Sicherung verteilter Systeme mit Bacula

Stefan Schumacher
Stefan.Schumacher@Kaishakunin.com

KAISHAKUNIN.COM
IT-Sicherheitsberatung

Veröffentlicht in der GUUG UpTimes Dezember 2006.

Literaturverzeichnis

Schumacher, Stefan (2006). „Sicherung verteilter Systeme mit Bacula“. Deutsch. In: *UpTimes* 4, Seiten 14 – 20. ISSN: 1860-7683. URL: http://kaishakunin.com/publ/guug-uptimes-bacula.pdf (besucht am 07. 11. 2009).

```
@article{Schumacher-ut-bacula,
    author = {Stefan Schumacher},
    title = {Sicherung verteilter Systeme mit Bacula},
    year = {2006},
    volume="4",
    publisher = {German Unix User Group},
    journaltitle="UpTimes",
    issn="1860-7683",
    language = {german},
    url={http://kaishakunin.com/publ/guug-uptimes-bacula.pdf},
    urldate="2009-11-07",
    pages="14\,--\,20",
}
```

Sicherung verteilter Systeme mit Bacula

Stefan Schumacher

Dieser Artikel stellt Bacula vor, ein Sicherungssystem für verteilte heterogene Systeme. Es arbeitet Client/Server-basiert und verwendet eine Datenbank als Katalog, daher ist es hervorragend geeignet, um komplexe Sicherungen durchzuführen. Bacula wird unter der GPL vertrieben, läuft auf fast jedem Unix-artigem Betriebssystem und bietet auch einen Client für MS-Windows an.

Bacula

Heterogene verteilte Systeme zu sichern erfordert in der Regel mehr Aufwand für die Konfiguration der Sicherung als für die eigentliche Sicherung der Dateien. Hier stoßen kleinere Backupsysteme oder selbstgemachte Shellskript-Lösungen mit Dump oder Rsync schnell an ihre Grenzen. Besonders delikat ist in solchen Systemen auch die Verwaltung der Sicherungsmedien, weiß doch der geneigte Systemadministrator, dass der GAU immer in seiner Freizeit eintreten wird. Und wer will schon den Sonntag damit verbringen, im Keller unzählige Dump-Bänder auf der Suche nach Frau Müllers `Kaffekasse.xls` durchzuspulen?

Ab einem gewissen Rechner-Umfang müssen spezielle Sicherungssysteme für große Netzwerke eingesetzt werden. Bisher gab es diese nur für teures Geld von kommerziellen Anbietern. Amanda, die einzige freie Lösung für komplexere Situationen, stößt wegen der zugrundeliegenden Prinzipien schnell an gewisse Grenzen.

Seit 2002 existiert mit Bacula ein System zur Datensicherung, das speziell für verteilte Systeme entwickelt wurde. Es wird unter der GPL/LGPL vertrieben und ist somit frei einsetzbar. Im Laufe der Zeit hat Bacula einen Funktionsumfang gewonnen, der es mit den meisten kommerziellen Systemen problemlos aufnehmen kann.

Der Bacula-Server kann unter Linux, Solaris, NetBSD oder FreeBSD betrieben werden, der notwendige Client neben o.g. Systemen auch auf weiteren Unix-artigen, wie HP/UX, AIX, IRIX oder MacOS X/Darwin. Exotischere Betriebssysteme können notfalls ihre Verzeichnisse per NFS o. ä. auf Hosts exportieren, die über einen Bacula-Client verfügen und dort ihre Daten sichern lassen. Eine fertige Binärdatei des Clients existiert auch für Windows, so das diese Rechner gesichert werden können. Ich beschreibe in diesem Artikel die Einrichtung auf einem NetBSD-Server und Client mit einsatzbereitem Paketverwaltungssystem pkgsrc [8].

Der Aufbau

Bacula besteht aus verschiedenen Daemons, die relativ einfach den Client-Server-Betrieb ermöglichen:

- Director (bacula-dir)
- Storage (bacula-sd)
- File (bacula-fd)
- Catalog (PostgreSQL)
- Console (bconsole)

Der „Director“ läuft als Steuerungsprozess im Hintergrund und koordiniert alle Sicherungsaktivitäten. Er wird eingesetzt, um Daten zu sichern und wiederherzustellen und um Aufgaben zu planen.

