

OpenZFS: Storage-Management für Einsteiger

19. Augsburger Linux-Infotag

Benedict Reuschling
benedict@reuschling.org

29.04.2023

Einführung in das ZFS Dateisystem

Dieser Foliensatz beschreibt das ZFS Dateisystem. Neben den zugrundeliegenden Ideen werden besonders die Eigenschaften, die für Nutzer des Dateisystems von Interesse sind, dargestellt. Es wird erläutert, wie man einen Storage-Pool anlegt und verwaltet, sowie ZFS-Parameter konfiguriert. Im einzelnen werden Komprimierung, Serialisierung der gespeicherten Pooldaten, Selbstheilung im Fehlerfall und Deduplizierung besprochen.

Die Beispiele sind so weit wie möglich unabhängig von bestimmten Distributionen. Bei Gerätebezeichnungen werden die unter FreeBSD gebräuchlichen Bezeichnungen verwendet. Diese lassen sich einfach durch ihre Entsprechung in der verwendeten Distribution ersetzen. Auf Besonderheiten wird entsprechend eingegangen. Da ZFS mehr Features beinhaltet als dieser Foliensatz fassen kann, wird versucht, auf die wichtigsten Themen für Anwender einzugehen. Hinweise auf weiterführende Informationen befinden sich am Ende.

Überblick

- 1 Einleitung
- 2 Probleme heutiger Dateisysteme
- 3 Eigenschaften von ZFS
 - Einfache Administration
 - Quota und Reservierung
 - Snapshots
 - Selbstheilende Daten
 - Komprimierung
 - Deduplizierung
 - ZFS Serialisierung

Einleitung

ZFS¹ ist ein von der Firma Sun (von Oracle übernommen) von Grund auf neu entwickeltes und als Open Source Software frei verfügbares Dateisystem mit eingebauten Volume Manager (für RAID-Verbünde). Es ist besonders für den Einsatz im Serverbereich gedacht, in der sehr grosse Datenmengen gespeichert werden müssen. Es lässt sich jedoch auch für den Heimgebrauch verwenden. Seine Features lassen sich jedoch auch für weniger anspruchsvolle Einsatzzwecke nutzen. Beim Design des Dateisystems standen besonders folgende Punkte im Vordergrund:

- Datenintegrität - erkennen und korrigieren von Datenfehlern
- Speicherkapazität - das erste 128-Bit Dateisystem
- Einfache Administration - 2 Befehle genügen zur Verwaltung
- Hohe Geschwindigkeit

¹früherer Name Zettabyte File System, heute einfach nur Z Filesystem

Probleme heutiger Dateisysteme

Dateisysteme sind im Vergleich zu der Entwicklung von immer leistungsfähigeren Festplatten (schnellerer Zugriff, höhere Kapazität) immer noch auf dem Stand von vor ca. 20 Jahren. Nicht zuletzt durch die Verfügbarkeit von SSD (Solid State Disks) müssen Dateisysteme von damals an die heutigen Speicherbedürfnisse angepasst werden, um den heutigen Anforderungen gerecht zu werden.

Ein paar Nachteile aktueller Dateisysteme sind hier aufgelistet:

Unbemerkte Datenfehler: Fehler können in vielen Bereichen der I/O-Kette auftreten, bevor die Daten sicher gespeichert werden. Diese werden meist nicht bemerkt, so dass die fehlerhaften Daten erst beim erneuten Aufruf bemerkt werden, wenn es meistens schon zu spät ist. Probleme können auftreten in Festplatten, Controllern, Kabeln, Treibern/Firmware oder RAM ohne ECC² und so das korrekte Speichern der Daten verhindern, ohne dass der Anwender bzw. das Betriebssystem dies mitbekommen.

²Error-Correcting Code

Probleme heutiger Dateisysteme

Schlechte Verwaltung: Viele Datenverluste werden unbeabsichtigt von Anwendern oder Administratoren verursacht. Dies liegt nicht zuletzt an der Komplexität der Datenverwaltung. Es müssen Label, Partitionen, Volumes, das hinzufügen weiteren Speicherplatzes und Konfigurationsdateien wie z.B. `/etc/fstab` richtig konfiguriert sein. Die Menge an Befehlen zu dieser Verwaltung trägt nicht gerade zur Vereinfachung bei und erhöht eher das Fehlerpotential.

Viele Beschränkungen: Volume- und Dateigrößen, Anzahl an Dateien, Dateien pro Verzeichnis und Anzahl Snapshots sind immer in der Anzahl/Grösse beschränkt.

Langsame Verarbeitung: Locking, feste Blockgrößen, langsame Schreibvorgänge von nicht zusammenhängenden Dateien bremsen weiterhin die I/O-Performance aus.

Gerade im Serverbereich spielen diese Faktoren eine wichtige Rolle, um die gespeicherten Daten schnell, sicher und effizient abzurufen.

Um die oben genannten Probleme zu beheben, wurde **ZFS** entwickelt.

Überblick

- 1 Einleitung
- 2 Probleme heutiger Dateisysteme
- 3 Eigenschaften von ZFS**
 - Einfache Administration
 - Quota und Reservierung
 - Snapshots
 - Selbstheilende Daten
 - Komprimierung
 - Deduplizierung
 - ZFS Serialisierung

Eigenschaften von ZFS

ZFS wurde von Grund auf neu entwickelt, um den heute gegebenen Anforderungen an Geschwindigkeit und Datensicherheit gerecht zu werden. Dazu wurde besonderer Wert auf die folgenden Eigenschaften gelegt:

- Poolbasierte Speicherung - man verwaltet einen Pool, auf dem die gesamte Speicherlogik und -verwaltung aufbaut
- Datenintegrität von Anfang bis Ende - Daten werden an jedem Punkt der Speicherkette (vom RAM bis zum Festspeicher) mittels Prüfsummen auf Konsistenz geprüft und ggfs. korrigiert
- Transaktionale Operationen - von Datenbanken bekanntes Modell der Speicherung ermöglicht nicht nur ständige Konsistenz, sondern auch hohe Gewinne in Sachen Geschwindigkeit
- Copy-on-write (COW) - bestehende Daten werden nie überschrieben, sondern neu abgelegt. Dadurch sind die Daten ständig konsistent und es wird auch kein `fsck` benötigt.

Anforderungen von ZFS

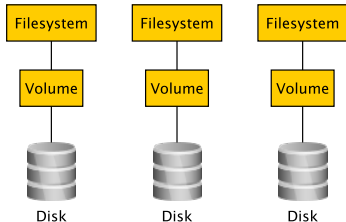
ZFS stellt etwas höhere Anforderungen an das System als andere Dateisysteme. Hauptspeicher ist für ZFS besonders wichtig, da viele Daten im RAM vorsortiert und verarbeitet werden, um dann geordnet auf die Platte geschrieben zu werden. Auf 32-Bit Systemen wird der Einsatz von ZFS nicht für den Produktiveinsatz empfohlen aufgrund der architekturbedingten Beschränkung dieser Systeme auf max. 4 GB RAM. Es sollte min. eine 64-Bit Architektur vorliegen und ≥ 4 GB RAM *nur für ZFS* reserviert werden. Die dadurch gewonnene Performance ist es jedoch wert, da das Dateisystem zur Verwaltung der Tertiärspeicher (Festplatten) eines Computers weiterhin das langsamste Medium darstellt.

Besonders speicherhungrig ist die Deduplizierung. Es wird empfohlen, für jedes TB an Speicher, das dedupliziert wird, > 2 GB RAM zu reservieren. Durch den Einsatz von SSDs als Schreib- bzw. Lese-Cache lassen sich ZFS Dateisysteme weiter beschleunigen (sog. Hybrid Storage Pool). Weiterhin ist ZFS *kein* Cluster- bzw. verteiltes Dateisystem und wurde auch nicht als solches entwickelt. Für diesen Einsatzzweck ist ZFS weniger geeignet.

