

Dauerhafter Schutz mit ZFS Auto Snapshots

Betriebsdokumentation · Proxmox & ZFS

ZFS-Snapshots in Proxmox: Aufbewahrung und Wiederherstellung

Wie wir mit ZFS Auto Snapshot ältere Datenstände erhalten und im Notfall VMs, LXC-Container und NAS-Dateien wiederherstellen.

- [Grundlagen](#)
- [Aufbewahrung](#)
- [Auto Snapshot](#)
- [80-%-Kontrolle](#)
- [SSD-TRIM](#)
- [Wiederherstellung](#)
- [VM klonen](#)
- [Quellen](#)

Was ist ein ZFS-Snapshot?

Ein ZFS-Snapshot ist eine schreibgeschützte Momentaufnahme eines Dateisystems oder ZVOLs. ZFS nutzt **Copy-on-Write**: Änderungen werden in neue Blöcke geschrieben. Der Snapshot hält die Verweise auf die bisherigen Blöcke fest.

Es entsteht keine vollständige Datenkopie. Zusätzlichen Speicher benötigen vor allem ältere Blöcke, die nach Änderungen oder Löschungen nur noch für Snapshots erhalten bleiben. Unveränderte Blöcke werden gemeinsam genutzt.

ZFS-Snapshots und VMware-Snapshots

Unterschied zur klassischen VMware-Snapshot-Technik

Merkmal	ZFS	VMware vSphere / ESXi
Ebene	Dateisystem oder ZVOL	Virtuelle Maschine
Verfahren	Alte Datenblöcke bleiben referenziert.	Neue Schreibzugriffe landen bei klassischen Snapshots in Delta-Dateien.
Arbeitsspeicher	Wird nicht erfasst.	Kann optional erfasst werden.
Löschen	Gibt nicht mehr benötigte Blöcke frei.	Konsolidiert die Delta-Daten; der aktuelle Stand bleibt erhalten.

Klassische VMware-Snapshots sind für kurzfristige Rückkehrpunkte gedacht; Broadcom empfiehlt höchstens 72 Stunden Aufbewahrung. Diese Empfehlung lässt sich nicht auf ZFS-Snapshots übertragen. Die VMware-Umsetzung kann je nach Speicherplattform abweichen.

VMware-Empfehlungen

Unsere Snapshot-Aufbewahrung: ungefähr drei Monate

Die Crontab bestimmt die Ausführungszeitpunkte. Pro Zeitstufe behalten wir eine feste Anzahl von Snapshots:

Intervall	Reihe	Anzahl	Ungefähres Fenster
Alle 15 Minuten	frequent	12	3 Stunden
Stündlich	hourly	96	4 Tage
Täglich	daily	21	3 Wochen
Wöchentlich	weekly	6	6 Wochen
Monatlich	monthly	3	3 Monate

Je neuer der Datenstand, desto feiner die Auswahl. Für heutige Fehler gibt es viele Rückkehrpunkte. Für spät bemerkte Fehler bleiben ältere Tages-, Wochen- und

Monatsstände.

Die Fenster sind Richtwerte, keine garantierten Mindestzeiten. Die tatsächliche Reichweite hängt vom Cron-Zeitpunkt und erfolgreichen Ausführungen ab. Eine bestimmte Dateiversion ist nur wiederherstellbar, wenn mindestens ein noch vorhandener Snapshot sie erfasst hat.

Wie arbeitet ZFS Auto Snapshot?

Cron startet das Programm. Es wählt die vorgesehenen Datenbereiche aus, erstellt einen Snapshot mit Zeitstufe und Zeitstempel und entfernt ältere passende Snapshots nach der eingestellten Anzahl.

Bei der Stundenreihe bleibt beispielsweise mit `--label=hourly --keep=96` nach dem Aufräumen eine Auswahl von 96 Stundenständen. Die Reihen sind unabhängig: Ein Viertelstunden-Snapshot wird nicht später zum Tages- oder Monatssnapshot umgewandelt.

Die verwendeten ZFS-Befehle

Befehl	Aufgabe
<code>zpool status</code>	Poolzustand und laufende Scrubs prüfen.
<code>zfs list</code>	Datenbereiche, Auto-Snapshot-Eigenschaften und vorhandene Snapshots auslesen.
<code>zfs snapshot</code>	Neue Momentaufnahme erstellen; mit <code>-r</code> auch für untergeordnete Datasets.
<code>zfs destroy -d</code>	Alte Snapshots entfernen oder deren Löschung vormerken, wenn Holds oder Clones sie noch benötigen.
<code>zfs get</code>	Optional die seit dem letzten Snapshot geschriebenen Daten prüfen.

Die Auswahl lässt sich über Eigenschaften wie `com.sun:auto-snapshot` und `com.sun:auto-snapshot:hourly` steuern. Diese Beschreibung bezieht sich auf die Linux-Implementierung von

[**zfsonlinux/zfs-auto-snapshot**](#).