Der „Storage“-Daemon übernimmt das Schreiben der Daten auf die Sicherungsmedien und ggf. das Auslesen der Sicherung. Die Medien können Bänder, Wechselfestplatten, CDs, DVDs oder Disketten sein.

Der „File“-Daemon liefert die Daten des zu sichernden Rechners auf Befehl des Directors an den Storage-Daemon. Der File-Daemon ist betriebssystemabhängig. Für NetBSD kann er via pkgsrc installiert werden, für andere Unix-artige Betriebssysteme steht er ebenfalls zur Verfügung, meist schon in den jeweiligen Paketverwaltungssystemen.

Der „Catalog“ speichert alle Metadaten zu den Sicherungsläufen, also Daten wie: „welche Datei wurde in welcher Version wann auf welches Band gesichert und hat dabei folgende Prüfsumme?“. Mit ihm ist es möglich, die Sicherungsbänder zu organisieren und ggf. für eine Rücksicherung das gewünschte Band bequem zu finden.

Da hierzu mehr als eine einfache Textdatei notwendig ist, kann sich Bacula an SQLite, PostgreSQL oder MySQL binden. Ich beschränke mich hier auf PostgreSQL, da ich selber einige PostgreSQL-Server betreue. Hat man keinerlei Ambitionen selbst per SQL in der Index-Datenbank zu suchen oder überhaupt noch keine Datenbanken im Einsatz, empfiehlt es sich, als Index SQLite zu verwenden.

Der Katalog ist wichtig, wenn man Dateien zur Rücksicherung suchen will. Daher sollte er zwingend gesichert werden, das Bacula-Handbuch [3] zeigt, wie man den Katalog exportiert und für eine Notfall-CD aufbereitet. Außerdem kann man natürlich PostgreSQL-eigene Funktionen verwenden, um den Katalog zu sichern oder per Replikation auf mehreren Rechnern vorhalten.

Die „Console“ ermöglicht den Zugriff auf die Bacula-Dienste. Neben einer einfachen Shell-basierten Textkonsole existieren auch klicki-bunte Varianten für Gnome, Java oder wxWidgets.

Alle Komponenten kommunizieren über das Netz miteinander und können so auf verteilten Systemen eingesetzt

werden. Zur Authentifizierung verwenden sie CRAM-MD5 und zur Verschlüsselung kann TLS eingesetzt werden. Bacula verwendet wohldefiniert nur die Ports 9101 bis 9103, lässt sich also sehr pflegeleicht durch Firewalls schleusen.

Hardwareunterstützung

Neben der Betriebssystemunterstützung ist die Hardwareunterstützung ein wichtiges Einsatzkriterium. Neben CD/DVD-Brennern unterstützt Bacula auch die Sicherung in einfache Dateien, die dann bspw. per SCP oder NFS auf entfernte Systeme oder eine USB-Wechselplatte kopiert werden können. Im professionellen Bereich sind Bandlaufwerke und Bandwechsler weit verbreitet. Bacula unterstützt alle Laufwerke und Wechsler, die in den Kompatibilitätslisten [6], [7] angegeben sind. Zusätzlich zu den Bacula-Listen gibt auch die Kompatibilitätsliste des Linux-MTX-Band-Treibers ([5]) Auskunft zu unterstützten Geräten.

Normalerweise unterstützt Bacula alle Geräte, die auch vom Betriebssystem angesteuert werden können. Möchte man auf Nummer sicher gehen, kann man mit mt(1) die Bandansteuerung testen und [3, Kapitel „Testing Your Tape Drive With Bacula„] durcharbeiten.

Bandrotation und Archivmedien („Freitagsband“) können mit mehreren sog. Pools realisiert werden. Die Vorgehensweise wird im Abschnitt zur Konfiguration beschrieben.

Existiert bereits Hardware, die nicht von Bacula unterstützt wird und nicht ersetzt werden kann, kann man als Workaround die Sicherung von Bacula in eine Datei schreiben lassen, die dann mit anderen Programmen (dump(8), dd(1)) auf das Bandlaufwerk gesichert wird. Man muss dann aber die Verwaltung der Bänder händisch organisieren und im Falle einer Rücksicherung die Sicherungsdatei für Bacula vom Band wieder auf Platte spielen.

