Snapshot-Backups mit Rsync

Unwetterzentrale

Backups sind wie Regenschirme: Hat man einen, passiert nichts. Vergisst man ihn, dauert es nicht mehr lange bis zur Katastrophe. Zumindest gegen drohende Datenschäden helfen ein kleiner Server, ein paar Festplatten und Linux-Bordmittel. Peer Heinlein

Ein gutes Backup muss schnell angefertigt, zuverlässig rekonstruierbar und automatisierbar sein. Bereits simple Vollkopien des Datenbestands auf andere Festplatten erfüllen diese Grundanforderungen. Wünschenswert und zeitgemäß sind jedoch Backups übers lokale Netz, die zudem zu mehreren Zeitpunkten Snapshots speichern, was versehentlich gelöschte Dateien zu einem zweiten Leben verhilft. Dafür mehrere Vollkopien – egal ob auf Festplatten oder Bändern – anzufertigen, ist zu aufwändig. Es bieten sich inkrementelle Backups an. Die hier vorgestellte Lösung vereint trickreich die Vorzüge beider Arten.

Ein Hot-Backup auf Festplatten

Die Festplatten aktueller Client-Systeme sind so riesig, dass geeignete zentrale Bandlaufwerke nur im Profibereich noch halbwegs wirtschaftlich sind. Im Vergleich zu Bändern sind Festplatten zudem erheblich schneller beschreibbar. Im Schadensfall sind die Daten auch super schnell im Zugriff – übers Netzwerk, auf diesem Wege sind sie ja auch auf den Backup-Server gekommen, oder im Notfall durch einen kurzen Einbau in den betroffenen Host. Alles in allem ist festzustellen, dass Festplatten ein passables Sicherungsmedium sind.

Eine gute Lösung ist ein kleiner Server mit einem Raid-1-System mit insgesamt drei Hot-Swap-Festplatten, von denen zwei immer im Raid-Array im Server laufen, die dritte lagert der Admin extern. Schnell ist so für wenig Geld ein Backup-Server mit 160 GByte Speicherkapazität oder mehr zusammengebaut. Etwas billiger und langsamer ist es, abwechselnd zwei mobile USB-Festplatten zu nutzen. Natürlich dürfen die Verantwortlichen nicht vergessen, die externe Sicherungsplatte in einem regelmäßigen Zyklus, zum Beispiel jeden Freitagabend, gegen eine aktuelle Platte aus dem Backup-Server auszutauschen. Denn es nützt nichts, wenn ein Brand oder Wasserschaden Hosts und Backups zusammen zerstört.

Sicheres Backup über das Netzwerk

Bei einem Vollbackup werden fast alle Daten eines Hosts über das Netzwerk transportiert – Passwort- und ähnlich sensible Dateien eingeschlossen. Eine Verschlüsselung der Daten ist darum zwingend. Das hier vorgestellten Beispiel greift auf das bewährte SSH zurück, um die Daten zwischen dem zu sichernden PC und dem Backup-Server zu transportieren.

Ganz simpel das auf SSH beruhende SCP dafür heranzuziehen wäre unelegant, denn es würde bei jedem Backup sämtliche Daten des Hosts kopieren und damit immense Datenmengen. Sinnvoller ist dagegen die Kombination aus Rsync und einem SSH-Tunnel (siehe **Abbildung 1**). SSH – aktuelle Implementierungen vorausgesetzt – ist so sicher, dass die beteiligten Rechner nicht mal im eigenen LAN stehen müssen. Zentrale Backups aus anderen Rechenzentren oder Filialen ohne VPN-Anbindung sind damit kein Problem mehr.