Mit zpool list die 80%-Grenze prüfen

Unser Betriebsziel ist, den Pool **unter 80 % Belegung** zu halten. Die Reserve lässt Platz für neue Daten, Snapshot-Wachstum und Änderungen an Clones. 80 % ist eine betriebliche Richtlinie, keine feste ZFS-Funktionsgrenze.

```
zpool list -o name,size,alloc,free,cap,health rpool
```

Entscheidend ist `CAP`: Steht dort `78%`, bleibt wenig Reserve bis zu unserem Grenzwert. Bei `80%` oder mehr reduzieren wir die Belegung. `ALLOC` zeigt den belegten, `FREE` den freien Poolplatz. Der tatsächlich für Datasets nutzbare Platz kann wegen Reservierungen und anderer Faktoren abweichen.

Zur Einordnung prüfen wir zusätzlich, wo Platz belegt wird:

```
zfs list -r -o name,used,avail,usedbysnapshots,usedbydataset rpool
zfs list -t snapshot -r -o name,used,referenced -s creation rpool
zfs list -t volume -r -o name,used,referenced,refreservation rpool
```

`usedbysnapshots` zeigt den von Snapshots gebundenen Platz eines Datasets. Snapshot-Werte dürfen nicht einfach addiert werden, da Blöcke gemeinsam genutzt werden. Bei ZVOLS prüfen wir auch `refreservation`, also reservierten Platz. [OpenZFS: zpool list](#)

Aufbewahrung bei Platzmangel reduzieren

In den bestehenden Cron-Aufrufen ändern wir `--keep`. Die Ausführungszeiten bleiben erhalten. Eine mögliche kürzere Aufbewahrung ist:

Reihe	Bisher	Kürzeres Beispiel	Ungefähres Fenster
frequent	12	8	2 Stunden
hourly	96	48	2 Tage

Reihe	Bisher	Kürzeres Beispiel	Ungefähres Fenster
daily	21	14	2 Wochen
weekly	6	4	4 Wochen
monthly	3	2	2 Monate

Beispiel für eine Benutzer- oder Root-Crontab; den Programmpfad gegebenenfalls anpassen. In `/etc/cron.d/` steht zusätzlich nach den fünf Zeitfeldern der Benutzer `root`. Bestehende Aufgaben ersetzen, keine parallelen doppelten Aufgaben anlegen.

```
*/15 * * * * /usr/sbin/zfs-auto-snapshot --quiet --syslog --label=frequent --keep=8 //
0 * * * * /usr/sbin/zfs-auto-snapshot --quiet --syslog --label=hourly --keep=48 //
0 0 * * * /usr/sbin/zfs-auto-snapshot --quiet --syslog --label=daily --keep=14 //
0 0 * * 0 /usr/sbin/zfs-auto-snapshot --quiet --syslog --label=weekly --keep=4 //
0 0 1 * * /usr/sbin/zfs-auto-snapshot --quiet --syslog --label=monthly --keep=2 //
```

Die Zeitfelder sind ein Beispiel. Bei unserer Installation übernehmen wir die vorhandenen Zeiten, Dataset-Auswahl und sonstigen Optionen und ändern nur die gewünschten Aufbewahrungszahlen. Auto Snapshot räumt jede Reihe beim nächsten erfolgreichen Aufruf auf. Die Monatsreihe wird deshalb nicht sofort durch eine Änderung der Crontab verkürzt.

Für zeitnahes Aufräumen kann ein entsprechender normaler Aufruf außerhalb von Cron erfolgen: zuerst mit `--dry-run` prüfen, dann nach Prüfung ohne diese Option ausführen. Dieser Aufruf erstellt auch einen neuen Snapshot. Für jede betroffene Reihe ist ein eigener Aufruf nötig; er sollte nicht gleichzeitig mit dem regulären Cron-Lauf erfolgen.

```
# Vorschau für die Stundenreihe:
zfs-auto-snapshot --dry-run --label=hourly --keep=48 //
# Nach Prüfung: neuer Snapshot und Aufräumen der Stundenreihe
zfs-auto-snapshot --label=hourly --keep=48 //
# Anschließend Belegung erneut kontrollieren:
zpool list -o name,size,alloc,free,cap,health rpool
```

Eine kürzere Aufbewahrung entfernt Wiederherstellungspunkte dauerhaft. Sie gibt nur Blöcke frei, die danach niemand mehr benötigt. Clones und Holds können alte Blöcke weiter binden; die Freigabe kann verzögert erfolgen. Bleibt der Pool zu voll, müssen zusätzlich aktuelle Daten, Reservierungen und nicht mehr benötigte Wiederherstellungs-Clones geprüft werden. TRIM allein verringert `CAP` nicht.