Installation

Bacula ist als pkgsrc/sysutils/bacula in NetBSDs Paketverwaltung pkgsrc erhältlich. Ein Windows-Client existiert ebenfalls, er ist auf der Bacula-Webseite erhältlich und wird wie ein Unix-Bacula-File-Daemon in einer Textdatei oder via Bacula-Console konfiguriert.

Bacula wird standardmäßig mit SQLite als Katalog kompiliert. Um PostgreSQL einzubinden, muss man die Paketoptionen in /etc/mk.conf auf `PKG_OPTIONS.bacula= -catalog-sqlite catalog-pgsql` setzen. Mit dem üblichen `make install clean` wird Bacula nun installiert. Für den Einsatz auf Clients kann man pkgsrc/sysutils/bacula-clientonly verwenden.

PostgreSQL kann mit den Skripten in /usr/pkg/libexec/bacula/ (siehe Abb. 1) eingerichtet, aktualisiert oder wieder von Bacula gesäubert werden. Voraussetzung ist hier, das ein PostgreSQL-Cluster schon existiert, PostgreSQL also neue Datenbanken anlegen kann. Im selben Verzeichnis befinden sich auch die Skripte, um SQLite oder MySQL einzurichten.

Konfiguration

Für Baculas Daemons existieren insgesamt drei rc-Skripte, die man nach /etc/rc.d kopieren und in /etc/rc.conf starten kann, siehe Abb. .

Sind alle notwendigen Prozesse gestartet, kann man mit der bconsole Verbindung zum Server aufnehmen. Allerdings sucht bconsole standardmäßig im aktuellen Verzeichnis nach der Konfigurationsdatei bconsole.conf. Eine Beispieldatei befindet sich in `/usr/pkg/share/examples/bacula/bconsole.conf`, so dass man die Datei bspw. ins Homeverzeichnis kopieren und anpassen kann oder einfach ein passendes Alias in der Shell erzeugt.

Man kann in der bconsole Sicherungsläufe konfigurieren, starten, stoppen oder auch überwachen. Oder man verwendet vi(1) um die .conf-Dateien anzupassen. Standardmäßig hat Bacula bereits zwei „FileSets“ definiert, nämlich *Full Set*, das das Quellverzeichnis von Bacula sichert, und *Catalog*, das den Index von Bacula sichert. Eine kurze Sitzung mit der bconsole zeigt Abb. 3.

Die Sicherungsläufe und zu verwendenden Geräte sowie Hosts werden in den drei .conf-Dateien bacula-dir.conf, bacula-fd.conf und bacula-sd.conf konfiguriert.

Für jeden der Daemons muss festgelegt werden, welcher Director sich mit ihnen verbinden darf. Dies geschieht mit der `Director`-Direktive:

```
# Dieser Director darf sich
# an den Daemon binden
Director {
  Name = fenris-dir
  Password = "..."
  Description = "Bacula auf
          fenris.net-tex.de"
}

# eingeschränkt für
# den Monitor
Director {
  Name = fenris-mon
  Password = "..."
  Monitor = yes
}
```

Am einfachsten lässt sich der File-Daemon anpassen, allerdings muss dies auch auf jedem zu sichernden Rechner geschehen:

```
FileDaemon {
Name = fenris-fd

# Port des Directors
FDport = 9102

WorkingDirectory = /var/ \
           spool/bacula
Pid Directory = /var/run
Maximum Concurrent Jobs = 20
}
```

Die Sicherungsgeräte (Streamer, Wechselplatten etc.) steuert der Storage Daemon an, daher werden sie in bacula-sd.conf eingerichtet:

```
Device {

# logischer Gerätename
Name = FileStorage

# Medientyp, hier Datei
Media Type = File

# physikalischer
# Gerätenename
Archive Device = \
  /usr/bacula/


# Medien automatisch labeln
LabelMedia = yes;

# Random-Access-Medientyp
```

```
# su - pgsql
pgsql$ cd /usr/pkg/libexec/bacula/
pgsql$ ./create_postgresql_database
pgsql$ ./make_postgresql_tables
pgsql$ ./grant_postgresql_privileges
 [...]
CREATE USER
pgsql$ psql -U bacula bacula
Welcome to psql 8.0.4, the PostgreSQL interactive terminal
 [...]
bacula=#
```