Clever abgecheckt

Rsync seinerseits prüft dateiweise mit seinen cleveren Algorithmen vor der Übertragung, ob auf dem Zielsystem bereits identische Kopien vorliegen. Gleiche Dateien überträgt Rsync dann nicht. Mehr noch: Weichen ursprünglich gleiche Dateien voneinander ab, überträgt

Rsync nur die geänderten Bereiche, sodass es bei ein paar neuen Zeilen am Ende eines etliche MByte großen Logfile nur wenige Bytes zu übertragen braucht statt des kompletten File. [1]

Nur der erste Lauf ist prinzipbedingt aufwändig: Er muss nämlich sämtliche Daten über das Netz übertragen, jeder weitere Lauf hingegen fällt in der Praxis ungleich kürzer aus, die Netzbelastung ist dann im Vergleich viel geringer. Rsync misst die Ersparnis und gibt sie am Ende als Speedup-Faktor aus. Die Datenkompression von SSH ist dabei natürlich hoch willkommen. Ein einzelnes

Listing 1: »backup-rsync«

```
001 #!/bin/bash
002 #
003 # Das Skript zieht per Rsync Backups
004 # http://www.heinlein-partner.de
005 #
006 # Aufruf: backup-rsync <FQDN-Servername>
007 #
008
009 # ### Aufrufparameter des Skripts ist ein FQDN-Hostname
010 if [ -n "$1" ] ; then
011         SERVER="$1"
012 else
013         echo "Error: Usage $0 <fqdn-hostname>"
014         exit
015 fi
016
017 # ### Konfiguration
018 # Pruefen, ob noch ein gewisser Prozentsatz
019 # an Plattenplatz und Inodes frei ist?
020 CHECK_HDMINFREE=true
021 HDMINFREE=90
022
023 # Soll die Daten-Partition readonly gemountet werden,
024 # wenn sie nicht in Gebrauch ist?
025 MOUNT_RO=false
026 MOUNT_DEVICE=/dev/hdc1
027
028 # Unter welchem Pfad wird gesichert?
029 DATA_PATH=/DATA
030
031 # Liste von Dateipattern, die nicht gesichert werden sollen
032 EXCLUDES=/etc/rsync-excludes
033
034 # Weitere Optionen für Rsync. Eventuell ist eine Limitierung
035 # der Bandbreite sinnvoll, Angabe in Kbyte/s:
036 # EXTRAOPT="--bwlimit=256"
037 EXTRAOPT=""
038
039 # ### Let's Rock`n`Roll
040
041 # Pruefe auf freien Plattenplatz
042 GETPERCENTAGE='s/.* \([0-9]\{1,3\}\)%.*/\1/'
043 if $CHECK_HDMINFREE ; then
044         KBISFREE=`df /$DATA_PATH | tail -n1 | sed -e "$GETPERCENTAGE"`
045         INODEISFREE=`df -i /$DATA_PATH | tail -n1 | sed -e
    "$GETPERCENTAGE"`
046         if [ $KBISFREE -ge $HDMINFREE -o $INODEISFREE -ge $HDMINFREE ]
    ; then
047                 echo "Fatal: Not enough space left for rsyncing
    backups!"
048                 logger "Fatal: Not enough space left for rsyncing
    backups!"
049                 exit
050         fi
051 fi
052
053 # Festplatte rw remounten falls gewuenscht!
054 if $MOUNT_RO ; then
055         if `mount -o remount,rw $MOUNT_DEVICE $DATA_PATH` ; then
056                 echo "Error: Could not remount $MOUNT_DEV readwrite"
057                 logger "Error: Could not remount $MOUNT_DEV readwrite"
058                 exit
059         fi
060 fi
061
062 # Gegebenenfalls Verzeichnis anlegen
063 if ! [ -d $DATA_PATH/$SERVER/daily.0 ] ; then
064         mkdir -p $DATA_PATH/$SERVER/daily.0
065 fi
066
067 # Los geht`s: Rsync zieht ein Vollbackup
068 echo "Starting rsync backup from $SERVER..."
069 logger "Starting rsync backup from $SERVER..."
070
071 rsync  -avz --numeric-ids -e ssh --delete --delete-excluded  \
072         --exclude-from="$EXCLUDES"  $EXTRAOPT                \
073         $SERVER:/ $DATA_PATH/$SERVER/daily.0
074
075 # Rückgabewert pruefen.
076 # 0 = fehlerfrei,
077 # 24 ist harmlos; tritt auf, wenn waehrend der Laufzeit
078 # von Rsync noch (/tmp?)-Dateien verändert oder geloescht wurden.
079 # Alles andere ist fatal -- siehe man (1) rsync
080 if ! [ $? = 24 -o $? = 0 ] ; then
081         echo "Fatal: rsync finished $SERVER with errors!"
082         logger "Fatal: rsync finished $SERVER with errors!"
083 fi
084
085 # Verzeichnis anfassen, um Backup-Datum zu speichern
086 touch $DATA_PATH/$SERVER/daily.0
087
088 # Fertig!
089 echo "Finished rsync backup from $SERVER..."
090 logger "Finished rsync backup from $SERVER..."
091
092 # Sicher ist sicher...
093 sync
094
095 # Festplatte ro remounten falls gewuenscht!
096 if $MOUNT_RO ; then
097         if `mount -o remount,ro $MOUNT_DEVICE $DATA_PATH` ; then
098                 echo "Error: Could not remount $MOUNT_DEV readonly"
099                 logger "Error: Could not remount $MOUNT_DEV readonly"
100                 exit
101         fi
102 fi
```