Traditionelle Speicherarchitekturen im Vergleich zu ZFS

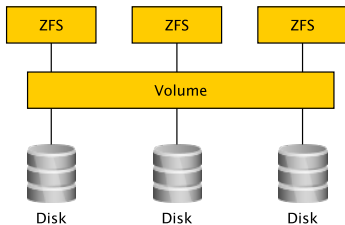
Traditionelle Architektur

- Partition(en)/Volume pro Filesystem
- Fragmentierte Kapazität und I/O Bandbreite
- Manuelles verkleinern und vergrössern des Speichers

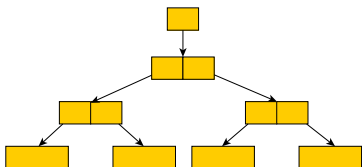


ZFS Architektur

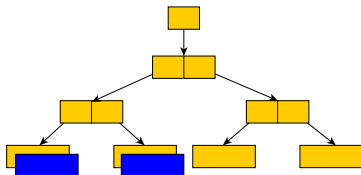
- Keine Partitionen notwendig
- Flexible Zuteilung des Speichers auf die Dateisysteme (Quota/Reservierung)
- Integrierte Fehlerbehandlung
- Snapshots, Klone, Komprimierung, Deduplizierung



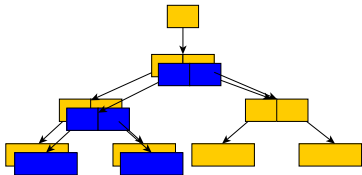
Copy on Write (COW) transaktionales Dateisystem



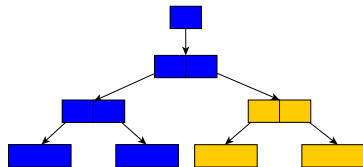
1. Konsistenter Zustand



2. Schreibvorgang an Dateien



3. Metadaten schreiben



4. Überblock schreiben (commit)

Überblick

- 1 Einleitung
- 2 Probleme heutiger Dateisysteme
- 3 Eigenschaften von ZFS
 - Einfache Administration
 - Quota und Reservierung
 - Snapshots
 - Selbstheilende Daten
 - Komprimierung
 - Deduplizierung
 - ZFS Serialisierung

ZFS on Linux installieren

Hinweise und aktuelle Versionen sind zu finden unter:

- <http://zfsonlinux.org> und
- http://open-zfs.org/wiki/Main_Page

Beispiel-Paketinstallation unter Ubuntu:

```
# apt install -y zfsutils-linux
```

Nach der erfolgreichen Installation stehen die im folgenden gezeigten ZFS-Kommandos zur Verfügung. Manche Befehle müssen mit `root` bzw. `sudo`-Rechten gestartet werden.

Einfache Administration

Um ZFS zu administrieren, muss man nur zwei Befehle und dessen Optionen kennen: `zpool` und `zfs`.

Durch `zpool` wird der Speicherpool und die daran angeschlossenen Geräte verwaltet. Mit `zfs` lassen sich die Parameter des Dateisystems anpassen. Eine Reihe von Unteroptionen bietet den Zugriff auf alle Features, von denen einige der wichtigsten im folgenden genauer dargestellt werden.

Einfache Administration - Erstellen eines Storagepools

Um einen Pool auf einer einzigen Platte anzulegen, wird folgender Befehl verwendet:

```
# zpool create tank disk1
```

Über die Bezeichnung `tank` (frei wählbar) wird der Pool zukünftig angesprochen und abstrahiert die im Pool befindlichen Geräte. Für `disk1` muss der Gerätenamen aus `/dev` verwendet werden, z.B. `sdb`.

Mit `zpool create` werden weitere Aktionen automatisch ausgeführt:

- Erzeugen eines Dateisystems mit dem gleichen Namen
- Einhängen des Dateisystems unter `/tank` (je nach Name)
- Die Einstellungen werden als Teil des Pools abgespeichert und werden nach einem Systemneustart genauso wieder angewendet
- Der Speicher steht sofort zur Verfügung

Somit entfallen (evtl. fehlerhafte) Einträge in `/etc/fstab`. Bei einem reinen ZFS-System (booten von ZFS), kann diese Datei komplett leer bleiben. Alle nötigen Konfigurationseinstellungen sind Teil des Pools.

Einfache Administration - Poolstatus anzeigen

Den Status des Pools kann man über das Kommando `zpool status` abfragen. Ein gesunder Pool sieht folgendermassen aus:

```
# zpool status
  pool: tank
  state: ONLINE
  scan: none requested
config:

  NAME                STATE          READ  WRITE CKSUM
  tank                ONLINE         0     0     0
    disk1             ONLINE         0     0     0

errors: No known data errors
```

Alle Pools werden durch den Befehl `zpool list` angezeigt.

```
# zpool list
NAME    SIZE  ALLOC   FREE      CAP  DEDUP  HEALTH  ALTROOT
tank   1016M  89.5K  1016M      0%  1.00x  ONLINE  -
```


Einfache Administration - I/O Statistiken anzeigen

ZFS besitzt ein eingebautes Werkzeug zur Anzeige von I/O-Statistiken (`iostat`) von Pools. Es zeigt den freien und belegten Speicher an, sowie die Grösse von Lese-/Schreiboperationen und deren I/O-Bandbreite.

```
$ zpool iostat
```

pool	capacity		operations		bandwidth	
	alloc	free	read	write	read	write
mypool	694M	1.31G	0	98	0	1.55M

Durch Angabe eines Intervalls in Sekunden als weiteren Parameter werden kontinuierlich diese Statistiken angezeigt, bis der Benutzer diesen Vorgang mit `Ctrl+C` abbricht.

Eine detailliertere Anzeige erlaubt die Option `-v` (verbose). Es wird dann der I/O für einzelne Geräte, die Bestandteil des Pools sind, aufgelistet.

Einfache Administration - RAID0 (Stripe) anlegen

Ein Pool mit nur einer Platte bietet weder ausreichend Redundanz, Kapazität noch angemessene Performance. Aus diesem Grund werden Stripes (grössere Kapazität und Geschwindigkeit, aber weiterhin **ohne** Redundanz) erstellt:

```
# zpool create mystripe disk1 disk2
```

```
# zpool status
  pool: mystripe
  state: ONLINE
  scan: none requested
  config:

    NAME            STATE          READ  WRITE CKSUM
    mystripe        ONLINE         0     0     0
      disk1         ONLINE         0     0     0
      disk2         ONLINE         0     0     0

  errors: No known data errors

# zpool list
NAME      SIZE  ALLOC   FREE   CAP  DEDUP  HEALTH  ALTROOT
mystripe 1.98G 92.5K  1.98G   0%  1.00x  ONLINE  -
```

Einfache Administration - Mirror (RAID1) anlegen

Gespiegelte Platten besitzen zwar weniger Gesamtkapazität, sichern aber Ausfallredundanz zu und können parallel gelesen werden.

```
# zpool create spiegel mirror disk1 disk2
```

Durch das Schlüsselwort `mirror` wird ZFS angewiesen, einen Spiegel aus den angegebenen Platten zu erstellen.

```
# zpool status
  pool: spiegel
  state: ONLINE
  scan: none requested
  config:

    NAME            STATE             READ WRITE CKSUM
    spiegel         ONLINE            0     0     0
      mirror-0     ONLINE            0     0     0
        disk1      ONLINE            0     0     0
        disk2      ONLINE            0     0     0

  errors: No known data errors

# zpool list
  NAME      SIZE  ALLOC   FREE      CAP  DEDUP  HEALTH  ALTROOT
  spiegel  1016M  92.5K  1016M     0%  1.00x  ONLINE  -
```

Einfache Administration - Mirror aus Pool erzeugen

Es kann ebenso aus einem bereits bestehenden Pool mit nur einem Gerät ein Spiegel erstellt werden. Dazu muss der Befehl `zpool attach` verwendet werden. Damit das neu aufgenommene Gerät die Daten des bestehenden Datenträgers spiegelt, muss der Pool zuerst “resilvert” werden. Bei diesem Vorgang synchronisiert der Pool beide Datenträger, so dass diese auf dem gleichen Stand sind. Während dieses Vorgangs ist der Zugriff zwar verlangsamt, aber weiterhin möglich.

Einfache Administration - Dateisysteme/Datasets anlegen

Nachdem nun der Pool mit den zugrundeliegenden Geräten angelegt wurde, lassen sich weitere Dateisysteme³ mit `zfs create` erzeugen. Hier können nun beispielsweise statt Ordnern ganze Dateisysteme mit individuellen Optionen zur Verfügung gestellt werden.

```
# zfs create tank/home
```

Auch hier steht das neu erzeugte Dateisystem sofort zur Verfügung und zwar mit der vollen Kapazität des gesamten Pools, sofern nichts anderes als Option übergeben wird (dazu später mehr).

```
# zfs list
NAME                USED    AVAIL    REFER  MOUNTPOINT
tank                146K    984M     31K    /tank
tank/home           31K     984M     31K    /tank/home
```

³ZFS verwendet den Begriff `dataset`, wir verwenden aber weiterhin den Begriff `Dateisysteme`, um den Vergleich zu traditionellen Dateisystemen einfach zu halten.