Warum Snapshots auch bei SSD-TRIM erhalten bleiben

TRIM meldet der SSD, welche Speicherbereiche der Pool nicht mehr benötigt. Die SSD darf deren Inhalt anschließend intern verwerfen.

Blöcke, die ein Snapshot noch benötigt, bleiben für ZFS belegt und werden nicht zum Trimmen freigegeben. Eine im aktuellen Dateisystem gelöschte Datei bleibt dadurch aus einem vorhandenen Snapshot wiederherstellbar.

Erst wenn keine Referenz mehr besteht, können die Blöcke freigegeben und später getrimmt werden. Snapshot-Löschung und TRIM sind getrennte Vorgänge; Auto Snapshot führt keinen eigenen Pool-TRIM aus.

Wir bewahren Snapshots für die Wiederherstellung auf. Unbegrenzte Aufbewahrung zur Vermeidung von TRIM bietet keinen pauschalen Vorteil für SSD-Leistung oder Lebensdauer und bindet Speicher.

[OpenZFS-Dokumentation zu TRIM](#)

Proxmox: Rollback, Snapshot-Verzeichnis oder Clone?

Wir verwenden ZFS Auto Snapshot. Unsere Wiederherstellung erfolgt auf ZFS-Ebene; Proxmox stoppt und startet die Gäste.

Ziel	Geeigneter Weg
Einzelne LXC- oder NAS-Dateien retten	Aus dem Snapshot-Verzeichnis herauskopieren.
Alten Stand untersuchen oder reparieren	Separaten beschreibbaren Clone erstellen.
Produktiven Speicher vollständig zurücksetzen	Gast stoppen, Rollback durchführen, Gast starten.

Rollback einer VM oder eines LXC-Containers

Die folgenden Befehle sind Vorlagen. IDs, Speicherbereiche und Snapshot-Namen müssen zum Gast passen. Vorher wird sichergestellt, dass beispielsweise HA den Gast nicht automatisch neu startet.

VM mit ZVOL:

```
qm stop 100
zfs rollback rpool/data/vm-100-disk-0@SNAPSHOTNAME
# Erst nach erfolgreichem Rollback starten:
qm start 100
```

LXC mit Dateisystem-Dataset:

```
pct stop 200
zfs rollback rpool/data/subvol-200-disk-0@SNAPSHOTNAME
# Erst nach erfolgreichem Rollback starten:
pct start 200
```

Rollback verwirft spätere Änderungen. `qm stop` und `pct stop` stoppen unmittelbar. Anwendungen werden vorher möglichst geordnet beendet. Bei mehreren Festplatten oder Datasets müssen alle benötigten Bereiche auf zusammenpassende Stände gebracht werden; erst danach startet der Gast.

Ohne Zusatzoption erlaubt ZFS nur den neuesten Snapshot. Für einen älteren Stand ermöglicht `zfs rollback -r` das Zurücksetzen, löscht dabei aber neuere Snapshots und Bookmarks des betroffenen Speicherbereichs. Untergeordnete Datasets werden nicht automatisch zurückgesetzt. [OpenZFS: Rollback](#)

Snapshot-Verzeichnis für LXC und NAS

Bei Dateisystem-Datasets erreichen wir die alten Dateien auf dem Host unter:

```
<MOUNTPOINT>/zfs/snapshot/<SNAPSHOTNAME>/
```

Der Inhalt ist schreibgeschützt. Wir kopieren benötigte Dateien zunächst in ein separates Wiederherstellungsverzeichnis. Beim Zurückkopieren bleiben Eigentümer und Rechte erhalten, insbesondere bei unprivilegierten LXC-Containern. Vor dem Ersetzen aktiver Anwendungsdateien stoppen wir die betroffene Anwendung.

`snapdir=visible` macht `.zfs` in Verzeichnislisten sichtbar. Bei `hidden` funktioniert normalerweise der direkte Pfad. Ein ZVOL für eine VM besitzt kein solches Dateiverzeichnis.

Clones für eine separate Wiederherstellung

Ein ZFS-Clone ist ein beschreibbarer Speicherbereich aus einem Snapshot. Für LXC oder NAS mounten wir ihn separat. Einen ZVOL-Clone können wir als zusätzliche Festplatte einer Wiederherstellungs-VM zuordnen und darin auf das Gast-Dateisystem zugreifen.

Der aktuelle Datenbestand bleibt erhalten. Clones benötigen anfangs kaum zusätzlichen Platz, eigene Änderungen belegen neue Blöcke. Der Ursprungssnapshot bleibt erforderlich, solange der Clone davon abhängt.

Ein manueller ZFS-Clone ist noch kein startfertiger Proxmox-Gast. Speicherzuordnung und Konfiguration müssen eingerichtet werden. Ein daraus gestarteter Testgast erhält zunächst ein getrenntes Netzwerk, damit keine doppelten IP-Adressen oder gleichzeitig aktiven Dienste entstehen.