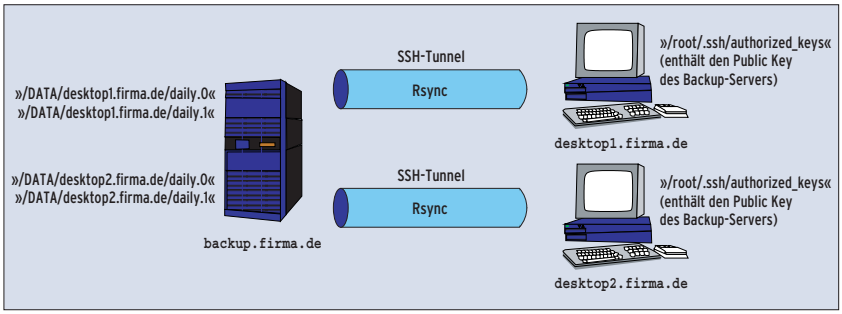

Abbildung 1: Die Kombination aus Rsync und einem SSH-Tunnel ist sicher und spart viel Netzlast.

Rsync-Kommando reicht schon, um die gesamte Festplatte eines Hosts zu übertragen. Wenn man die Platte bedarfsabhängig read-only mountet und ein paar Logging-Funktionen drum herum strickt, entsteht Listing 1.

Viele Rsync-Parameter

Unter den Rsync-Parametern sorgt »-a« für einen Archivmodus, der Zeiten und Zugriffsrechte übernimmt. Der Parameter »--numeric-ids« verhindert das Umwandeln der Dateibesitzrechte auf die des Zielsystems. Gelöschte Dateien auf dem Host sollen auch auf dem Backup-System gelöscht werden, »--delete« und »--delete-excluded« erledigen das.

Apropos Excludes: Wer ordentlich Platz auf seinem Backup-System hat, sichert möglichst alle Dateien seiner Backup-Clients. Die paar zusätzlichen MBytes sind gut angelegt, nur zu leicht ist es sonst, etwas Wichtiges im Backup zu vergessen. Trotzdem erlaubt es die Datei »/etc/rsync-excludes«, eine Liste von Verzeichnissen und Datei-Pattern zu definieren, die Rsync bei seiner Arbeit auslässt, Listing 2 zeigt eine.

Wichtigste Ausnahme ist hier das »/proc«-Verzeichnis, das für ein Restore keinen Wert besitzt und (wegen »/proc/kcore«) entsprechend groß ist. Überlegenswert ist es auch, auf die Datei »/etc/shadow« zu verzichten. In ihr stehen die Passwörter aller Nutzer. Fiele der Backup-Server einem Angreifer in die Hände, hätte der dank Cracktools wenig Probleme an die Root-Passwörter der Hosts zu kommen.

Fehlt »/etc/shadow« im Backup, bedeutet das allerdings auch Arbeit: Wirft die Festplatte eines Clients endgültig das Handtuch, spielt der Admin das Backup zurück und muss die Shadow-Datei samt Passwörtern manuell neu aufbauen. Jeder Administrator sollte selbst entscheiden, welche Strategie er hier verfolgen mag.

