiguration begonnen werden:

```
nano /etc/fail2ban/jail.conf
```

In den ersten Einstellungen unter *DEFAULT* findet man die Zeit (in Sekunden), welche angibt, wie lange ein Angreifer geblockt werden soll. Standardmäßig sind dies 10 Minuten respektive 600 Sekunden. Im Bereich *JAILS* kann *Fail2ban* nun für bestimmte Dienste aktiviert werden, in dem der Wert von *enabled* auf *true* gesetzt wird. Selbstverständlich ist es auch möglich, eigene Jails zu definieren. Alle vorgefertigten Filterregeln findet man unter */etc/fail2ban/filter.d*. Die Überwachung des SSH-Dienstes ist dabei standardmäßig aktiviert. Nach der Anpassung der Konfiguration sollte der Dienst mittels

```
service fail2ban restart
```

neu gestartet werden. Das Log, in welchem alle Fail2ban-Aktionen verzeichnet sind, ist unter */var/log/fail2ban.log* zu finden.

Im Falle eines Falles

Absolute Sicherheit gibt es nicht und so kann es irgendwann einmal passieren, dass der eigene Server kompromittiert wurde. In einem solchen Fall sollte man wissen, was zu tun ist.

Der erste Schritt sollte sein, den entsprechenden Server vom Netz zu nehmen. Bei der im Buch empfohlenen Konfiguration mittels virtueller Maschinen kann man diese einfach herunterfahren. Auf der Hostmaschine sollten bis auf SSH und der KVM-Software keine weiteren Prozesse laufen.

Im zweiten Schritt sollte ein komplettes Backup vom Server erstellt werden. Dieses Backup sollte natürlich grundsätzlich als kompromittiert betrachtet werden.

Da der Angreifer irgendwie in das System gelangte, gilt es nun herauszufinden, wie er dies bewerkstelligte. Es bringt nicht wirklich etwas, den Server neu aufzusetzen, nur damit er anschließend wieder erfolgreich angegriffen wird. Hierzu sollte man sich anschauen, welche Server befallen sind. Auf den entsprechenden Servern sollte überprüft werden, ob sie mit der aktuellsten Software liefen und alle Sicherheitsupdates installiert wurden. Danach sollte man sich die Logdateien

vornehmen und in diesen nach Hinweisen suchen.

Nachdem herausgefunden wurde, durch welche Schwachstelle der Eindringling in das System kam, muss diese geschlossen werden. Anschließend wird der Server neu aufgesetzt. Dazu kopiert man natürlich nicht wieder 1 zu 1 das gesamte System, sondern man geht behutsam vor.

Vom bestehenden Backup werden nur die Daten übernommen. Standardsoftware wie WordPress u.ä. sollte neu heruntergeladen und installiert werden, da dies weniger aufwändig ist, als den gesamten Quellcode nach Hintertüren zu durchsuchen.

iptables und ufw

Wenn man unter Linux eingehende und abgehende Pakete kontrollieren, umleiten oder blockieren möchte, so kann man für diese Aufgabe *iptables* nutzen. Das Problem an *iptables* ist das es relativ kompliziert in der Anwendung ist.

Damit man hier nicht im Regen steht, gibt es (neben vieler anderer Firewall-Lösungen für Linux) die *uncomplicated firewall* kurz *ufw*. Technisch gesehen handelt es sich bei *ufw* um ein Frontend für iptables.

Seit der Version 8.04 (*Hardy Hedon*) ist *ufw* ein Bestandteil der Ubuntu-Distribution. Neben Ubuntu ist *ufw* unter anderem auch für Debian verfügbar. Installiert werden kann *ufw* mittels des Befehls:

```
apt-get install ufw
```

Standardmäßig ist *ufw* deaktiviert, so das die Installation im ersten Schritt keinerlei Auswirkungen hat. Den aktuellen Status sowie die definierten Regeln können dabei mittels:

```
ufw status
```

eingesehen werden. Das könnte dann z.B. so aussehen:

Status: Aktiv

```
Zu                     Aktion      Von
--                     ------      ---
8080/tcp               DENY        Anywhere
22/tcp                 ALLOW       Anywhere
22/tcp (v6)            ALLOW       Anywhere (v6)
```

Ist der Status auf inaktiv gesetzt, so muss *ufw* erst mit dem Befehl:

```
ufw enable
```

aktiviert werden. Hierbei muss man beachten das eine unbedachte Aktivierung von

ufw dazu führen kann das man sich aus dem Rechner aussperrt. Dies liegt daran das am Anfang keinerlei Regeln definiert sind – so werden Pakete an Port 22 ignoriert; dies führt dazu das keine Verbindung per SSH möglich ist. Um dem vorzubeugen sollte eine Regel für SSH definiert werden, bevor *ufw* aktiviert wird:

```
ufw allow 22/tcp
```