OpenZFS: Snapshot-Zugriff und Clones

Grenzen der Wiederherstellung

Auto-Snapshots erfassen die ausgewählten Speicherbereiche. RAM, separat gespeicherte Proxmox-Gastkonfigurationen und externe Mounts gehören nicht automatisch dazu. Ein konsistenter ZFS-Snapshot garantiert außerdem keine Anwendungskonsistenz einer laufenden Datenbank.

Bei Verlust des Pools brauchen wir ein separates Backup. ZFS Auto Snapshot allein repliziert keine Daten. Eine Übertragung mit `zfs send` und `zfs receive` auf ein anderes System muss zusätzlich eingerichtet werden.

Alle VM-Disks nach rpool/clone klonen und eine neue VM anlegen

Wir erstellen aus den Auto-Snapshots beschreibbare ZVOL-Clones und ordnen sie einer neuen Proxmox-VM zu. Die produktive VM und ihre aktuellen Disks bleiben erhalten. Die folgenden Schritte sind eine **Vorlage für VM 100 → VM 900** auf dem Host mit `rpool`; VMID 900 und die Zielnamen müssen frei sein.

1. Alle Disks und den Wiederherstellungsstand erfassen

```
qm config 100
zfs list -t volume -r rpool
zfs list -t snapshot -r -o name,creation -s creation rpool/data
```

Aus `qm config` übernehmen wir jede Festplatten-Zuordnung, einschließlich vorhandener `efidisk0` und `tpmstate0`. Speicherpfade lassen sich mit `pvesm path <VOLUME-ID>` auflösen. CD-ROM-ISOs werden nicht als ZVOL geklont. Eine vorhandene Cloud-Init-Disk und zusätzliche ZVOLs müssen ebenfalls berücksichtigt werden.

Für jede Disk muss der gewählte Snapshot existieren. Derselbe Snapshot-Name bedeutet bei einzeln erstellten Snapshots nicht zwingend denselben Erstellungszeitpunkt. Für Datenbanken über mehrere Disks benötigen wir einen koordinierten, anwendungskonsistenten Stand oder müssen nach dem Start mit Wiederherstellungsarbeiten rechnen.

Beispiel-Zuordnung – anhand der echten VM-Konfiguration prüfen

Anschluss	Quell-ZVOL	Ziel-ZVOL
scsi0	rpool/data/vm-100-disk-0	rpool/clone/vm-900-disk-0
scsi1	rpool/data/vm-100-disk-1	rpool/clone/vm-900-disk-1
efidisk0	rpool/data/vm-100-disk-2	rpool/clone/vm-900-disk-2

Anschluss	Quell-ZVOL	Ziel-ZVOL
tpmstate0	rpool/data/vm-100-disk-3	rpool/clone/vm-900-disk-3

2. ZFS-Bereich und Proxmox-Storage einrichten

Einmalig anlegen, wenn noch nicht vorhanden. `rpool/clone` ist der ZFS-Bereich; `zfs-clone` ist die Storage-ID in Proxmox. Hier ist mit „Datastore“ ein Proxmox-VE-Storage gemeint, kein Proxmox-Backup-Server-Datastore.

```
zfs create -o mountpoint=none -o com.sun:auto-snapshot=false rpool/clone

# PVE-NODENAME durch den tatsächlichen Proxmox-Knotennamen ersetzen:
pvesm add zfspool zfs-clone --pool rpool/clone --content images --sparse 1 --nodes
PVE-NODENAME
pvesm status
```

Die Beschränkung auf den richtigen Knoten ist besonders im Cluster wichtig: Dieser Storage liegt lokal. `com.sun:auto-snapshot=false` schließt den Wiederherstellungsbereich über die geerbte Eigenschaft von der normalen Auto-Snapshot-Rotation aus.

3. Für jede Disk einen ZFS-Clone erstellen

`SNAPSHOTNAME` durch den tatsächlich vorhandenen Auto-Snapshot-Namen ersetzen. Die vier Befehle decken alle Disks unseres Beispiels ab. Hat die VM weitere ZVOLs, wird die Liste entsprechend erweitert.

```
zfs clone -o refreservation=none -o com.sun:auto-snapshot=false rpool/data/vm-100-
disk-0@SNAPSHOTNAME rpool/clone/vm-900-disk-0
zfs clone -o refreservation=none -o com.sun:auto-snapshot=false rpool/data/vm-100-
disk-1@SNAPSHOTNAME rpool/clone/vm-900-disk-1
zfs clone -o refreservation=none -o com.sun:auto-snapshot=false rpool/data/vm-100-
disk-2@SNAPSHOTNAME rpool/clone/vm-900-disk-2
```

```
zfs clone -o refreservation=none -o com.sun:auto-snapshot=false rpool/data/vm-100-disk-3@SNAPSHOTNAME rpool/clone/vm-900-disk-3

zfs list -t volume -r -o name,origin,used,referenced,refreservation rpool/clone
pvesm list zfs-clone --vmid 900
```

`refreservation=none` vermeidet eine vollständige Platzreservierung pro Clone. Eigene Schreibzugriffe brauchen trotzdem echten freien Poolplatz. Die Namen `vm-900-disk-N` ordnen die Volumes der neuen VMID zu. ZFS-Clones müssen innerhalb desselben Pools bleiben.