Einfache Administration - Optionen anzeigen und ändern

Verschiedene ZFS-Optionen lassen sich mit dem Befehl `zfs set` ändern.

```
# zfs set atime=off tank
```

Alle aktuell verfügbaren Optionen lassen sich mit `zfs get all` ausgeben. Um eine bestimmte Option und deren Werte auszugeben, muss man den Namen der Option hinter `zfs get` angeben:

```
# zfs get atime
NAME          PROPERTY  VALUE   SOURCE
tank          atime     off     local
tank/home     atime     off     inherited from tank
```

Wie aus der Objektorientierung bekannt, vererben hierarchisch höhere Objekte ihre Eigenschaften an darunterliegende, daher die Ausgabe `inherited from tank` in der `SOURCE`-Spalte des Dateisystems `tank/home`. So spart man einiges an Tipparbeit und setzt auf diese Weise Optionen für den gesamten Pool, egal wieviele Dateisysteme vorhanden sind. Natürlich lassen sich auch für Unterobjekte separate Optionen (Ausnahmen) festlegen.

Einfache Administration - hilfreiche Optionen

Mit `zfs get all` lassen sich alle momentan gesetzten Optionen für ein Dateisystem anzeigen. Optionen, welche in der `SOURCE`-Spalte ein `-` aufweisen, sind nur-Lese Optionen, wie z.B. das Erstellungsdatum (`zfs get creation`) oder der verwendete Plattenplatz (`used`).

```
# zfs get creation,used tank
NAME  PROPERTY  VALUE                               SOURCE
tank  creation  Sun May 13 08:13 2012              -
tank  used      634M                               -
```

Andere Optionen können vom Benutzer geändert werden. Viele davon werden beim Erstellen des Dateisystems entweder vom Vater-Dateisystem bzw. dem Pool geerbt oder sind auf Standardwerte (`default`) gesetzt.

Einfache Administration - Mountpoint ändern

Standardmässig hängt ZFS alle neuen Dateisysteme unter dem Namen des Pools ein.

```
# zfs create tank/home
# zfs list
NAME          USED    AVAIL    REFER    MOUNTPOINT
tank          146K    984M    31K     /tank
tank/home     31K     984M    31K     /tank/home
```

Soll ein anderer Pfad verwendet werden, so geschieht dies wie im folgenden Befehl:

```
# zfs set mountpoint=/usr/home tank/home
# zfs list
NAME          USED    AVAIL    REFER    MOUNTPOINT
tank          146K    984M    31K     /tank
tank/home     31K     984M    31K     /usr/home
```

Zu beachten ist, dass der Mountpoint immer mit einem führenden Slash spezifiziert werden muss (wie unter Unix üblich), der Pfad im Pool selbst aber ohne führenden Slash.

Einfache Administration - hilfreiche Optionen: exec

Mittels der `exec`-Option lässt sich definieren, ob auf dem Dateisystem Dateien ausgeführt werden können oder nicht. Dies macht in manchen Dateisystemen auch keinen Sinn, z.B. `/var/log`. Zugleich erhöht es die Sicherheit des Systems, wenn diese Option z.B. für die Home-Dateisysteme gesetzt ist.

```
# zfs create tank/home
# zfs set exec=off tank/home
# zfs create tank/home/susi
# zfs get exec
```

NAME	PROPERTY	VALUE	SOURCE
tank	exec	on	default
tank/home	exec	off	local
tank/home/susi	exec	off	inherited from tank/home

Einfache Administration - hilfreiche Optionen: exec

Äusserlich ist die Veränderung nicht sichtbar, z.B. kann das x-Flag weiterhin gesetzt werden.

```
$ ls -lah /tank/home
total 5
drwxr-xr-x  3 root  wheel    3B Dec  2 23:01 .
drwxr-xr-x  6 root  wheel    6B Dec  2 22:59 ..
drwx-----  2 susi  wheel    2B Dec  2 23:00 susi
$ cd susi
susi$ cat myscript.sh
#!/bin/sh
date ; ls
susi$ chmod +x myscript.sh
susi$ ls -l myscript.sh
-rwx--x--x  1 root  wheel   21 Dec  2 23:12 myscript.sh
susi$ ./myscript.sh
unable to execute ./myscript.sh: Permission denied
```

Selbst root darf keine Dateien auf diesem Dateisystem ausführen, so lange die exec-Option auf off gesetzt ist.

Einfache Administration - hilfreiche Optionen: readonly

Manche Datasets sollen gar nicht erst geschrieben, sondern nur gelesen werden. Für diesen Fall bietet sich die Option `readonly` an.

Standardmässig ist diese Option deaktiviert, d.h. die Datasets werden mit Lese- und Schreibberechtigungen angelegt (Vererbung beachten!).

```
# zfs create projects/finished
susi$ cp -R /home/susi/fotosafari /projects/finished
# zfs get readonly projects/finished
NAME                PROPERTY  VALUE    SOURCE
projects/finished  readonly  off      local
# zfs set readonly=on projects/finished
# zfs get readonly projects/finished
NAME                PROPERTY  VALUE    SOURCE
projects/finished  readonly  on       local
susi$ cp -R /home/susi/zoofotos /projects/finished
cp: /projects/finished/zoofotos: Read-only file system
```

Backups und Archivdateien sind typische Kandidaten für die `readonly`-Option, um diese vor einem versehentlichen Löschen durch Anwender, Anwendungen oder Skripte zu schützen.

Dateisysteme zerstören

Dateisysteme, Pools, Snapshots und Klone können über das Kommando `zfs destroy` bzw. `zpool destroy` wieder entfernt werden. Alle gespeicherten Daten gehen damit unwiederbringlich verloren. Sollten noch darunterliegende Kind-Dateisysteme existieren, warnt ZFS und bietet die entsprechende Option zum rekursiven durchlaufen an.

```
# zfs list
NAME          USED  AVAIL  REFER  MOUNTPOINT
tank          146K  984M   31K    /tank
tank/home     31K   984M   31K    /tank/home
# zfs destroy tank
cannot destroy 'tank': filesystem has children
use '-r' to destroy the following datasets:
tank/home
```

Überblick

- 1 Einleitung
- 2 Probleme heutiger Dateisysteme
- 3 Eigenschaften von ZFS
 - Einfache Administration
 - Quota und Reservierung
 - Snapshots
 - Selbstheilende Daten
 - Komprimierung
 - Deduplizierung
 - ZFS Serialisierung

Quota und Reservierung

Wie wir bereits gesehen haben, steht die gesamte Kapazität allen ZFS-Dateisystemen zur Verfügung. Traditionelle Dateisysteme nutzen dafür u.a. Partitionen, um Dateisystemen eine bestimmte Grösse vorzugeben. Da es keine Partitionen in ZFS gibt, muss es einen anderen Mechanismus geben, um den verfügbaren Speicherplatz für Dateisysteme einzuschränken. ZFS verwendet dazu **Quotas**, die genau für diesen Zweck auch in anderen Dateisystemen zur Verfügung stehen, um Speicherplatzbeschränkungen dynamisch zu ändern.

Auf diese Weise lassen sich z.B. für Home-Verzeichnisse (genauer gesagt: Home-Dateisysteme) eine Grössenbeschränkung festlegen, um zu verhindern, dass ein Benutzer das Dateisystem komplett vollschreibt und den anderen Systembenutzern den Platz wegnimmt.

Ein anderer Fall ist, bestimmten Benutzern oder Dateisystemen eine bestimmte Menge Platz zu **reservieren**. ZFS garantiert damit, dass auf jeden Fall die angegebene Menge an Speicherplatz zur Verfügung stehen wird, egal wieviel Platz bereits von anderen belegt ist (natürlich nur so viel, wie auch Speicherplatz verfügbar ist).

ZFS Quota zur Platzbeschränkung verwenden

Wir wollen eine Quota für die zuvor angelegten Home-Dateisysteme festlegen. Standardmässig steht jedem Benutzer der volle Speicherplatz des ZFS-Pools zur Verfügung, eine Quota wird nicht festgelegt.