Abbildung 1: PostgreSQL für Bacula einrichten

```
# cp /usr/pkg/share/examples/rc.d/bacula* /etc/rc.d/
# echo 'bacula=yes' >> /etc/rc.conf
# echo 'baculadir=yes' >> /etc/rc.conf
# echo 'baculasd=yes' >> /etc/rc.conf
# echo 'baculafd=yes' >> /etc/rc.conf
```

Abbildung 2: Bacula per rc-Skript starten

```
$ bconsole -c /usr/pkg/share/examples/bacula/bconsole.conf
Connecting to Director balmung:9101
1000 OK: balmung-dir Version: 1.38.2 (20 November 2005)
Enter a period to cancel a command.
*time
17-Jan-2006 21:55:38
*version
fenris-dir Version: 1.38.2 (20 November 2005)
*show filesets
FileSet: name=Full Set
      O M
      N
      I /usr/pkgsrc/sysutils/bacula/work/bacula-1.38.2
      N
      E /proc
      E /tmp
      E /.journal
      E /.fsck
      N
FileSet: name=Catalog
      O M
      N
      I /var/spool/bacula/bacula.sql
```

Abbildung 3: Baculas bconsole

```
# d.h. der Medientyp
#  unterstützt
# lseek bzw. lseek64
Random Access = Yes;


# Wechselmedium
RemovableMedia = no;

AutomaticMount = yes;
AlwaysOpen = no;
}
```

```
Device{
Name = DDS4
Description = "DDS-4 \
Streamer"
Media Type = DDS-4
Archive Device = \
/dev/nrst0
AutomaticMount = yes;
AlwaysOpen = yes
Offline On Unmount = no
Hardware End of Medium =\
no
BSF at EOM = yes
Backward Space Record = \
no
Fast Forward Space File \
= no
TWO EOF = yes
}
```

Die eigentliche Konfiguration zur Sicherung wird im Director, in der Datei `/usr/pkg/etc/bacula/bacula-dir.conf` vorgenommen.

Im Fileset werden mit der Include- und Exclude-Direktive die zu sichernden bzw. auszuschließenden Pfade festgelegt. Innerhalb einer Include-Direktive werden in der Options-Anweisung die Optionen für die angegebenen Pfade festgelegt. Besonders interessant sind folgende Optionen:

signature = [MD5 |SHA1] erzeugt eine Prüfsumme für jede gesicherte Datei

compression = GZIP komprimiert jede Datei mit GZIP

verify = Optionen für einen Verifikationslauf, die Bacula verwendet, wenn die Sicherung mit dem Quellsystem verglichen wird. Im Bsp. sind dies: *p*=Permission, *i*=Inodes, *n*=Number of Links, *s*=Size, *5*=MD5-Hash

onefs = yes |no bei No werden auch gemountete Dateisysteme mitgesichert, bei Yes nur das angegebene

sparse = yes |no zur besseren Behandlung von Sparsefiles, also Dateien mit Löchern

recurse = yes |no gibt an, ob in Unterverzeichnisse abgestiegen werden soll; Standard ist yes

portable = yes |no sichert Win32-Dateien in einem portablen Format, das auch unter Unix entpackt werden kann; Standard ist no

```
FileSet{
 Name = "home"

 # kapselt
 # einzuschließende Pfade
 Include{
  File = /home
   # Optionen für og. \
    Pfade
   Options{
 signature = MD5
 verify = pins5

     compression = GZIP
 onefs = no
 sparse = no
   }
 }
 Include{
  File = /postgresql
  Options{
   signature = MD5
   verify = pins5
   sparse = yes
   onefs = yes
  }
 }
 # auszuschließende Pfade
 Exclude{
  File = /home/stefan/\
  *.bak
  File = /home/stefan/\
  temp
  File = /var/tmp
 }
}
```