4. Neue VM anlegen und geklonte Disks anschließen

Dieses Beispiel setzt eine UEFI-VM mit Q35, VirtIO-SCSI, einer EFI-Disk des Typs 4m und TPM 2.0 voraus. BIOS, Maschinenmodell, Controller, Disk-Anschlüsse, EFI-/TPM-Version und Disk-Optionen müssen zur Quell-VM passen. Bei einer BIOS-VM entfällt beispielsweise die EFI-Disk. Die VM wird zunächst ohne Netzwerkkarte angelegt.

```
qm create 900 --name recovery-vm100 --memory 4096 --cores 2 --bios ovmf --machine q35
--scsihw virtio-scsi-pci --onboot 0
qm set 900 --scsi0 zfs-clone:vm-900-disk-0
qm set 900 --scsi1 zfs-clone:vm-900-disk-1
qm set 900 --efidisk0 zfs-clone:vm-900-disk-2,efitype=4m,pre-enrolled-keys=1
qm set 900 --tpmstate0 zfs-clone:vm-900-disk-3,version=v2.0
qm set 900 --boot 'order=scsi0'
qm config 900
# Erst nach Prüfung aller Zuordnungen starten:
qm start 900
```

Die Befehle verbinden bestehende Clones; sie importieren oder kopieren keine neuen Disk-Inhalte. Die aktuelle Konfiguration der Quell-VM dient nur als Vorlage: Bei alten Datenständen kann die damalige Hardware-Konfiguration nötig sein. Bei verschlüsselten Gästen können außerdem Wiederherstellungsschlüssel erforderlich sein.

Über die Proxmox-Konsole prüfen wir den Start und die Daten. Für Dateirettung kann stattdessen eine vorhandene Wiederherstellungs-VM die geklonten Daten-Disks als zusätzliche Laufwerke erhalten. Erst wenn nötig, verbinden wir den Testgast mit einem isolierten Netz.

5. Abhängigkeiten und Speicherplatz nach der Rettung

Der Clone bleibt von seinem Ursprungssnapshot abhängig. Auto Snapshot kann diesen deshalb nicht sofort vollständig entfernen. Nicht mehr benötigte Recovery-VMs und ihre Clone-Disks werden nach geprüfter Datenrettung über Proxmox entfernt; anschließend kontrollieren wir `rpool/clone` und `zpool list`. Soll die Recovery-VM dauerhaft bestehen, planen wir eine unabhängige vollständige Kopie statt eines dauerhaften Recovery-Clones.

`rpool/clone` nutzt denselben Pool. Es ist eine separate Arbeitskopie, aber kein unabhängiges Backup und kein zusätzlicher physischer Speicher.

Referenzen: [Proxmox-ZFS-Storage](#), [pvesm](#), [qm](#), [ZFS-Clone](#).

Weiterführende Dokumentation

- [OpenZFS: Snapshots, Clones und Zugriff auf ältere Dateien](#)
- [OpenZFS: SSD-TRIM](#)
- [ZFS Auto Snapshot für Linux](#)
- [Proxmox: VM-Verwaltung mit qm](#)
- [Proxmox: LXC-Verwaltung mit pct](#)
- [OpenZFS: Replikation mit zfs send](#)

ZFS Auto Snapshot – Kurzreferenz

Proxmox VE | zpool list | VM/LXC Rollback | Retention anpassen

ZFS Auto Snapshot arbeitet auf ZFS-Ebene. Proxmox stoppt und startet den Gast.

1 Pool schnell prüfen

```
zpool list -o name,size,alloc,free,cap,health rpool
```

- CAP = Belegung des Pools
- Betriebsziel: unter 80 % bleiben
- ALLOC = belegt, FREE = frei

2 VM vorbereiten: Disks + Snapshotnamen prüfen



Beispiel: VM-ID

2.1 Disks der VM auflisten

Bei VMs mit mehreren Disks müssen alle relevanten Disks identifiziert und später jeweils auf den passenden Snapshot zurückgesetzt werden.

```
qm config 100 | egrep '^(scsi|sata|ide|virtio)[0-9]+:'
```



Ausgabe (Beispiel)

```
scsi0: local-zfs:vm-100-disk-0,size=32G
scsi1: local-zfs:vm-100-disk-1,size=100G
ide2:  local:iso/debian.iso,media=cdrom
```

Nur die tatsächlichen VM-Disks (z. B. scsi0, scs: müssen auf Snapshot zurückgesetzt werden.