Listing 2: »/etc/rsync-excludes«

```
01 /proc/*
02 /var/virusmails/*
03 /tmp/*
04 /etc/shadow
```

Limitierte SSH-Schlüssel schützen vor Missbrauch

Zu beachten ist die Art, wie der SSH-Zugriff des Backup-Servers auf die Hosts erfolgen soll. Über ein Passwort? Dann wäre kein automatisierter Cronjob möglich. Über einen SSH-Key ohne Passphrase? Dann hätte ein beim Backup-Server erfolgreicher Angreifer eine Root-Shell auf jedem Client-Host. SSH bietet daher die Möglichkeit, einen Schlüssel auf den Aufruf eines genau festgelegten Kommandos zu limitieren – hier auf den Start von Rsync. Eine universelle Root-Shell ist dann vom Server aus nicht mehr aufrufbar.

Zuerst erzeugt der Administrator auf jedem Backup-Host mit dem Programm »ssh-keygen« einen SSH-Schlüssel für den User »root«. Dann fügt er in die Datei »/root/.ssh/id_dsa.pub«, wie in Listing 3 zu sehen ist, das Rsync-Kommando ein, das der zu sichernde Host später ausführt. Die Datei »id_dsa.pub« ist auf den einzelnen Hosts als »/root/.ssh/authorized_keys« zu hinterlegen. Danach darf sich der Backup-Server als Root auf allen Hosts einloggen und dort ausschließlich das vorbestimmte Kommando ausführen.

Listing 3: SSH Public Keys limitieren

```
01 command="rsync --server --sender -vlogDtprz
   --delete-excluded --numeric-ids . /" ssh-dss
   AABAB3B3NzaC1kc3...und der Rest des Keys
```

Das Rsync-Kommando muss im Schlüssel exakt hinterlegt sein. Wer Rsync im Listing 1 modifiziert hat, muss die Änderung daher auch ins Listing 3 übernehmen. Erweist sich das unter Umständen als zu schwierig, führt der Admin einfach das Skript einmal manuell mit Passwortabfrage aus und holt sich das genaue Kommando mit »ps axw | grep rsync« aus der Prozessliste des gerade sichernden Hosts.

Schnappschüsse per Hardlinks

Bisher kann die Lösung bequem und schnell 1:1-Kopien jedes Hosts anfertigen. Ziel ist es aber, tageweise Snapshots anzufertigen, die bei Bedarf selektiv wiederherstellbar sind. Das Skript aus Listing 4 ist dafür zuständig. Das Kommando »cp -al« kopiert das Verzeichnis »daily.0« nach »daily.1« – jedoch nicht physikalisch, sondern durch Hardlinks. Jede Datei erhält dabei einen entsprechenden neuen Eintrag im Dateisystem unterhalb von »daily.1«, was keinen erwähnenswerten zusätzlichen Plattenplatz belegt. Das Skript verschiebt auch die weiteren Daily-Verzeichnisse und löscht die jeweils älteste Variante.

Spannend wird es dann beim allnächtlichen Rsync in Listing 1: Unveränderte Dateien überspringt Rsync erwartungsgemäß. Bei einer modifizierten Datei überträgt Rsync die neue Version übers

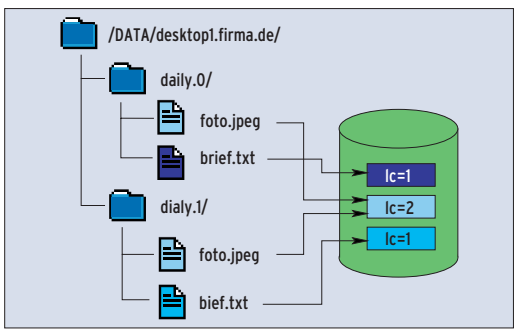

Abbildung 2: Die Schnapshots speichern unveränderte Dateien nur ein Mal, Rsync setzt nur den Link Count (Lc) um eins hoch. Auf dem Client geänderte Daten speichert der Backup-Server dagegen neu.

Netz und speichert sie zwischen. Dann löscht es den Hardlink auf die Datei unterhalb von »daily.0«, sodass sie nur noch unterhalb von »daily.1« liegt. Dann kopiert es die neue Datei unter »daily.0«.