Momentan sieht die Verteilung des Speicherplatzes folgendermassen aus:

```
$ df -h
Filesystem      Size  Used Avail Capacity  Mounted on
mypool          1.3G   31k   1.3G    0%    /mypool
mypool/home     1.3G   97k   1.3G    0%    /usr/home
mypool/home/bcr 1.3G   84k   1.3G    0%    /usr/home/bcr
$ zfs get quota
NAME                PROPERTY  VALUE   SOURCE
mypool              quota     none    default
mypool/home         quota     none    default
mypool/home/bcr     quota     none    default
```

ZFS Quota zur Platzbeschränkung verwenden

Wir setzen nun eine Quota von 500 MB auf `mypool/home/bcr` mit dem `zfs set quota`-Befehl.

```
# zfs set quota=500m mypool/home/bcr
```

```
$ df -h
Filesystem      Size      Used      Avail  Capacity  Mounted on
mypool          1.3G      31k       1.3G    0%        /mypool
mypool/home     1.3G      97k       1.3G    0%        /usr/home
mypool/home/bcr 500M      84k       499M    0%        /usr/home/bcr
$ zfs get quota
NAME                PROPERTY  VALUE   SOURCE
mypool              quota     none    default
mypool/home         quota     none    default
mypool/home/bcr    quota     500M   local
```

Die Quota betrifft also nur das angegebene Dateisystem. Durch die Vererbungsmöglichkeit liesse sich die Quota global auch für alle Home-Dateisysteme festlegen. Somit erhalten auch Dateisysteme, die in Zukunft noch angelegt werden, automatisch diese Grössenbeschränkung.

ZFS Quota zur Platzbeschränkung verwenden

Nun schreiben wir zum testen der Quota eine Datei, die grösser ist als der zur Verfügung stehende Speicherplatz in `mypool/home/bcr`.

```
# dd if=/dev/urandom of=./bigfile bs=1g
...
dd: ./bigfile: Disc quota exceeded
1+0 records in
0+1 records out
512020480 bytes transferred in 94.152408 secs (5438209 bytes/sec)
% ls -lah bigfile
-rw-r--r--  1 bcr  bcr  499M Nov 30 15:46 bigfile
$ df -h
Filesystem      Size      Used      Avail Capacity  Mounted on
mypool          1.3G       31k       1.3G    0%    /mypool
mypool/home     1.3G       97k       1.3G    0%    /usr/home
mypool/home/bcr 500M      500M        0B   100%   /usr/home/bcr
```

Die Quota-Regeln werden von ZFS strikt durchgesetzt. Mehr Speicherplatz als in der Quota angegeben können nicht allokiert werden. Der Benutzer muss also Dateien in seinem Dateisystem löschen, komprimieren oder um eine Vergrößerung der Quota bitten. Mit `zfs set quota=none` lässt sich die Quota wieder deaktivieren.

ZFS Reservierung zur Platzgarantie einsetzen

Oft kann man vorher nicht abschätzen, wieviel Platz eine bestimmte Partition verbrauchen wird. Ärgerlich ist es dann bei der Verwendung von nicht-ZFS Dateisystemen, wenn aus diesem Grund der Platz für das Betriebssystem nicht mehr ausreicht und man neu formatieren und installieren muss.

Über Reservierungen kann man in ZFS festlegen, dass ein Dateisystem eine festzulegende Menge von Platz auf jeden Fall auch bei Bedarf zugewiesen bekommt. Die Syntax dazu lautet:

```
zfs set reservation=Grösse
```

ZFS Reservierung zur Platzgarantie einsetzen

Wir wollen jetzt für das Home-Dateisystem dem Benutzer einen Plattenplatz von 500 MB von ZFS zusichern lassen.

Die Ausgangssituation ist wie zuvor:

```
$ df -h
Filesystem      Size      Used      Avail Capacity  Mounted on
mypool          1.3G      31k       1.3G      0%      /mypool
mypool/home     1.3G      97k       1.3G      0%      /usr/home
mypool/home/bcr 1.3G      84k       1.3G      0%      /usr/home/bcr
# zfs set reservation=500m mypool/home/bcr
$ df -h
Filesystem      Size      Used      Avail Capacity  Mounted on
mypool          801M      31k       801M      0%      /mypool
mypool/home     801M      97k       801M      0%      /usr/home
mypool/home/bcr 1.3G      82k       1.3G      0%      /usr/home/bcr
$ zfs get reservation
NAME                PROPERTY          VALUE          SOURCE
mypool              reservation       none          default
mypool/home         reservation       none          default
mypool/home/bcr    reservation       500M         local
```

Die Ausgabe von `df` ist zuerst etwas verwirrend. Dem Gesamt-Pool stehen nun 500 MB weniger zur Verfügung, aber das betreffende Dateisystem hat weiterhin die ursprünglichen 1.3 GB Platz verfügbar.

ZFS Reservierung zur Platzgarantie einsetzen

```
$ df -h
```

Filesystem	Size	Used	Avail	Capacity	Mounted on
mypool	801M	31k	801M	0%	/mypool
mypool/home	801M	97k	801M	0%	/usr/home
mypool/home/bcr	1.3G	82k	1.3G	0%	/usr/home/bcr

Wird von anderen Dateisystemen aber mehr Speicherplatz allokiert, fällt auch bei mypool/home/bcr die Größenangabe. Allerdings sind 500 MB diesem Dateisystem garantiert zugesichert.

```
# zfs create mypool/home/myboss
mypool$ dd if=/dev/urandom of=./mypool/home/myboss/bigfile bs=700m
dd: ./bigfile: No space left on device
2+0 records in
1+1 records out
781864960 bytes transferred in 176.822419 secs (4421752 bytes/sec)
$ df -h
```

Filesystem	Size	Used	Avail	Capacity	Mounted on
mypool	0M	31k	0M	100%	/mypool
mypool/home	0M	97k	0M	100%	/usr/home
mypool/home/bcr	500M	82k	500M	0%	/usr/home/bcr
mypool/home/myboss	0M	700M	0M	100%	/usr/home/myboss

Die Reservierung sichert selbst bei Platzmangel noch Speicherplatz zu.

ZFS Quota und Reservierung kombinieren

ZFS Quota und Reservierung lassen sich auch kombinieren. So lässt sich sowohl eine Reservierung sicherstellen und es wird gleichzeitig von ZFS dafür Sorge getragen, dass der reservierte Speicherplatz auch nicht überschritten wird.

```
# limit=500m
# zfs set quota=$limit mypool/home/bcr
# zfs set reservation=$limit mypool/home/bcr
$ df -h
```

Filesystem	Size	Used	Avail	Capacity	Mounted on
mypool	791M	31k	791M	0%	/mypool
mypool/home	791M	97k	791M	0%	/usr/home
mypool/home/bcr	500M	82k	500M	0%	/usr/home/bcr

Diese Ausgabe des verfügbaren Speicherplatzes entspricht eher derjenigen, wie man sie von traditionellen Dateisystemen gewöhnt ist.

Überblick

① Einleitung

② Probleme heutiger Dateisysteme

③ Eigenschaften von ZFS

Einfache Administration

Quota und Reservierung

Snapshots

Selbstheilende Daten

Komprimierung

Deduplizierung

ZFS Serialisierung

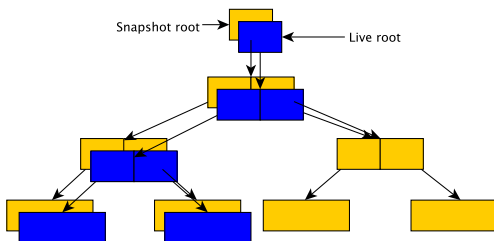
Snapshots

Unter einem **Snapshot** versteht man eine nur-Lese Kopie eines Dateisystems oder Volumes. Diese lassen sich schnell erstellen und können für die unterschiedlichsten Einsatzgebiete verwendet werden. Snapshots in ZFS sind nur auf ganzen Dateisystemen möglich, nicht auf einzelnen Dateien. Es wird also immer der übergeordnete Dateisystemeintrag verwendet und dieser beinhaltet automatisch die darin enthaltenen Dateien und Verzeichnisse.

Snapshots eignen sich besonders gut als schnelles Backup vor riskanten Aktionen. Wird ein Snapshot *vor* einer Aktion wie z.B. Systemupgrade, Softwareinstallation, Tests mit dem `rm`-Befehl, u.a. erstellt und es kommt zu Fehlern, lässt sich der Snapshot wieder einspielen und der vorherige Zustand des Dateisystems zum Zeitpunkt des Snapshots ist (ohne Neustart) wiederhergestellt.

ZFS Snapshots

Snapshots werden durch das Copy On Write-Speichermodell implizit jedesmal beim Kopieren des (Teil-)Baums erstellt. Wird ein Snapshot erstellt, hebt ZFS nur die alte Version des Dateisystems zur späteren Verwendung auf. Wird kein Snapshot erstellt, wird die Kopie, die durch COW entstanden ist, verworfen. Deshalb *kostet* das Erstellen von Snapshots auch *nichts*, denn diese sind sozusagen ein Beiprodukt der normalen Tätigkeit des Dateisystems und sofort verfügbar. Da nur Änderungen gespeichert werden, sind Snapshots obendrein noch platzsparend.



