OpenZFS Cheat-Sheet

Storage-Pool erstellen

Single Disk Nicht redundant, unter `/mypool` sofort im System verfügbar, kein Eintrag in `/etc/fstab` nötig.
`zpool create mypool disk1`

Stripe (RAID-0) Keine Redundanz, bei Ausfall einer Platte sind alle Pooldaten unwiederbringlich verloren! Gute Geschwindigkeit, Speicherplatz beider Platten kann voll genutzt werden.
`zpool create mystripe disk1 disk2`

Mirror (RAID-1) Ausfallsicher, `mirror-0` VDEV bei `zpool status`, paralleles Lesen von allen Platten, langsames Schreiben auf alle Platten, geringere Gesamtkapazität als bei Stripe.
`zpool create spiegel mirror disk1 disk2`

Single-Parity (RAID-Z1) Min. 3 Platten nötig, Ausfallsicher (eine Platte), verteilt Parity-Informationen über alle Platten, schnelles Lesen und Schreiben mit etwa gleicher Geschwindigkeit, Kapazität 66%.
`raidz1-0` VDEV bei `zpool status`.
`zpool create paritypool raidz disk1 disk2 disk3`

Double-Parity (RAID-Z2) Min. 4 Platten nötig, Ausfallsicher (zwei Platten), zwei Parity-Platten pro VDEV, etwas langsamer als RAID-Z1, Kapazität 50%.
`zpool create double raidz2 disk1 disk2 disk3 disk4`

RAID10 Min. 4 Platten nötig, Ausfallsicher (2 Platten, eine pro VDEV), hohe Lesegeschwindigkeit, halb so schnelle Schreibgeschwindigkeit, Kapazität 50%. Guter Kompromiss zwischen Ausfallsicherheit, Kapazität und Geschwindigkeit.
`zpool create myraid10 mirror disk1 disk2 mirror disk3 disk4`

Triple-Parity (RAID-Z3) Min. 5 Platten nötig, Ausfallsicher (drei Platten), drei Parity-Platten pro VDEV, nur wenig langsamer als RAID-Z2, Kapazität 40%.
`zpool create triple raidz3 disk1 disk2 disk3 disk4 disk5`

Poolinformationen anzeigen

`zpool status` Poolstatus anzeigen, inkl. Plattenkonfiguration, evtl. Fehler und Updates, letzter Scrub-Zustand.

`zpool list` Poolkapazität, belegten und freien Platz anzeigen.

`zpool iostat` I/O-Statistiken anzeigen, sowohl über den gesamten Pool als auch über einzelne Geräte.

`zpool history` Administrative Historie des Pools anzeigen.

`zpool get` Pool-Eigenschaften (Properties) und dessen Werte anzeigen.

ZFS Datasets

Bauen auf Zpool auf und werden über dessen Namen angesprochen. Jeder Pool besitzt auch ein Dataset mit gleichem Namen als oberste Ebene. **Beispiel:**
`zpool create testpool disk1`
erzeugt ein Dataset `testpool`, unter dem sich weitere Datasets als Kind-Datasets anlegen lassen. Unter `/testpool` ist der Pool im Dateisystem eingehängt. Administrative ZFS-Befehle dürfen keinen führenden / besitzen!

Dataset anlegen Erzeugt ein neues Dataset namens `ds`, das die meisten Eigenschaften vom Eltern-Dataset (hier `mypool`) erbt.
`zfs create mypool/ds`
Mehrere Kinddatasets lassen sich mit dem Parameter `-p` anlegen: `zfs create -p mypool/home/fred`

Dataset anzeigen Listet alle Datasets, den verbrauchten und verfügbaren Platz, sowie evtl. Mountpoint auf.
`zfs list`
Ein einzelnes Dataset anzeigen durch Angabe des ZFS-Pfads (kein führender `/`): `zfs list mypool/ds`
Dataset und evtl. Kind-Datasets rekursiv anzeigen:
`zfs list -r mypool/ds` Anzeigetiefe von tief verschachtelten Datasets begrenzen (`-d`: depth):
`zfs list -d 1 mypool/home`
Ausgabe filtern und nur den Dataset-Namen anzeigen:
`zfs list -o name mypool/home`
Ausgabe filtern und Header entfernen (für Skripte):
`zfs list -Ho name mypool/home`
Reihenfolge der Ausgabe ändern:
`zfs list -o used,avail,refer,mountpoint,name`