Platz sparend

Erst ab diesem Zeitpunkt wird physikalisch mehr Festplattenplatz belegt: Unter »daily.0« ist die aktuelle Fassung zu finden, unter »daily.1« jene vom Vortage – das Snapshot-Problem ist damit gelöst (siehe Abbildung 2).

Unveränderte Dateien sind also unter beiden Verzeichnissen verlinkt, ohne doppelten Speicherplatz zu belegen. Die Lösung verbindet die Vorteile von Vollbackups und inkrementellen: Unterhalb der Daily-Verzeichnisse existiert von jedem Tag ein sofort einspielbares Vollbackup, gleichzeitig wird nicht mehr Speicherplatz als bei inkrementellen Backups benötigt, da nur geänderte Dateien mehrmals Platz auf dem Backup-Medium blocken – perfekt.

Nicht einfach: Konsistente Datenbestände

Eine Tatsache verdient aber noch Aufmerksamkeit: Das Abbild der Client-Festplatte wird brutal zu einem bestimmten Zeitpunkt gezogen. Ungeprüft dabei bleibt, ob das Abbild in sich konsistent ist. Ändern sich während des

Listing 4: »backup-rotate«

```
01 #!/bin/bash
02 #
03 # Das Skript rotiert Backup-Verzeichnisse
04 # http://www.heinlein-partner.de
05 #
06 # Aufruf: backup-rotate <FQDN-Servername>
07 #
08
09 # ### Aufrufparameter des Skripts ist ein FQDN-Hostname
10 if [ -n "$1" ] ; then
11         SERVER="$1"
12 else
13         echo "Error: Usage $0 <fqdn-hostname>"
14         exit
15 fi
16
17 # ### Konfiguration
18 # Pruefen, ob noch ein gewisser Prozentsatz
19 # an Plattenplatz und Inodes frei ist?
20 CHECK_HDMINFREE=true
21 HDMINFREE=90
22
23 # Soll die Daten-Partition readonly gemountet werden,
24 # wenn sie nicht in Gebrauch ist?
25 MOUNT_RO=false
26 MOUNT_DEVICE=/dev/hdc1
27
28 # Unter welchem Pfad wird gesichert?
29 DATA_PATH=/DATA
30
31 # ### Let`s Rock`n`Roll....
32
33 # Pruefe auf freien Plattenplatz
34 GETPERCENTAGE='s/.* \([0-9]\{1,3\}\)%.*/\1/'
35 if $CHECK_HDMINFREE ; then
36         KBISFREE=`df /$DATA_PATH | tail -n1 | sed -e "$GETPERCENTAGE"`
37         INODEISFREE=`df -i /$DATA_PATH | tail -n1 | sed -e
   "$GETPERCENTAGE"`
38         if [ $KBISFREE -ge $HDMINFREE -o $INODEISFREE -ge $HDMINFREE ]
   ; then
39             echo "Fatal: Not enough space left for rotating backups!"
40             logger "Fatal: Not enough space left for rotating backups!"
41             exit
42         fi
43 fi
44
45 # Festplatte rw remounten falls gewuenscht!
46 if $MOUNT_RO ; then
47         if `mount -o remount,rw $MOUNT_DEVICE $DATA_PATH` ; then
48               echo "Error: Could not remount $MOUNT_DEV readwrite"
49               logger "Error: Could not remount $MOUNT_DEV readwrite"
50               exit
51         fi
52 fi
53
54 # Gegebenenfalls Verzeichnis anlegen
55 if ! [ -d $DATA_PATH/$SERVER/daily.0 ] ; then
56         mkdir -p $DATA_PATH/$SERVER/daily.0
57 fi
58
59 echo "Rotating snapshots of $SERVER..."
60 logger "Rotating snapshots of $SERVER..."
61
62 # Das hoechste Snapshot abloeschen
63 if [ -d $DATA_PATH/$SERVER/daily.7 ] ; then
64         rm -rf $DATA_PATH/$SERVER/daily.7
65 fi
66
67 # Alle anderen Snapshots eine Nummer nach oben verschieben
68 for OLD in 6 5 4 3 2 1  ; do
69         if [ -d $DATA_PATH/$SERVER/daily.$OLD ] ; then
70               NEW=$[ $OLD + 1 ]
71               # Datum sichern
72               touch $DATA_PATH/.timestamp -r
   $DATA_PATH/$SERVER/daily.$OLD
73               mv $DATA_PATH/$SERVER/daily.$OLD
   $DATA_PATH/$SERVER/daily.$NEW
74               # Datum zurueckspielen
75               touch $DATA_PATH/$SERVER/daily.$NEW -r
   $DATA_PATH/.timestamp
76         fi
77 done
78
79 # Snapshot von Level-0 per hardlink-Kopie nach Level-1 kopieren
80 if [ -d $DATA_PATH/$SERVER/daily.0 ] ; then
81         cp -al $DATA_PATH/$SERVER/daily.0 $DATA_PATH/$SERVER/daily.1
82 fi
83
84 # Festplatte ro remounten falls gewuenscht!
85 if $MOUNT_RO ; then
86         if `mount -o remount,ro $MOUNT_DEVICE $DATA_PATH` ; then
87              echo "Error: Could not remount $MOUNT_DEV readonly"
88              logger "Error: Could not remount $MOUNT_DEV readonly"
89              exit
90         fi
91 fi
```