Snapshots erstellen

Um Snapshots unter ZFS zu erstellen, wird folgende Syntax verwendet:

```
zfs snapshot dataset@name
```

Für `name` lässt sich eine beliebige Bezeichnung festlegen (z.B. das Datum und die Uhrzeit). Der Snapshot wird sofort erstellt. Er kann über den Parameter `-t snapshot` bei `zfs list` angezeigt werden.

```
# zfs snapshot mypool/home/bcr@backup
# zfs list -t snapshot
NAME                                USED   AVAIL   REFER  MOUNTPOINT
mypool/home/bcr@backup              0      -    85.5K   -
```

Anhand der Ausgabe kann man erkennen, dass Snapshots nicht direkt ins System eingehängt werden, deshalb gibt es auch keine Pfadangabe unter `MOUNTPOINT`. Eine Angabe des verfügbaren Speicherplatzes (`AVAIL`) ist auch nicht vorhanden, da Snapshots nicht beschrieben werden können (read-only).

Arbeiten mit Snapshots

Vergleicht man den Snapshot mit dem Dateisystem, auf dem dieser beruht, wird klar, wie dieser entstanden ist.

```
# zfs list -rt all mypool/home/bcr
```

NAME	USED	AVAIL	REFER	MOUNTPPOINT
mypool/home/bcr	85.5K	1.29G	85.5K	/usr/home/bcr
mypool/home/bcr@backup	0	-	85.5K	-

Hier wird eine weitere Eigenschaft der ZFS Snapshots erkennbar. Snapshots speichern immer nur die Veränderungen (Delta) ab, die sich zwischen dem letzten Snapshot ergeben haben und nicht noch einmal den kompletten Inhalt des Dateisystems, um Speicherplatz zu sparen. Das bedeutet, dass ein erneuter Snapshot eines unveränderten Dateisystems keinen zusätzlichen Platz benötigt.

```
# cp /etc/passwd /usr/home/bcr
# zfs snapshot mypool/home/bcr@after_cp
# zfs list -rt all mypool/home/bcr
```

NAME	USED	AVAIL	REFER	MOUNTPPOINT
mypool/home/bcr	115K	1.29G	88K	/usr/home/bcr
mypool/home/bcr@backup	27K	-	85.5K	-
mypool/home/bcr@after_cp	0	-	88K	-

Unterschiede zwischen Snapshots anzeigen

Möchte man erfahren, welche Unterschiede zwischen zwei Snapshots bestehen, so kann dafür `zfs diff` verwendet werden. Für unser Beispiel ergibt sich dadurch:

```
# zfs diff mypool/home/bcr@backup
M /usr/home/bcr/
M /usr/home/bcr/.histfile
+ /usr/home/bcr/passwd
```

Die folgende Tabelle beschreibt den Änderungsstatus in der ersten Spalte:

Zeichen	Art der Änderung
+	Datei wurde hinzugefügt
-	Datei wurde gelöscht
M	Datei wurde geändert
R	Datei wurde umbenannt

Arbeiten mit Snapshots - Rollback

Sollte es einmal nötig sein, das Dateisystem auf den Stand eines Snapshots zurückzusetzen, so lässt sich, ähnlich der Funktionalität bei Datenbanken, der Befehl `zfs rollback` dazu nutzen.

Die Syntax lautet:

```
zfs rollback snapshot
```

Wird ein Snapshot zurückgerollt, werden alle Daten, die sich seit dem anlegen des Snapshots geändert haben, verworfen und das Dateisystem kehrt zu dem Zustand zurück, den es zum Zeitpunkt des Snapshots hatte. Standardmässig wird immer zum letzten (sprich: dem aktuellsten) Snapshot zurückgerollt. Um zu einem älteren als dem aktuellen Snapshot zurückzurollen, müssen die dazwischenliegenden Snapshots zerstört werden. Hierzu kann die Option `-r` (recursive) verwendet werden.

Beispiel zum Rollback eines Snapshots

In diesem Beispiel wird der letzte Snapshot aufgrund eines versehentlichen Löschvorgangs zurückgerollt.

```
# rm /mypool/home/bcr/*
# ls
# zfs rollback mypool/home/bcr@after_cp
# ls
passwd bundesliga.txt
# zfs list -t snapshot
```

NAME	USED	AVAIL	REFER	MOUNTPOINT
mypool/home/bcr@backup	27K	-	85.5K	-
mypool/home/bcr@after_cp	0	-	88K	-

Das Backup ist geglückt und alle Daten aus dem Snapshot wurden wiederhergestellt!

Arbeiten mit Snapshots - Rollback zum ersten Snapshot

Jetzt soll zum allerersten Snapshot zurückgerollt werden, da wir die Datei passwd nicht mehr in unserem Verzeichnis haben möchten⁴.

```
# zfs list -t snapshot
NAME                               USED   AVAIL   REFER  MOUNTPOINT
mypool/home/bcr@backup             27K    -    85.5K  -
mypool/home/bcr@after_cp           0      -     88K   -
# rm /mypool/home/bcr/*
# ls
# zfs rollback mypool/home/bcr@backup
cannot rollback to 'mypool/home/bcr@backup': more recent snapshots exist
use '-r' to force deletion of the following snapshots:
mypool/home/bcr@after_cp
# zfs rollback -r mypool/home/bcr@backup
# ls
bundesliga.txt
# zfs list -t snapshot
NAME                               USED   AVAIL   REFER  MOUNTPOINT
mypool/home/bcr@backup             27K    -    85.5K  -
```

⁴und wir zu faul sind, diese Datei einfach zu löschen. ;-)

Einzelne Dateien aus dem Snapshot zurückholen

Was kann man tun, falls man nicht den kompletten Snapshot zurückrollen möchte, aber nur ein paar Dateien wiederhergestellt werden sollen? Für diesen Fall gibt es das spezielle Verzeichnis `.zfs`, welches auf jedem ZFS Dateisystem vorhanden ist, aber standardmässig nicht angezeigt wird (selbst von `ls -a` nicht). Dieses Verhalten lässt sich durch `zfs set snapdir=visible pool` auch ändern.

Wir gehen davon aus, dass wir den Snapshot aus der vorherigen Folie nicht zurückgerollt haben.

```
# ls .zfs/snapshot
after_cp backup
# ls .zfs/snapshot/after_cp
passwd
# cp .zfs/snapshot/after_cp/passwd /mypool/home/bcr
```

Es lassen sich also einzelne Dateien aus diesem versteckten Verzeichnis wiederherstellen. Dateien *in* das Verzeichnis kopieren schlägt aufgrund der nur-lese Eigenschaft von Snapshots fehl.

```
# cp /mypool/home/bcr/bundesliga.txt .zfs/snapshot/after_cp/passwd
cp: .zfs/snapshot/after_cp/bundesliga.txt: Read-only file system
```

Arbeiten mit Snapshots - Snapshots klonen

Aus Snapshots lassen sich Klone erstellen, die eine *beschreibbare* Version eines Snapshots darstellen und als eigenständiges Dateisystem fungieren. Die Syntax dazu lautet folgendermassen:

```
zfs clone snapshot dateisystem
```

Wenn ein Klon angelegt wird, erstellt ZFS eine implizite Abhängigkeit zwischen Klon und Snapshot. Dadurch kann der Snapshot nach anlegen eines Klons nicht gelöscht werden, so lange noch abhängige Klone existieren.

Um diese Abhängigkeit aufzulösen, lassen sich Klone zu echten Dateisystemen mit dem Kommando `zfs promote` Klon "ernennen" (engl. promote). Der Snapshot ist nun ein vom Klon unabhängiges Dateisystem und kann dann bei Bedarf gelöscht werden. Der Klon lässt sich an einer beliebigen Stelle im ZFS Dateisystem einhängen, es muss also nicht unbedingt der ursprüngliche Ort des Snapshots sein.