Im Schedule können verschiedene Zeitpläne für die Sicherungsläufe erstellt werden. Dabei verwendet Bacula die Optionen Full, Differential und Incremental, um voll, differentiell oder inkrementell zu sichern. Im Konfigurationsbeispiel sind zwei Zyklen angegeben: „Archivband“ sichert jeden Sonntag um 21:00 komplett, „Wochenzyklus“ erzeugt am ersten Montag des Monats eine Komplettsicherung, am zweiten bis fünften eine differentielle Sicherung (also alles, was seit dem ersten Montag geändert oder erzeugt wurde). Dienstags bis freitags wird dann inkrementell gesichert.

```
Schedule {
 Name = "Archivband"
 Run = Full Sun at 21:00
}

Schedule{
 Name = "Wochenzyklus"
 Run = Full 1st Mon \
       at 06:00
 Run = Differential \
       2nd-5th Mon \
       at 06:00
 Run = Incremental \
       Tue-Fri at 06:00
}
```

Schlussendlich werden im Job die Konfigurationsangaben zu einem Sicherungslauf zusammgefasst. Da es oft mehrere Jobs geben kann, die sich nur geringfügig unterscheiden, z. B. im Hostnamen, können beliebige Optionen einer Job-Definition in sog. JobDefs ausgelagert werden. Diese JobDefs können dann in die eigentlichen Job-Konfigurationen eingelesen werden.

Wichtige Konfigurationsparameter sind unter anderem:

Type = *Typ* Art des Jobs; entweder eine Sicherung, eine Rücksicherung, ein Verifikationslauf (Vergleich der Quelle mit der Sicherung) oder ein Administrationslauf (z. B. Bereinigung des Katalogs oder Rückspulen und Auswerfen der Bänder)

Level = *Level* Art der Sicherung; entweder *Full*, *Incremental* oder *Differential*. Für einen Verify-Job existieren eigene Level, die im Handbuch [3] beschrieben sind.

Pool = *Name* gibt den zu verwenden Pool an. Mit mehreren Pools können mehrere Medien (z. B. Tagesbänder) verwaltet werden. Es existieren weitere Optionen, um Pools für verschiedene Sicherungstypen zu definieren.

Recycle = yes — no gibt an, ob ein existierendes Volume überschrieben werden darf

VolumeRetention = *Zeit* Aufbewahrungsdauer des Volumes

Schedule = *Name* gibt den zu verwendenden Zeitplan an

Storage = *Name* gibt den zu verwendenden Storage-Daemon an. Somit kann man verschiedene Sicherungsgeräte verwenden.

Client = *Name* Name des zu sichernden Rechners

FileSet = *Name* Name des zu sichernden FileSets

RerunFailedLevels = yes |no wenn vorhergehende, höherlevelige Sicherungsläufe fehlgeschlagen sind, wird das Level des jetzigen Laufes erhöht. Nützlich für Laptops, die nicht immer im Netz sind und bspw. bei der letzten Komplettsicherung übergangen wurden.

RunBeforeJob = *Befehl* führt *Befehl* vor dem Sicherungslauf aus. Nützlich für Aufräumarbeiten oder um bspw. eine Datenbank in eine zu sichernde Datei zu exportieren.

RunAfterJob = *Befehl* führt *Befehl* nach dem Sicherungslauf aus, z. B. Auswurf der Bänder

```
JobDefs {
  Name = "sicherung"
  Client = fafnir-fd
  Type = Backup
  Level = Full
  FileSet = "home"
  Messages = Standard
  Pool = Default
  Priority = 10
}

Job {
  Name = "Archivierung"
  JobDefs = "DefaultJob"
```

```
  Schedule = "Archivband"
  Storage = FileStorage
  RunBeforeJob = \
  "/root/bin/my_pg\
  _dumpall.sh"
  RunAfterJob = \
  "cp /usr/bacula \
      /vol* /mnt/nfs/\
      archiv"
}

Job {
  Name = "Client1"
  JobDefs = "DefaultJob"
  Schedule = "Wochen\
  zyklus"
  Storage = DDS4
```