Speicherplatz anzeigen Listet verfügbaren und benutzten Speicherplatz pro Dataset und dessen Kindern auf, sowie den von Snapshots und Reservierungen belegten Platz an. **Empfehlung:** als Ersatz für `df -h` benutzen.
`zfs list -o space`

Dataset umbenennen Neuen Namen vergeben bzw. die Position im Dateisystem-Pfad) ändern. Entspricht dem Unix `mv`-Kommando.
`zfs rename mypool/home/fred mypool/home/eva`

Dataset löschen Zerstört das Dataset und die darin enthaltenen Daten sofort und unwiederbringlich.
Simulieren der Aktion mit Anzeige der betroffenen Datasets, ohne die Aktion durchzuführen (Dry-Run):
`zfs destroy -nv mypool/home/eva`
Ausgabe: `would destroy mypool/home/eva`
Zerstören wirklich durchführen und Ergebnis anzeigen (verbose):
`zfs destroy -v mypool/home/eva`
Ausgabe: `will destroy mypool/home/eva`
Rekursives Zerstören von Kind-Datasets ohne Ausgabe:
`zfs destroy -r mypool/testdataset`

Properties

Die Flexibilität von Datasets besteht darin, dass sie einerseits vom Vater-Dataset Eigenschaften übernehmen, diese aber auch bei Bedarf mit eigenen Werten überschreiben können. Nur Properties mit `default` in der Spalte `SOURCE` lassen sich ändern. Ähnliches gilt für Pool-Properties.

Properties anzeigen Alle Properties eines Pools anzeigen:
`zpool get all mypool`
Alle Properties eines Datasets anzeigen:
`zfs get all`
Einzelne Property (`capacity`) anzeigen:
`zpool get capacity mypool`
Mehrere Properties mit Komma getrennt (ohne Leerzeichen) auflisten:
`zpool get capacity,health mypool`

Properties ändern Die `atime` (access time) Property deaktivieren:
`zfs set atime=off mypool`
Mountpoint eines Datasets ändern:
`zfs set mountpoint=/mnt/dataset mypool/ds`

Benutzerdefinierte Properties Eigene Properties lassen sich definieren und mit einem eigenen Namensraum von den ZFS-eigenen abgrenzen:
`zfs set platten:kaufdatum=12.03.2022 mypool`
Jedes Dataset erbt automatisch diesen Wert, kann diesen aber auch ändern. Anzeige der eigenen Property wie jede andere auch:
`zfs get platten:kaufdatum mypool` Um eine selbst-erstellte Property zurückzusetzen (kein Wert mehr vorhanden), wird das Unterkommando `inherit` benutzt:
`zfs inherit platten:kaufdatum mypool`
Um eine Property komplett zu entfernen, dient der Parameter `-r`:
`zfs inherit -r platten:kaufdatum mypool`

Scrub

ZFS speichert Prüfsummen mit den Daten und im gesamten Dateisystembaum, um Fehler zu erkennen und zu korrigieren. Durch verschiedene Faktoren (I/O-Fehler, fehlerhafte Treiber oder RAM, defekte Kabel, usw.) kann diese Prüfsumme fehlerhaft sein und passt nicht mehr zu den gespeicherten Daten. Ist genug Redundanz im Pool vorhanden (VDEV), findet ZFS diese Fehler nicht nur, sondern korrigiert sie auch (sog. Self-Healing). Es wird empfohlen, regelmässig sog. `scrub`s auf dem Pool durchzuführen (monatlicher `cron`-Job), um diese Prüfsummen auszulesen und ggfs. zu korrigieren.
`zpool scrub mypool`
Mit `zpool status` lässt sich der Fortschritt beobachten und evtl. aufgetretene Fehler anzeigen. Ein `fsck` benötigt ZFS nicht.