2.2 Verfügbare Snapshotnamen der VM-Disks anzeigen

Alle vorhandenen Snapshots der VM-Disks auflisten und den gewünschten Zeitpunkt (Snapshotnamen) auswählen.

```
zfs list -t snapshot -o name -s creation | grep 'vm-100-disk-'
```



Ausgabe (Beispiel)

```
rpool/data/vm-100-disk-0@zfs-auto-snap_hourly-2024-10-1
rpool/data/vm-100-disk-0@zfs-auto-snap_daily-2024-10-1
rpool/data/vm-100-disk-1@zfs-auto-snap_hourly-2024-10-1
```

Für jede betroffene VM-Disk muss später ein passender Snapshot verwendet werden.

3 Rollback einer VM (Stop → Rollback → Start)



Beispiel: VM-ID

Den Gast anhalten, alle relevanten VM-Disks auf den gewünschten Snapshot zurücksetzen und den Gast anschließend wieder starten.

```
qm stop 100
zfs rollback rpool/data/vm-100-disk-0@SNAPSHOTNAME
zfs rollback rpool/data/vm-100-disk-1@SNAPSHOTNAME # ggf. weitere Disks
qm start 100
```



Bedeutung der Schritte

- **stop** = Gast vor dem Eingriff anhalten
- **rollback** = Speicherstand auf Snapshot zurücksetzen
- **start** = Gast erst nach erfolgreichem Rollback wieder starten



Hinweis

Bei mehreren Disks müssen alle relevanten VM-Disks auf den gleichen passenden Snapshot zurückgesetzt werden.

4 Rollback eines LXC-Containers (Dataset)



Beispiel: CT-ID

Container anhalten, Dataset auf den gewünschten Snapshot zurücksetzen und anschließend wieder starten.

```
pct stop 200
zfs rollback rpool/data/subvol-200-disk-0@SNAPSHOTNAME
pct start 200
```



Hinweis

- Bei mehreren Datasets oder Bind-Mounts müssen alle relevanten Datasets auf den gleichen Snapshot zurückgesetzt werden.
- Vorgehen ist identisch: stop → rollback → start

5 Retention anpassen: wo und wie

Wo werden die Retention-Werte konfiguriert?

- Häufige Snapshots (z. B. stündlich, alle 5 Minuten) werden über die Datei /etc/cron.d/zfs-auto-snapshot gesteuert.
- Tägliche Snapshots: Skript(e) in /etc/cron.daily/
- Wöchentliche Snapshots: Skript(e) in /etc/cron.weekly/
- Monatliche Snapshots: Skript(e) in /etc/cron.monthly/

Beispiele: Kürzere Keep-Werte in den jeweiligen Jobs

Die Retention wird über den Parameter --keep in den entsprechenden Cron-Jobs gesetzt. Beispielbefehle:

```
/usr/sbin/zfs-auto-snapshot --label=hourly --keep=48 //
/usr/sbin/zfs-auto-snapshot --label=daily --keep=14 //
/usr/sbin/zfs-auto-snapshot --label=weekly --keep=4 //
/usr/sbin/zfs-auto-snapshot --label=monthly --keep=2 //
```



Vorher testen (Dry-Run)

```
zfs-auto-snapshot --dry-run --label=hourly --keep=48 //
Danach normal ausführen und Belegung erneut prüfen.
```

6 Wichtige Hinweise



- Rollback verwirft alle Änderungen nach dem Snapshot.
- Für ältere Stände kann "zfs rollback -r" nötig sein.
- Dabei werden neuere Snapshots und Bookmarks dieses Datasets entfernt.
- Bei mehreren Disks oder Datasets alle auf den passenden Stand bringen.
- HA/Autostart beachten.

Ein Klonskript zum Testen (KI-Generiert) könnt ihr hier mal probieren

```
#!/usr/bin/env python3
```

```
"""Clone a local Proxmox VM from external ZFS snapshots; no source rollback.
```

Run as root on the source node. Python 3 and the Proxmox/ZFS CLI are required.

Example: `python3 proxmox-zfs-clone.py 100 900 zfs-auto-snap_daily-2026-10-08-0000`