Anstatt auf Bänder kann Bacula die Sicherungen auch als Datei auf eine Wechselplatte schreiben. Diese Methode ist recht kostengünstig und schnell, erfordert aber einigen Konfigurationsaufwand. Normalerweise labelt Bacula jedes Volume (also jedes Band oder hier – jede Datei) mit einem eigenen Label, das der Verwaltung dient. Möchte man nicht nur ein Volume verwenden, das immer wieder überschrieben wird, muss man verschiedene Pools definieren, in denen die Verwendung eines Volumes beschränkt wird. Abbildung 4 zeigt eine mögliche Konfiguration. Hierbei werden fünf verschiedene, gleichartige Pools, für jeden Werktag einer, erzeugt. Die Pools verwenden eine Wechselfestplatte als Medium, auf das sie zwei Volumes (*Maximum Volume Jobs = 2*) schreiben dürfen. Sind beide Volumes voll, wird das erste wieder überschrieben (*Recycle = yes*). Mit den RunBeforeJob- und RunAfterJob-Befehlen wird die Wechselfestplatte ein- bzw. ausgemountet und kann so nach der Sicherung entfernt werden. Im Schedule wird für jeden der täglichen Sicherungsläufe ein eigener Pool definiert, der die Pool-Option in der Job-Konfiguration überschreibt. Zusätzlich wird im Job noch eine *Max Start Delay* von 14 Stunden festgelegt, d.h. Bacula versucht bei Fehlern 14 Stunden lang den Lauf auszuführen. Somit hat der Administrator die Möglichkeit evtl. auftretende Fehler zu beheben. Weiterhin wird mit *Write Bootstrap* eine Bootstrap-Datei erzeugt und auf der jeweiligen Wechselplatte abgelegt. Diese Bootstrapdatei enthält alle notwendigen Metadaten zur Sicherung, ist also quasi nichts anderes als das passende Kapitel aus dem Katalog. Mit dieser Datei kann man die Sicherung auch von Hand zurückspielen.

Rücksicherung

Zur Rücksicherung verwendet Bacula ebenfalls wieder JobDefinitions, diese können aber manuell in der Konsole angepasst werden. In der Konsole gibt es den Befehl „restore“, dessen Ausgabe in Abbildung 5 gezeigt wird. Mit den vorgegebenen 12 Optionen lässt sich auf verschiedene Arten festlegen, was zurückgesichert werden soll.

1. Listet die letzten 20 Sicherungsläufe auf
2. Listet alle Sicherungsläufe auf, die eine bestimmte Datei enthalten
3. Übernimmt eine kommaseparierte Liste von Sicherungsläufen, die zurückgespielt werden sollen
4. Eine direkte SQL-Anfrage kann an PostgreSQL gestellt werden; die Ergebnismenge wird zurückgespielt
5. Der letzte Sicherungslauf eines bestimmten Rechners wird zurückgespielt
6. Der letzte Sicherungslauf eines bestimmten Rechners vor einem bestimmten Zeitpunkt wird zurückgespielt
7. Erwartet eine Liste mit Dateien, die zurückgespielt werden sollen
8. Erwartet eine Liste mit Dateien, deren letzte Version vor einem bestimmten Zeitpunkt zurückgespielt werden sollen
9. Sucht die ID für die neuste Sicherung eines Rechners
10. Sucht die ID für die neuste Sicherung eines Rechners vor einem bestimmten Zeitpunkt
11. Erwartet eine Liste mit Verzeichnissen, in die die gefundenen Jobs zurückgesichert werden sollen
12. Abbruch

Fazit

Bacula ist ein ausgereiftes System, um komplette heterogene Netzwerke zu sichern. Der Funktionsumfang braucht sich sicherlich nicht hinter kommerziellen Lösungen verstecken. Besteht Bedarf an weiteren Funktionen können diese entweder selbst oder über Finanzierung der Entwicklungsarbeiten erstellt werden. Die Hardwareunterstützung für Bandlaufwerke und -wechsler ist ausgereift, DVD-Brenner und Sicherungen auf Festplatten sind ebenso möglich.
Bacula ist naturgemäß ein äußerst komplexes System und erfordert als solches auch eine umfangreiche Konfiguration und daher Einarbeitungszeit. Dies ist aber bei allen Sicherungssystemen dieses Umfangs gegeben. Graphische Konfigurationstools existieren zwar, man sollte aber keine Scheu vor einem Texteditor und .conf-Dateien haben.

Hat man Bacula im Netzwerk eingerichtet, muss man zwingend entsprechende Testläufe starten und regelmäßige Inventuren und Testrestaurationen der Sicherungsmedien durchführen, da nicht einmal ein so komplexes System wie Bacula den reibungslosen Ablauf der Sicherungen über Jahre hinweg garantieren kann.