Quota

Jedes Dataset kann den gesamten Speicherplatz des Pools nutzen. Dies lässt sich mit Quotas einschränken, so dass diese nicht mehr als die definierte Menge Platz belegen dürfen. Schreibvorgänge über die Quota hinaus bricht ZFS rigoros ab.

Quota festlegen Die Quota-Property lässt sich mit `set` für ein Dataset bestimmen:
`zfs set quota=10G mypool/dataset`
Diese Quota gilt dann für dieses Dataset sowie alle Kind-Datasets, die ggfs. schon existieren oder noch entstehen. Die Datasets teilen sich also alle die Gesamt-Quota. Soll nur dieses Dataset ohne die Kinder eine Quota erhalten, kommt stattdessen `refquota` zum Einsatz:
`zfs set refquota=10G mypool/dataset`
Quota für einen bestimmten Benutzer im System auf diesem Dataset festlegen:
`zfs set userquota@fred=10G mypool/home/fred`
Quota für eine Benutzergruppe im System auf diesem Dataset festlegen:
`zfs set groupquota@projektX=100G mypool/projektX`

Quota anzeigen Die Quota-Property mit `get` abfragen:
`zfs get quota mypool/dataset`
Ebenso für die `refquota`:
`zfs get refquota mypool/home/fred`
Bei Benutzerquotas lässt sich anzeigen, wieviel diese bereits verbraucht haben:
`zfs userspace mypool/home/fred`
Gleiches gilt für die Gruppenquota:
`zfs groupspace mypool/projektX`

Quota deaktivieren Durch den Wert `none` wird die Quota wieder deaktiviert: `zfs set quota=none mypool/home/fred`

Reservierung

Durch eine Reservierung wird einem Dataset eine bestimmte Menge Speicherplatz des Pools garantiert, egal wieviel andere Datasets verbrauchen (reduziert den Gesamtspeicher des Pools). So wird ein komplettes Vollschreiben des Pools verhindert und ermöglicht Kapazitätsplanung für die Zukunft.

Reservierung festlegen Die `reservation`-Property bestimmt den zu reservierenden Speicherplatz für ein Dataset:
`zfs set reservation=100G mypool/home` Genau wie bei Quotas auch, gilt diese Reservierung auch für Kind-Datasets. Um ein einzelnes Dataset *ohne* Kind-Datasets mit einer Reservierung zu versehen ist `refreservation` zu verwenden:
`zfs set refreservation=10G mypool/home/eva`

Reservierung anzeigen Die `reservation`-Property zeigt die eingestellte Reservierung (oder `none`) an:
`zfs get reservation mypool/home/eva`
Ebenfalls zeigen `zfs list -o space` die verwendete Reservierung in der Spalte `USEDREFRESERV` an.

Reservierung entfernen Durch zuweisen von `none` zu den `refreservation` und `reservation` Properties wird diese wieder zurückgesetzt:
`zfs set reservation=none mypool/home/eva`

Snapshots

Snapshots lassen sich schnell anlegen und "frieren" den Stand des Datasets zu diesem Zeitpunkt ein (readonly). Bei Bedarf wird wieder zu diesem Snapshot zurückgerollt oder einzelne Dateien wiederhergestellt.

Snapshot anlegen `zfs snapshot mypool/ds@snapshotname`
Kürzere Form: `zfs snap mypool/ds@snapshotname`
Rekursive Snapshots über alle Kind-Datasets anlegen:
`zfs snap -r mypool/ds@snapshotname`

Snapshot anzeigen Snapshot eines Datasets anzeigen:
`zfs list -t snap mypool/ds`
Rekursiv Snapshots mit allen Kind-Datasets anzeigen:
`zfs list -rt snap mypool/ds`

Platzverbrauch anzeigen Die `written`-Property zeigt an, wieviele Daten seit dem letzten Snapshot geschrieben wurden. Zusammen mit used und referenced ergibt sich ein guter Überblick über den Platzbedarf:
`zfs list -rt all -o name,used,refer,written mypool`