Use `--dry-run` to inspect without changing anything. The new VM stays stopped and has no network interfaces. No Proxmox-internal snapshot is required.

```
"""
```

```
import argparse
```

```
import fcntl
```

```
import json
```

```
import os
```

```
import re
```

```
import shlex
```

```
import shutil
```

```
import socket
```

```
import subprocess
```

```
import sys
```

```
import uuid
```

```
from dataclasses import dataclass
```

```
class CloneError(Exception):
```

```
    pass
```

```
def read(*command):
```

```
    result = subprocess.run(command, text=True, capture_output=True)
```

```
    if result.returncode:
```

```
        raise CloneError(f"{' '.join(command[:2])}: {result.stderr.strip() or result.stdout.strip()}")
```

```
    return result.stdout.strip()
```

```
def api(path, *options):
```

```
    return json.loads(read("pvesh", "get", path, *options, "--output-format", "json"))
```

```

def change(command, dry_run):
    print(("VORSCHAU: " if dry_run else "> ") + shlex.join(command), flush=True)
    if not dry_run:
        subprocess.run(command, check=True)

def config(vmid):
    result = {}
    for line in read("qm", "config", str(vmid), "--current", "1").splitlines():
        key, sep, value = line.partition(": ")
        if sep:
            result[key] = value
    return result

@dataclass
class Disk:
    slot: str
    source: str
    target: str
    options: list
    creation: int

def plan_disks(source_config, target_id, snapshot, dataset):
    disks = []
    skipped = []
    sources = set()
    for slot, value in sorted(source_config.items()):
        if not re.fullmatch(r"(?:scsi|sata|ide|virtio)\d+|efidisk0|tpmstate0", slot):
            continue
        parts = value.split(",")
        volume = parts[0].removeprefix("file=")
        options = parts[1:]
        if volume in {"none", "cdrom"} or ":iso/" in volume:
            skipped.append(slot)
            continue
        if ":" not in volume:
            raise CloneError(f"{slot}: kein verwaltetes ZFS-Volume ({volume}).")

```

```

    path = read("pvesm", "path", volume)
    if not path.startswith("/dev/zvol/"):
        raise CloneError(f"{slot}: {volume} ist kein lokales ZVOL. Alle Disks
müssen auf ZFS liegen.")
    zvol = path[len("/dev/zvol/"):]
    if zvol.split("/", 1)[0] != dataset.split("/", 1)[0]:
        raise CloneError(f"{slot}: {zvol} liegt nicht im Ziel-Pool. ZFS-Clones
können Pools nicht wechseln.")
    if zvol in sources:
        raise CloneError(f"Das ZVOL {zvol} ist mehrfach angeschlossen; Zuordnung
manuell prüfen.")
    sources.add(zvol)
    snap = f"{zvol}@{snapshot}"
    # Missing snapshots abort here, before any mutation.
    creation = int(read("zfs", "get", "-Hp", "-o", "value", "creation", snap))
    kind = read("zfs", "get", "-H", "-o", "value", "type", zvol)
    if kind != "volume":
        raise CloneError(f"{zvol} ist kein ZVOL.")
    # No source replication flags or reservations are transferred.
    options = [p for p in options if p.split("=", 1)[0] not in
                {"size", "replicate", "shared", "import-from"}]
    disks.append(Disk(slot, zvol, f"{dataset}/vm-{target_id}-disk-{len(disks)}",
                      options, creation))

if not disks:
    raise CloneError("Keine klonbaren ZVOL-Disks gefunden.")
return disks, skipped

```

```

HARDWARE = {
    "acpi", "agent", "arch", "balloon", "bios", "bootdisk", "cores", "cpu",
    "hotplug", "keyboard", "kvm", "localtime", "machine", "memory", "numa",
    "ostype", "scsihw", "sockets", "tablet", "tdf", "vga", "smbios1",
    "vmgenid", "serial0", "serial1", "serial2", "serial3",
}

```

```

def creation_command(args, source_config, slots):
    command = ["qm", "create", str(args.target), "--name", args.name,
               "--onboot", "0", "--description",

```

```

        f"ZFS-Recovery von VM {args.source}; Snapshot {args.snapshot}; ohne
Netzwerk.")]
    for key in sorted(HARDWARE):
        if key not in source_config:
            continue
        value = source_config[key]
        if key == "smbios1":
            # Keep hardware identifiers for guest compatibility; assign a new VM UUID.
            fields = [p for p in value.split(",") if not p.startswith("uuid=")]
            value = ",".join([f"uuid={uuid.uuid4()}"] + fields)
        elif key == "vmgenid":
            value = str(uuid.uuid4())
        elif key.startswith("serial") and value != "socket":
            raise CloneError(f"{key}: physische serielle Schnittstelle wird nicht
automatisch übernommen.")
        elif key == "bootdisk" and value not in slots:
            continue
        command += [f"--{key}", value]
    boot = source_config.get("boot", "")
    if boot.startswith("order="):
        devices = [p for p in boot[6:].split(";") if p in slots]
        if devices:
            command += ["--boot", "order=" + ";".join(devices)]
    elif boot:
        command += ["--boot", boot]
    return command

def main(args):
    for tool in ("qm", "pvesh", "pvesm", "zfs", "zpool"):
        if not shutil.which(tool):
            raise CloneError(f"{tool} fehlt. Auf dem Proxmox-Quellknoten ausführen.")
    if os.geteuid() != 0:
        raise CloneError("Auf dem Proxmox-Knoten als root ausführen (auch für die
Vorschau).")
    # Serializes this script's local executions, not other PVE or ZFS operations.
    with open("/run/lock/proxmox-zfs-clone.lock", "a") as lock:
        try:
            fcntl.flock(lock, fcntl.LOCK_EX | fcntl.LOCK_NB)