Bevor man sich ans Werk macht und die Sicherungsmethoden implementiert, muss man unbedingt die strategische und taktische Organisation der Backups durchplanen, da dies in derartigen Umgebungen gerne 75% des Arbeitsaufwandes ausmachen kann. Als Literatur empfiehlt sich hier das sehr gute [2], das zwar schon sechs Jahre alt ist, in den Grundlagen und der Theorie aber nichts an Aktualität und leider auch Brisanz eingebüßt hat.

Stefan Schumacher beschäftigt sich in seiner Freizeit mit japanischen Kampfkünsten sowie mit NetBSD und PostgreSQL. Seine persönliche Webseite ist unter www.net-tex.de erreichbar.

Literaturverzeichnis

[1] SCHUMACHER, Stefan: „Methoden zur Datensicherung“, `http://www.net-tex.de/backup.pdf`

[2] PRESTON, W. Curtis: „UNIX Backup and Recovery“, O’Reilly Media, 1999

[3] SIBBALD, Kern: „Bacula User’s Manual“, `http://www.bacula.org/rel-bacula.pdf`

[4] SIBBALD, Kern: „Bacula Developer’s Manual“, `http://www.bacula.org/bacula.pdf`

```
Schedule {
Name = "TaeglPlatte"
Run = Level=Full Pool=MontagPool Monday at 06:00
Run = Level=Full Pool=DienstagPool Tuesday at 06:00
Run = Level=Full Pool=MittwochPool Wednesday at 06:00
Run = Level=Full Pool=DonnerstagPool Thursday at 06:00
Run = Level=Full Pool=FreitagPool Friday at 06:00
}

Pool {
 Name = MontagPool
 Pool Type = Backup
 Recycle = yes
 AutoPrune = yes
 Volume Retention = 6d
 Accept Any Volume = yes
 Maximum Volume Jobs = 2
 RunBeforeJob = "/sbin/mount -o softdep,noatime /dev/sd4a /mnt/sd4a/"
 RunAfterJob = "/sbin/umount /dev/sd4a"
}
[...]
Pool {
 Name = FreitagPool
 Pool Type = Backup
 Recycle = yes
 AutoPrune = yes
 Volume Retention = 6d
 Accept Any Volume = yes
 Maximum Volume Jobs = 2
 RunBeforeJob = "/sbin/mount -o softdep,noatime /dev/sd4a /mnt/sd4a/"
 RunAfterJob = "/sbin/umount /dev/sd4a"
}

Job {
Name = "TaeglichKomplettAufPlatte"
Type = Backup
Client = ServerName
FileSet = "Full Set"
Schedule = "TaeglPlatte"
Media Type = File
Archive Device = /mnt/sd4a
LabelMedia = yes
Random Access = Yes
AutomaticMount = yes
RemovableMedia = no;
AlwaysOpen = no
Messages = Standard
Pool = Default
Write Bootstrap = "/mnt/sd4a/NightlySave.bsr"
Max Start Delay = 14h
}
}
```

Abbildung 4: Sicherung auf Festplatte

[5] „MTX compatibility list“, http://mtx.badtux.net/compatibility.php

[6] http://www.bacula.org/dev-manual/Supported_Tape_Drives.html

[7] http://www.bacula.org/dev-manual/Supported_Autochangers.html

[8] www.NetBSD.org/Documentation/pkgsrc/

```
*restore

First you select one or more JobIds that contain files
to be restored. You will be presented several methods
of specifying the JobIds. Then you will be allowed to
select which files from those JobIds are to be restored.

To select the JobIds, you have the following choices:
  1: List last 20 Jobs run
  2: List Jobs where a given File is saved
  3: Enter list of comma separated JobIds to select
  4: Enter SQL list command
  5: Select the most recent backup for a client
  6: Select backup for a client before a specified time
  7: Enter a list of files to restore
  8: Enter a list of files to restore before a specified time
  9: Find the JobIds of the most recent backup for a client
 10: Find the JobIds for a backup for a client before a specified time
 11: Enter a list of directories to restore for found JobIds
 12: Cancel
```

Abbildung 5: Restore-Optionen in der bconsole