Backup-Laufs Dateien auf dem Host, gelangen sie zuweilen ins Backup, teilweise nicht mehr. Highend-Festplattensysteme (und einige experimentelle Linux-Dateisysteme) lassen sich binnen Sekunden einfrieren und anschließend in Ruhe sichern. Diese Möglichkeit fehlt in der Praxis zumeist.

Problematisch wird es, wenn beispielsweise eine MySQL-Datenbank, sie besteht in der Regel aus jeweils drei Dateien pro Tabelle, ausgerechnet dann gesichert wird, wenn sie der Client gerade beschreibt. Im günstigsten Fall ist sie dann inkonsistent, aber reparierbar, im ungünstigsten Fall zerstört. Das Problem hat natürlich nicht nur die vorgestellte Lösung, sondern alle Programme, die auf Dateiebene sichern. Die Praxis zeigt, dass die Gefahr solcher Inkonsistenzen gering ist, solange das Backup zu einem Zeitpunkt erfolgt, wenn normalerweise kein Betrieb herrscht.

Eleganter gerade bei Datenbanken ist es, sie per Cronjob rechtzeitig vor dem Backup-Lauf (bei MySQL über den Befehl »mysqldump«) in eine Datei zu befördern [2]. Da der Dump über das Datenbankprogramm erfolgt, ist er garantiert konsistent und wird zusätzlicher Bestandteil im nächtlichen Backup-Lauf. Platz sparend lässt sich der Dump per Gzip packen:

```
/usr/bin/mysqldump --all-databases |➥
 /usr/bin/gzip >➥
 /var/lib/mysql/fulldump.sql.gz
```

Ab jetzt automatisch

Funktioniert im manuellen Testbetrieb alles zur vollen Zufriedenheit, trägt der Admin die Skripte als Cronjobs ein. Dabei muss er jeweils den vollen Namen des jeweiligen Servers als Aufrufparameter angeben. Zu beachten ist, dass zueinander gehörende Jobs sich zeitlich nicht überlappen – auf die Absicherung per Lock-Datei wurde aus Platzgründen verzichtet.

Trennung auf Zeit

Prinzipiell bestünde auch die Möglichkeit, beide Skripte zusammenzuwerfen. Doch die Nacht ist kurz und wenn viele Hosts zu sichern sind, wird der Admin die wertvolle Zeit nicht mit dem Rotieren der Snapshots vergeuden wollen. Besser ist es, ein weiteres Skript »do_backup« anzulegen, das der Reihe nach alle Backups durchführt und anschließend in Ruhe rotieren lässt.

Dieses Skript kann bequem durch einen einmaligen Eintrag als nächtlicher Termin in die Crontab aufgenommen werden. Gleichzeitig lohnt es sich, den Output eines jeden Skripts über eine Ausgabeumleitung in ein Logfile zu schreiben, um regelmäßig kontrollieren zu können, ob alles fehlerfrei funktioniert. Listing 5 zeigt ein Beispiel.