Bei dieser Schreibweise handelt es sich um die vereinfachte Form zum Anlegen einer Regel. Neben *allow*, sind dabei auch die Werte *deny* und *reject* möglich. Während bei *allow* die Pakete passieren können, werden sie bei *deny* blockiert – im Gegensatz dazu wird bei *reject* der Absender darüber informiert das die Pakete abgelehnt wurden. Möchte man komplexere Regeln definieren nutzt man *ufw* nach folgendem Schema:

```
ufw allow proto tcp from any to 127.0.0.1 port 1234
```

Damit werden alle Verbindungen per TCP von beliebigen IP-Adressen an die spezifizierte IP-Adresse weitergeleitet. Als Port wird als Eingangs- und Zielport Port 1234 genutzt. Die Regeln welche *ufw* verwaltet werden in drei Dateien gespeichert:

```
/etc/ufw/before.rules
/var/lib/ufw/user.rules
/etc/ufw/after.rules
```

Abgearbeitet werden die Regeln in der Reihenfolge wie oben angegeben – somit könnte eine Regel in der *user.rules*-Datei definiert sein, welche anschließend von einer anderen Regel in der *after.rules*-Datei überschrieben wird. Die selbst definierten Regeln sind dabei in der *user.rules* zu finden. Neben dem Anlegen ist es natürlich auch möglich Regeln wieder zu löschen. Für obige SSH-Regel würde das so aussehen:

```
ufw delete allow 22/tcp
```

Daneben ist es möglich *ufw* auf die Standardeinstellungen zu setzen. Dazu dient der Befehl:

```
ufw reset
```

Alle Regeln werden dabei auf die Standardeinstellungen zurückgesetzt. Für die bestehenden Regeln wird ein Backup im Verzeichnis */etc/ufw/* angelegt. Möchte man *ufw* wieder deaktivieren, so nutzt man den Befehl:

```
ufw disable
```

Beim beschriebenen *reset*-Befehl wird *ufw* ebenfalls deaktiviert, so das *ufw* nach diesem wieder aktiviert werden muss.

Weiterführendes

Dieses Buch soll einen Einblick in den Betrieb eines Servers geben, kann aber natürlich nicht das komplette Programm abdecken. In diesem Kapitel werden Literatur und Webseiten vorgestellt, welchen einer weiterführenden Einblick in bestimmte Themen wie Linux, Netzwerk und ähnliches bieten.

Bücher

Linux – Das umfassende Handbuch

Das Buch *Linux – Das umfassende Handbuch* ist landläufig nur unter dem Namen *Koffler*, nach seinem Autor Michael Koffler bekannt. Es ist dabei ein umfassendes Standardwerk und erklärt Linux dabei von A - Z. Bezogen werden kann es unter anderem über die Webseite des Autors.

Webseiten

ubuntuusers.de

Bei Problemen rund um Ubuntu hat sich im deutschsprachigen Bereich die Community *ubuntuusers* herauskristallisiert. Neben dem umfangreichen Wiki, gibt es auch ein gut besuchtes Forum in welchem man bei Fragen fachlich fundierten Rat findet.

Zum Buch

In diesem Kapitel finden Sie weitere Informationen zum Buch, wie den Zugang zu den Extras und die Neuerungen der aktuellen und vergangenen Versionen.

Extras

Neben dieser Version des Buches gibt es auf der Webseite des Autors unter seeseekey.net einen Bereich, aus dem weitere Fassungen wie die DRM-freie PDF-Version heruntergeladen werden können:

URL: https://seeseekey.net/buecher/selfhosting-extras

Password: 2hf08fnc20x9dh2yy0

Verlag:
BookRix GmbH & Co. KG
Sonnenstraße 23
80331 München
Deutschland

Tag der Veröffentlichung: 29.06.2015

https://www.bookrix.de/-gzd9a8a89a2ca45

ISBN: 978-3-7368-8984-2

BookRix-Edition, Impressumanmerkung
Wir freuen uns, dass Du Dich für den Kauf dieses Buches entschieden hast. Komme doch wieder zu BookRix.de um das Buch zu bewerten, Dich mit anderen Lesern auszutauschen oder selbst Autor zu werden.

Wir danken Dir für Deine Unterstützung unserer BookRix-Community.

Table of Contents

Einleitung 5
Beschaffung 7
Linux-Grundlagen 10
Grundeinrichtung 22
Wartung und Verwaltung 31
Mailserver 34
Gameserver 43
Webserver 45
Grundlegende Installation 46
  Webapplikationen 53
Weitere Dienste 59
  Git 59
  OpenVPN 61
  XMPP 64
Backup 67
Sicherheit 71
Weiterführendes 76
  Bücher 76
  Webseiten 76
Zum Buch 77