Arbeiten mit Snapshots - Klone aus Snapshots anlegen

Wir gehen wieder von folgendem Stand aus:

```
# zfs list -rt all mypool/home/bcr
NAME                                USED   AVAIL   REFER  MOUNTPOINT
mypool/home/bcr                     108K   1.29G   87K    /usr/home/bcr
mypool/home/bcr@backup               21K    -       85.5K  -
mypool/home/bcr@after_cp             0      -       87K    -
```

Es soll jetzt ein Klon des Dateisystems zum Zeitpunkt des letzten Snapshots angelegt werden.

```
# zfs clone mypool/home/bcr@after_cp mypool/home/bcrnew
# ls /home/bcr*
/home/bcr
bundesliga.txt passwd

/home/bcrnew
bundesliga.txt passwd
# df -h
Filesystem                Size      Used    Avail  Capacity  Mounted on
mypool                     1.3G      31k     1.3G    0%        /mypool
mypool/home                 1.3G      98k     1.3G    0%        /usr/home
mypool/home/bcr             1.3G      87k     1.3G    0%        /usr/home/bcr
mypool/home/bcrnew          1.3G      87k     1.3G    0%        /usr/home/bcrnew
```

Arbeiten mit Snapshots - Klon promoten

Das durch das Klonen erstellte Dateisystem besitzt die gleichen Eigenschaften wie der Snapshot, auf dem es basiert. Wir kopieren zur Demonstration eine weitere Datei in den beschreibbaren Klon.

```
# cp /boot/defaults/loader.conf /usr/home/bcrnew
# zfs get origin mypool/home/bcrnew
NAME                PROPERTY  VALUE                SOURCE
mypool/home/bcrnew  origin    mypool/home/bcr@after_cp  -
# zfs promote mypool/home/bcrnew
# zfs get origin mypool/home/bcrnew
NAME                PROPERTY  VALUE                SOURCE
mypool/home/bcrnew  origin    -                    -
```

Nun soll der Klon als neues Home-Dateisystem agieren. Dabei ist zu beachten, dass der Klon nicht den gleichen Namen haben darf wie ein bereits bestehender Snapshot. Mit `zfs rename` lassen sich Dateisysteme umbenennen.

```
# zfs destroy -f mypool/home/bcr
# zfs rename mypool/home/bcrnew mypool/home/bcr
# ls /home/bcr
passwd bundesliga.txt loader.conf
```

Die Datei `loader.conf` haben wir aus dem Klon erhalten (siehe oben).

Überblick

① Einleitung

② Probleme heutiger Dateisysteme

③ Eigenschaften von ZFS

Einfache Administration

Quota und Reservierung

Snapshots

Selbstheilende Daten

Komprimierung

Deduplizierung

ZFS Serialisierung

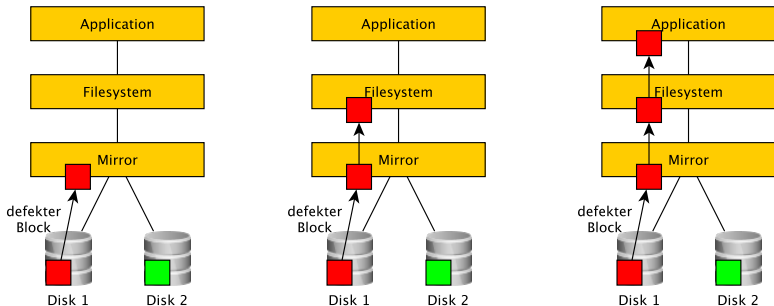
Selbstheilende Daten

Wie bereits angesprochen, führen traditionelle Dateisysteme ihre Operationen nicht unbedingt atomar aus. Daher sind nach einem Systemausfall inkonsistente (Meta-)Daten vorhanden, die durch einen `fsck` vor dem Mounten behoben werden. Dadurch fallen lange Auszeiten des Speichers an und Fehler wie der Verlust von nicht-persistenten (nur im RAM verfügbaren, noch nicht geschriebenen) Daten und inkonsistente Spiegel (Split-Brain) können nicht verhindert werden.

ZFS hingegen ist in der Lage, defekte Daten anhand der Prüfsummen im Betrieb zu erkennen und zu korrigieren. Ein `fsck` ist dadurch nicht nötig, die Prüfung mittels `zpool scrub` kann in weniger betriebsamen Zeiten (z.B. nachts) durchgeführt werden. Dadurch entfallen Auszeiten und die gespeicherten Daten sind weiterhin verfügbar. Lese-/Schreiboperationen sind während des Prüfvorgangs zwar verlangsamt, aber weiterhin möglich.

Traditionelle Dateisysteme erkennen nicht alle Fehler

Besonders bei traditionellen redundant vorgehaltenen Daten (z.B. RAID1) zeigt sich das Problem von unerkannten und nicht behobenen Fehlern deutlich.



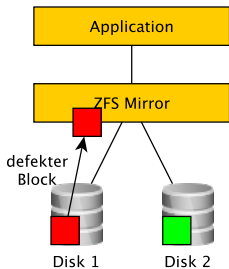
1. Defekten Block gelesen

2. Falsche Metadaten im FS

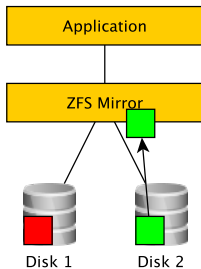
3. Falsche Daten in Anwendung

Traditionelle Dateisysteme können keine fehlerhaft gespeicherten Daten erkennen und korrigieren! Manche dieser Daten werden erst nach Jahren beim erneuten Zugriff erkannt. Backups sind besonders davon betroffen.

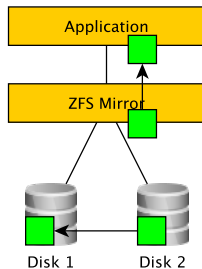
Selbstheilende Daten in ZFS durch Nutzung der Redundanz



1. Defekter Block wird gelesen



2. ZFS erkennt falsche Prüfsumme, liest den Spiegel mit korrekter Prüfsumme



3. Korrekte Daten werden vom Spiegel zur Anwendung geschickt *und* der defekte Block korrigiert!

Demo zur Selbstheilungsfähigkeit von ZFS

Dieses Beispiel führt das Beispiel auf der vorherigen Folie am einem ZFS Pool praktisch vor. Dazu erstellen wir einen Spiegel und kopieren ein paar Daten darauf. Eine Prüfsumme wird berechnet, um diese später als Referenz für einen intakten Pool zu verwenden. Anschliessend exportieren wir den Pool und schreiben auf eines der Geräte des Pools (nicht den Pool selbst) zufällige Daten. Danach importieren wir den Pool wieder, lassen uns den Status des Pools ausgeben und erzeugen eine zweite Prüfsumme, die wir nun mit der ersten vergleichen. Stimmen diese überein, hat ZFS den Fehler durch die Spiegelkopie automatisch korrigiert.

Demo zur Selbstheilungsfähigkeit von ZFS 1/5

```
# zpool create heiler mirror disk1 disk2
# zpool status
  pool: heiler
  state: ONLINE
  scan: none requested
config:

  NAME            STATE          READ  WRITE CKSUM
  heiler          ONLINE         0     0     0
    mirror-0     ONLINE         0     0     0
      disk1      ONLINE         0     0     0
      disk2      ONLINE         0     0     0

  errors: No known data errors
# zpool list
NAME      SIZE  ALLOC   FREE    CAP  DEDUP  HEALTH  ALTROOT
heiler    960M  92.5K   960M    0%   1.00x  ONLINE  -
# cp /some/important/data /heiler
# zfs list
NAME      SIZE  ALLOC   FREE    CAP  DEDUP  HEALTH  ALTROOT
heiler    960M  67.7M   892M    7%   1.00x  ONLINE  -
# sha1 /heiler > checksum.txt
# cat checksum.txt
SHA1 (/heiler) = 2753eff56d77d9a536ece6694bf0a82740344d1f
```


Demo zur Selbstheilungsfähigkeit von ZFS 2/5

Achtung:

Das hier gezeigte Beispiel mit `dd` kann Daten zerstören, wenn das falsche Gerät oder ein nicht-ZFS Dateisystem angegeben wird. **Benutzung auf eigene Gefahr!**

```
# zpool export heiler
# dd if=/dev/random of=disk1 bs=1m count=200
200+0 records in
200+0 records out
209715200 bytes transferred in 62.992162 secs (3329227 bytes/sec)
# zpool import heiler
```

Demo zur Selbstheilungsfähigkeit von ZFS 3/5

```
# zpool status
  pool: heiler
  state: ONLINE
  status: One or more devices has experienced an unrecoverable error. An
         attempt was made to correct the error. Applications are unaffected.
  action: Determine if the device needs to be replaced, and clear the errors
         using 'zpool clear' or replace the device with 'zpool replace'.
         see: http://www.sun.com/msg/ZFS-8000-9P
  scan: none requested
  config:

    NAME        STATE      READ  WRITE CKSUM
    heiler      ONLINE    0     0     0
    mirror-0    ONLINE    0     0     0
      disk1     ONLINE    0     0     0
      disk2     ONLINE    0     0     1

  errors: No known data errors
```

Wir vergleichen nun die Prüfsumme des Pools nach dem Import mit der vorherigen.

```
# sha1 /heiler >> checksum.txt
# cat checksum.txt
SHA1 (/heiler) = 2753eff56d77d9a536ece6694bf0a82740344d1f
SHA1 (/heiler) = 2753eff56d77d9a536ece6694bf0a82740344d1f
```