Snapshots vergleichen Die Unterschiede zwischen dem letzten Snaphot und dem derzeitigen Dataset zeigt das `diff`-Unterkommando an:
`zfs diff mypool/ds@backup`
Diese Tabelle zeigt die Bedeutung der Zeichen:

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

Auf Verzeichnissen/Datasets sind dies Änderungen der Metadaten. Ebenfalls lassen sich zwei Snapshots vergleichen:
`zfs diff mypool/ds@backup1 mypool/ds@backup2`

Snapshots zurückrollen Den aktuellen Stand des Datasets verwerfen und zum letzten Snapshot zurückkehren. Alle seitdem angelegten Daten gehen verloren:
`zfs rollback mypool/ds@backup2`
Zu anderem Snapshot als dem Letzten zurückkehren, entfernt die Snapshots dazwischen ebenfalls:
`zfs rollback -r mypool/ds@backup`

Snapshots mounten Einbinden eines Snapshots als nur-Lese Dataset:
`mount -t zfs mypool/ds@backup /mnt/backup`

Snapshots löschen Zuerst Snapshots mit den Parametern `-nv` anzeigen, da oft rekursiv (`-r`) mehr gelöscht wird, als geplant.
`zfs destroy -rvn mypool/ds@backup`
Ist man sich sicher, dass dies so in Ordnung ist, Parameter `-n` entfernen um die Aktion wirklich durchzuführen.
`zfs destroy -rv mypool/ds@backup`

Klone

Klone lassen sich *nur* aus Snapshots erstellen und sind eine beschreibbare Version des Snapshots. Er enthält alle Daten vom Zeitpunkt des Snapshots, kann aber neue Daten hinzufügen oder bestehende löschen.

Klon erzeugen Aus dem `backup`-Snapshot wird ein Klon `schulung` angelegt:
`zfs clone mypool/ds@backup mypool/schulung` Ein Klon zeigt in der `origin`-Property den Snapshot an, aus dem dieser erstellt wurde.
`zfs get origin mypool/schulung`

Klon unabhängig machen Der Snapshot wird so lange nicht entfernt, als noch ein von ihm abhängiger Klon existiert. Diese Abhängigkeit wird durch das `promote`-Kommando umgekehrt:
`zfs promote mypool/schulung`
Dadurch verschwindet die `origin`-Property und der Snapshot ist jetzt dem Klon zugeordnet.

Klon löschen Klone lassen sich mit `destroy` wie gewöhnliche Datasets löschen:
`zfs destroy mypool/schulung`

Snapshots senden und empfangen

Snapshots lassen sich lokal und über ein Netzwerk an einen anderen Pool zu Datensicherungszwecken übertragen.

Snapshot in Datei schreiben Snapshot in Datei umleiten:
`zfs send mypool/ds@backup > dsbackup`
Diese Datei kann gesichert und später erneut in einen Pool mit `zfs receive` zurückgespielt werden. Mehr Details zum Replikationsvorgang mit `-v` anzeigen:
`zfs send -v mypool/ds@backup > ziel`

Dataset aus Snapshot erzeugen Mit `receive` (oder kurz: `recv`) wird aus einem Snapshot wieder ein Dataset erzeugt, auch auf einem anderen Pool:
`zfs recv mypool/backup < dsbackup`
Mehr Details erhalten mit `-v`: `zfs recv -v mypool/backup < dsbackup`

Replikation ohne Datei Per Pipe können `send` und `recv` ohne Zwischendatei miteinander Daten austauschen:
`zfs send mypool/ds@backup|zfs recv mypool/neu`

Replikation per SSH Snapshots mit SSH an einen anderen Pool im Netzwerk übertragen:
`zfs send poolA/ds@backup|ssh host zfs recv poolB/neu`
Einem unprivilegierten Benutzer mit `zfs allow` die Berechtigung zum `send` bzw. `recv` erteilen, um einen SSH-Login als `root`-Benutzer zu vermeiden.

Benedict Reuschling (`benedict@reuschling.org`), April 2023