Manchmal ist es wünschenswert, wenn nicht nur die letzten Tage per Snapshot rekonstruierbar wären, sondern mehrere Monate. Mit wenigen Handgriffen ist das »backup-rotate«-Skript in ein anderes namens »backup-rotate-monthly« kopiert und angepasst: Dann wird »daily.0« zu »monthly.0« und »monthly.1«, »monthly.2«, »monthly.3« et cetera beginnen zu rotieren. Einmal im Monat per Cronjob aufrufen, das löst die Aufgabe – zusätzlich zu den Tages-Snapshots.

Hoffentlich nie nötig: Daten wieder herstellen

Leidet irgendwann mal eine Client-Festplatte unter Amnesie, spielt der Administrator die passenden Daten manuell vom Backup-Server per Rsync auf den betroffenen Host zurück. Wenn nicht sowieso schon zerstört, muss er dazu den in »authorized_keys« hinterlegten SSH-Key deaktivieren, damit er ein abweichendes Kommando absetzen kann. Ist die Platte total hinüber, hilft eine Netzwerk-taugliche Rettungs- oder Knoppix-Boot-CD. Müssen sehr viele GByte den Rechner wechseln, kann der vorübergehende Umbau der Backup-Platte in den Hosts den (dann lokal ablaufenden) Kopiervorgang beschleunigen.

Beobachtung erforderlich

Wie alles andere will auch der dauerhafte Betrieb dieser Lösung regelmäßig überwacht sein. Etwaige Fehler, wie eine unbedachte Änderung von Firewall-Regeln oder eine übervolle Festplatte, bleiben dann nicht über Wochen hinweg unbemerkt. Auch der Standort des Backup-Servers sollte wohl überlegt sein: ein anderen Raum, besser ein anderes Gebäude als für die zu sichernden Rechner. Das schafft bereits eine gute Absicherung, falls der Server-Raum beschädigt oder zerstört werden sollte.

Auf [3] ist die vorgestellte Lösung perfektioniert mit weiteren Absicherungsmaßnahmen dokumentiert. Weiteren Lesestoff liefert das Backup-Buch von Wolfgang Barth [4]. *(jk)*

Listing 5: »do_backup« für die Crontab

```
01 # Die Nacht beginnt mit dem Sichern alle Daten
02 /usr/local/bin/backup-rsync www.firma.de > /DATA/www.firma.de-log
03 /usr/local/bin/backup-rsync mail.firma.de > /DATA/mail.firma.de-log
04 /usr/local/bin/backup-rsync intra.firma.de > /DATA/intra.firma.de-log
05
06 # Anschließend kann in Ruhe rotiert werden
07 /usr/local/bin/backup-rotate www.firma.de >> /DATA/www.firma.de-log
08 /usr/local/bin/backup-rotate mail.firma.de >> /DATA/mail.firma.de-log
09 /usr/local/bin/backup-rotate intra.firma.de >> /DATA/intra.firma.de-log
```

Infos

[1] C. Dittmann, „Server schnell und kostengünstig spiegeln“: Linux-Magazin 04/02, S. 99
[2] Thomas Wölfer, „MySQL-Datenbanken sichern – Teil 1“: Linux-Magazin 05/04, S. 60
[3] Ausführlichere Skripts des Autors: [http://www.heinlein-partner.de/web/projekte/rsync-backup/]
[4] Wolfgang Barth, „Datensicherung unter Linux“: [http://www.opensourcepress.de]

Der Autor

Peer Heinlein betreibt seit 1992 einen Internet Service Provider und hat neben dem „Postfix-Buch“ bei Open Source Press noch zwei weitere Bücher zu LPIC-1 und zum Einbruchserkennungssystem Snort veröffentlicht. Er ist regelmäßig mit verschiedenen Vorträgen auf Linux-Tagen vertreten und gibt fortlaufend Schulungen und Weiterbildungen für Administratoren. [http://www.heinlein-partner.de]