Demo zur Selbstheilungsfähigkeit von ZFS 4/5

```
# zpool scrub heiler
# zpool status
  pool: heiler
  state: ONLINE
status: One or more devices has experienced an unrecoverable error. An
        attempt was made to correct the error. Applications are unaffected.
action: Determine if the device needs to be replaced, and clear the errors
        using 'zpool clear' or replace the device with 'zpool replace'.
        see: http://www.sun.com/msg/ZFS-8000-9P
scan: scrub in progress since Mon Dec 10 12:23:30 2012
      10.4M scanned out of 67.0M at 267K/s, 0h3m to go
      9.63M repaired, 15.56% done

config:

NAME      STATE      READ WRITE CKSUM
heiler    ONLINE    0     0     0
  mirror-0 ONLINE    0     0     0
    disk1  ONLINE    0     0     0
    disk2  ONLINE    0     0    627 (repairing)

errors: No known data errors
...
```

Demo zur Selbstheilungsfähigkeit von ZFS 5/5

```
# zpool status
  pool: heiler
  state: ONLINE
status: One or more devices has experienced an unrecoverable error. An
        attempt was made to correct the error. Applications are unaffected.
action: Determine if the device needs to be replaced, and clear the errors
        using 'zpool clear' or replace the device with 'zpool replace'.
       see: http://www.sun.com/msg/ZFS-8000-9P
       scan: scrub repaired 66.5M in 0h2m with 0 errors on Mon Dec 10 12:26:25 2012
config:

NAME          STATE      READ WRITE CKSUM
heiler        ONLINE    0     0     0
  mirror-0    ONLINE    0     0     0
    disk1     ONLINE    0     0     0
    disk2     ONLINE    0     0 2.72K

errors: No known data errors
# zpool clear heiler
# zpool status
  pool: heiler
  state: ONLINE
scan: scrub repaired 66.5M in 0h2m with 0 errors on Mon Dec 10 12:26:25 2012
config:

NAME          STATE      READ WRITE CKSUM
heiler        ONLINE    0     0     0
  mirror-0    ONLINE    0     0     0
    disk1     ONLINE    0     0     0
    disk2     ONLINE    0     0     0

errors: No known data errors
```

Überblick

① Einleitung

② Probleme heutiger Dateisysteme

③ Eigenschaften von ZFS

Einfache Administration

Quota und Reservierung

Snapshots

Selbstheilende Daten

Komprimierung

Deduplizierung

ZFS Serialisierung

Komprimierung

Ein klassischer Weg, um Speicherplatz zu sparen ist die Komprimierung von Dateien mit gängigen Werkzeugen wie Zip u.ä. In ZFS kann die Komprimierung für ein Dateisystem aktiviert werden, so dass das Dateisystem transparent Daten beim speichern komprimiert und beim laden automatisch entpackt. Somit entfällt der Umweg über ein externes Werkzeug und das Dateisystem kann so mehr Daten speichern.

Aktiviert man die Komprimierung für ein Dateisystem in ZFS, so werden ab diesem Zeitpunkt *nur neu gespeicherte Daten* komprimiert. Bereits bestehende Daten besitzen weiterhin ihre vorherige Grösse, es sei denn, diese werden neu abgespeichert.

Zur Komprimierung stehen verschiedene Algorithmen zur Verfügung, die je nach Art der gespeicherten Daten unterschiedlich starke Kompressionsraten aufweisen. Natürlich werden etwas mehr CPU-Ressourcen für die Komprimierung benötigt, was bei heutigen Mehrprozessorsystemen jedoch nicht mehr so stark ins Gewicht fällt wie früher.

Komprimierung - Praxisbeispiel

Der FreeBSD Ports Tree besteht aus Textdateien (Makefiles, Patche, Beschreibungsdateien, etc.), die sich gut komprimieren lassen. Zuerst legen wir die nötige Verzeichnishierarchie an:

```
# zfs create tank/usr
# zfs create tank/usr/ports
# zfs create -o mountpoint=/usr/ports tank/usr/ports
```

Danach setzen wir die Komprimierung auf den GZIP-Algorithmus; dies hätte auch bereits beim anlegen durch die `-o` Option passieren können.

```
# zfs set compression=gzip tank/usr/ports
# zfs get compressratio tank/usr/ports
NAME          PROPERTY      VALUE  SOURCE
tank/usr/ports compressratio  1.00x  -
```

Jetzt laden wir eine aktuelle Kopie der Ports nach `/usr/ports` herunter.

```
# git clone https://git.FreeBSD.org/ports.git /usr/ports
```

Nach diesem Vorgang prüfen wir die Kompressionsrate erneut.

```
# zfs get compressratio tank/usr/ports
NAME          PROPERTY      VALUE  SOURCE
tank/usr/ports compressratio  3.02x  -
```

Überblick

① Einleitung

② Probleme heutiger Dateisysteme

③ Eigenschaften von ZFS

Einfache Administration

Quota und Reservierung

Snapshots

Selbstheilende Daten

Komprimierung

Deduplizierung

ZFS Serialisierung

Deduplizierung

Oft werden Daten doppelt auf Datenträgern bzw. Dateisystemen gespeichert. Dies geschieht einerseits absichtlich aus Redundanzgründen, oft aber auch unabsichtlich und ungewollt. Manchmal ist es dem Benutzer auch gar nicht klar, dass Daten doppelt abgelegt werden, beispielsweise halten manche Applikationen bestimmte Daten nochmal separat in einem eigenen (versteckten) Verzeichnis vor. Diese mehrfach gespeicherten Daten nehmen natürlich auch durch jede Kopie zusätzlichen Speicherplatz in Anspruch. **Deduplizierung** ist ein Ansatz, das doppelte Vorhalten von Daten zu reduzieren. Es kann als Gegenstück zur Datenredundanz angesehen werden.

In ZFS ist Deduplizierung auf Blockebene implementiert. Bemerkt ZFS bei aktivierter Deduplizierung auf einem Pool anhand der Prüfsumme, dass ein bereits vorhandener Block A nochmal an eine andere Stelle geschrieben werden soll, wird dieser neue Block B durch einen Verweis auf Block A ersetzt. Dieser Verweis belegt weniger Speicherplatz und wird transparent für den Benutzer verwaltet. Somit lässt sich bei vielen redundanten Daten eine Menge Speicherplatz einsparen.

Deduplizierung nutzen

Um Deduplizierung zu aktivieren, genügt es, die folgende Einstellung vorzunehmen:

```
zfs set dedup=on pool
```

Durch diese Einstellung werden die Prüfsummen jedes neuen Blocks geprüft und falls ein solcher Block bereits existiert, durch einen Verweis ersetzt. Diese Art der Prüfung wird im RAM durchgeführt, aus diesem Grund ist ZFS mit Deduplizierung nicht unbedingt auf jedem Pool zu empfehlen. Bereits geschriebene und doppelt vorhandene Blöcke werden nicht im nachhinein durch die obige Aktivierung dedupliziert. Stattdessen müssen die Daten entweder neu kopiert oder verändert werden, um die erneute Prüfung auf Deduplizierung zu veranlassen.

Ein Pool, bei dem die Deduplizierung gerade erst aktiviert wurde, sieht typischerweise so aus:

```
# zpool list
NAME      SIZE  ALLOC  FREE   CAP  DEDUP  HEALTH  ALTROOT
mypool    2.84G  2.19M  2.83G   0%  1.00x  ONLINE  -
```

Die Spalte DEDUP zeigt die aktuelle Rate der Deduplizierung an.

Beispiel zur Deduplizierung

In diesem Beispiel werden Daten in unterschiedlichen Verzeichnissen auf dem gleichen ZFS Dateisystem abgelegt, das sich auf einem Pool mit aktivierter Deduplizierung befindet.

```
# zpool list
NAME      SIZE  ALLOC   FREE      CAP  DEDUP   HEALTH  ALTROOT
mypool    2.84G  2.19M   2.83G     0%  1.00x   ONLINE  -
# zfs get dedup mypool
NAME      PROPERTY  VALUE          SOURCE
mypool    dedup     on             local
# cd /mypool
# for d in dir1 dir2 dir3; do
for> mkdir $d && cp -R /usr/ports $d &
for> done
...
# zpool list
NAME      SIZE  ALLOC   FREE      CAP  DEDUP   HEALTH  ALTROOT
mypool    2.84G  20.9M   2.82G     0%  3.00x   ONLINE  -
# df -h mypool
Filesystem      Size      Used      Avail Capacity  Mounted on
mypool           2.8G       54M       2.8G      2%      /mypool
```

Prüfung, ob sich Deduplizierung für ein Dateisystem lohnt

Man kann ZFS prüfen lassen, ob sich die Aktivierung der Deduplizierung bei einer gegebenen Menge von Daten lohnt. Dazu nutzen wir den ZFS Debugger `zdb`, der eine Option zur Simulation von Deduplizierung bietet.

```
# zdb -S mypool
Simulated DDT histogram:
```

bucket	allocated				referenced				
	refcnt	blocks	LSIZE	PSIZE	DSIZE	blocks	LSIZE	PSIZE	DSIZE
1	17.6K	32.6M	32.6M	32.6M	17.6K	32.6M	32.6M	32.6M	32.6M
2	102	73.5K	73.5K	73.5K	224	164K	164K	164K	164K
4	6	3K	3K	3K	30	15K	15K	15K	15K
16	1	1K	1K	1K	23	23K	23K	23K	23K
Total	17.7K	32.7M	32.7M	32.7M	17.9K	32.8M	32.8M	32.8M	32.8M

dedup = 1.00, compress = 1.00, copies = 1.00, dedup * compress / copies = 1.00

In diesem Fall wird sich die Aktivierung der Deduplizierung *nicht lohnen*. Faktoren, welche die Deduplizierung begünstigen ist eine aktivierte Komprimierung und die Verwendung der `copies`-Eigenschaft. Bei letzterer legt ZFS selbstständig weitere Kopien im Dateisystem an, was bei weniger stabilen Datenträgern sinnvoll ist (z.B. USB-Sticks, Speicherkarten, Notebookplatten, etc.). Diese redundanten Kopien sind genau das richtige für die Deduplizierung.

Überblick

① Einleitung

② Probleme heutiger Dateisysteme

③ Eigenschaften von ZFS

Einfache Administration

Quota und Reservierung

Snapshots

Selbstheilende Daten

Komprimierung

Deduplizierung

ZFS Serialisierung

ZFS Serialisierung

ZFS enthält eine eingebaute Funktion zur Serialisierung des Speichers, so dass dieser als Stream auf die Standardausgabe geschrieben werden kann. Dazu werden Snapshots der Dateisysteme benötigt. Auf diese Weise ist es möglich, ein Dateisystem zwischen zwei Systemen oder ZFS Pools zu transferieren. Der Befehl dazu lautet `zfs send`.

An den folgenden Pools soll diese Funktionalität demonstriert werden:

#	zpool	list						
	NAME	SIZE	ALLOC	FREE	CAP	DEDUP	HEALTH	ALTROOT
	backup	960M	77K	960M	0%	1.00x	ONLINE	-
	mypool	984M	43.7M	940M	4%	1.00x	ONLINE	-

ZFS Serialisierung

An den folgenden Pools soll diese Funktionalität demonstriert werden:

```
# zpool list
NAME      SIZE  ALLOC   FREE      CAP  DEDUP  HEALTH  ALTROOT
backup    960M   77K   960M       0%  1.00x  ONLINE  -
mypool    984M  43.7M  940M       4%  1.00x  ONLINE  -
# zfs snapshot mypool@backup1
# zfs list -t snapshot
NAME                                     USED   AVAIL   REFER  MOUNTPOINT
mypool@backup1                          0      -   43.6M  -
# zfs send mypool@backup1
Error: Stream can not be written to a terminal.
You must redirect standard output.
# zfs send mypool@backup1 > /backup/backup1
# zpool list
NAME      SIZE  ALLOC   FREE      CAP  DEDUP  HEALTH  ALTROOT
backup    960M  63.7M  896M       6%  1.00x  ONLINE  -
mypool    984M  43.7M  940M       4%  1.00x  ONLINE  -
```

ZFS inkrementelle Backups

Änderungen zwischen zwei Snapshots lassen sich ebenfalls serialisieren und mittels `zfs send` übertragen. Dabei werden nur diejenigen Daten, welche sich zwischen den beiden Snapshots geändert haben, übertragen. Dadurch handelt es sich um ein inkrementelles Backup.

```
# zfs snapshot mypool@backup2
# zfs list -t snapshot
NAME                                USED  AVAIL  REFER  MOUNTPOINT
mypool@backup1                      5.72M   -    43.6M   -
mypool@backup2                       0       -    44.1M   -
# zpool list
NAME      SIZE  ALLOC  FREE    CAP  DEDUP  HEALTH  ALTROOT
backup    960M  61.7M  898M    6%   1.00x  ONLINE  -
mypool    960M  50.2M  910M    5%   1.00x  ONLINE  -
# zfs send -i mypool@backup1 mypool@backup2 > /backup/diff
# zpool list
NAME      SIZE  ALLOC  FREE    CAP  DEDUP  HEALTH  ALTROOT
backup    960M  80.8M  879M    8%   1.00x  ONLINE  -
mypool    960M  50.2M  910M    5%   1.00x  ONLINE  -
# ls -lah /backup
total 82247
drwxr-xr-x  2 root  wheel    4B Dec  3 11:46 .
drwxr-xr-x 21 root  wheel   28B Dec  3 11:32 ..
-rw-r--r--  1 root  wheel   61M Dec  3 11:36 backup1
-rw-r--r--  1 root  wheel   18M Dec  3 11:46 diff
```


ZFS inkrementelle Backups - Daten wiederherstellen

Wir haben zwar nun die Backups erhalten, diese liegen aber immer noch im binären Stream-Format vor. Wenn wir an die Daten selbst gelangen möchten, müssen wir `zfs send` mit `zfs receive` bzw. die Abkürzung `zfs recv` kombinieren.

```
# zfs send mypool@backup1 | zfs receive backup/backup1
# ls -lah /backup/
total 431
drwxr-xr-x    3 root  wheel   3B Dec  3 12:06 .
drwxr-xr-x   21 root  wheel  28B Dec  3 12:05 ..
drwxr-xr-x 4219 root  wheel  4.1k Dec  3 11:34 backup1
```

In `backup1` finden wir nun alle Daten, welche aus dem Snapshot `mypool@backup1` stammen.

```
# zfs list
```

NAME	USED	AVAIL	REFER	MOUNTPOINT
backup	43.7M	884M	32K	/backup
backup/backup1	43.5M	884M	43.5M	/backup/backup1
mypool	50.0M	878M	44.1M	/mypool

ZFS Backups - verschlüsselte Backups über das Netzwerk

Da ZFS mit der Standardausgabe arbeitet, lassen sich die bekannten Funktionen zur Umleitung der Ausgabe nutzen. Die Streams lassen sich über SSH verschlüsselt an einen anderen Rechner mit ZFS (sog. Cold Standby) über ein unsicheres Netzwerk bzw. das Internet senden. Damit dies funktioniert, müssen eine Reihe von Bedingungen erfüllt sein:

- Passwortloser SSH-Zugang mittels SSH-Keys zum Zielhost
- `root` muss sich über SSH anmelden dürfen
- Das Zielsystem sollte `root` nur die Ausführung des `zfs recv`-Kommandos erlauben (lässt sich in SSH konfigurieren)

Als `cron(8)`-Job lässt sich dieses Backup in regelmässigen Intervallen automatisiert durchführen.

ZFS Backups - verschlüsselte Backups über das Netzwerk




Als Beispiel soll ein rekursives Backup aller Homeverzeichnisse auf `host1` durchgeführt und über das Netzwerk an einen anderen Rechner `host2` übertragen werden.

```
host1# zfs snapshot -r tank/home@montag  
host1# zfs send -R tank/home@montag | ssh host2 zfs recv -dvu pool
```

Die Option `-R` sorgt dafür, dass alle angegebenen Dateisysteme und deren darunterliegenden Kinder rekursiv übertragen werden. Das schliesst Snapshots, Klone und vorgenommenen Einstellungen am Dateisystem mit ein.

Durch die Angabe von `-d` bei `zfs recv` wird der ursprüngliche Poolname auf der Empfängerseite entfernt und nur der Name des Snapshots verwendet. Die Option `-u` legt fest, dass das Empfängerdateisystem nicht gemountet ist. Mehr Informationen werden durch die Angabe `-v` ausgegeben, beispielsweise die während der Empfängeroperation verstrichene Zeit.

Weiterführende Informationen

-  [FBSDBZFS] The FreeBSD Handbook - 21. The Z File System (ZFS)
<https://docs.freebsd.org/en/books/handbook/zfs/>
-  [ZFSLinux] ZFS on Linux
<http://zfsonlinux.org/>
-  [CSIMUNICH] CSI:Munich - Demonstrating ZFS on USB sticks
<http://www.youtube.com/watch?v=1zw8V8g5eT0>