```

```

except BlockingIOError:
    raise CloneError("Ein anderer Lauf dieses Scripts ist bereits aktiv.")
execute(args)

def execute(args):
    resources = api("/cluster/resources", "--type", "vm")
    if any(int(r["vmid"]) == args.target for r in resources):
        raise CloneError(f"VMID {args.target} ist im Cluster bereits belegt.")
    source = next((r for r in resources if int(r["vmid"]) == args.source), None)
    if not source or source.get("type") != "qemu":
        raise CloneError(f"VM {args.source} nicht gefunden oder kein QEMU-Gast.")
    if source["node"] != args.node:
        raise CloneError(f"Quell-VM liegt auf {source['node']}; Script dort
ausführen.")
    source_config = config(args.source)
    if source_config.get("lock"):
        raise CloneError("Die Quell-VM ist gesperrt; laufende Operation abwarten.")
    # Hidden storage or passthrough in custom arguments cannot be safely inferred.
    if any(k in source_config for k in ("args", "amd-sev", "intel-tdx")):
        raise CloneError("Spezielle QEMU-Argumente/Confidential-VM-Konfiguration:
manuelle Wiederherstellung nötig.")
    disks, skipped = plan_disks(source_config, args.target, args.snapshot,
args.dataset)
    command = creation_command(args, source_config, {d.slot for d in disks})
    pool = args.dataset.split("/", 1)[0]
    capacity = int(read("zpool", "list", "-Hp", "-o", "capacity", pool).rstrip("%"))
    print(f"Pool {pool}: {capacity}% belegt. Ziel: unter 80%.")
    if capacity >= 80:
        print("HINWEIS: Pool hat wenig Reserve; Clone-Schreibzugriffe benötigen
zusätzlichen Platz.")
    datasets = set(read("zfs", "list", "-H", "-o", "name", "-r", pool).splitlines())
    if any(d.target in datasets for d in disks):
        raise CloneError("Mindestens ein Ziel-ZVOL existiert bereits; nichts wird
überschrieben.")
    if args.dataset in datasets:
        if read("zfs", "get", "-H", "-o", "value", "type", args.dataset) !=
"filesystem":
            raise CloneError("Zielbereich ist kein ZFS-Dateisystem.")

```

```

        if read("zfs", "get", "-H", "-o", "value", "com.sun:auto-snapshot",
args.dataset) != "false":
            raise CloneError(f"Vorhandenes {args.dataset}: com.sun:auto-snapshot muss
false sein.")
        storage = next((s for s in api("/storage") if s["storage"] == args.storage), None)
        if storage:
            nodes = storage.get("nodes", "")
            if isinstance(nodes, str):
                nodes = nodes.split(",") if nodes else []
            if (storage.get("type") != "zfspool" or storage.get("pool") != args.dataset
                or "images" not in storage.get("content", "").split(",")
                or int(storage.get("disable", 0)) or int(storage.get("shared", 0))
                or (nodes and args.node not in nodes)):
                raise CloneError("Vorhandener Ziel-Storage passt nicht zum lokalen Clone-
Bereich.")
            print(f"VM {args.source} -> VM {args.target}; Snapshot: {args.snapshot}")
            for d in disks:
                print(f"  {d.slot}: {d.source} -> {d.target}")
            if skipped:
                print("Nicht übernommen (ISO/leeres CD-Laufwerk): " + ", ".join(skipped))
            print("Netzwerk, Passthrough, Hooks und unused-Volumes werden nicht übernommen.")
            if any("cloudinit" in source_config[d.slot].split(",")[0] for d in disks):
                print("Cloud-Init-ZVOL wird als statischer Snapshot-Inhalt geklont; keine
Cloud-Init-Neugenerierung.")
            if len({d.creation for d in disks}) > 1:
                print("HINWEIS: Snapshot-Zeitpunkte der Disks unterscheiden sich;
Anwendungskonsistenz prüfen.")
            print("Hardware-Vorlage ist die aktuelle Quell-Konfiguration, nicht die
historische Snapshot-Konfiguration.")
            held = []
            created = []
            tag = "recovery-clone-" + uuid.uuid4().hex
            try:
                # Holds prevent auto-snapshot pruning between validation and cloning.
                for d in disks:
                    snap = f"{d.source}@{args.snapshot}"
                    change(["zfs", "hold", tag, snap], args.dry_run)
                    if not args.dry_run:
                        held.append(snap)

```

```

        if config(args.source) != source_config:
            raise CloneError("Quell-Konfiguration hat sich während der Prüfung
geändert; erneut ausführen.")
        if args.dataset not in datasets:
            change(["zfs", "create", "-o", "mountpoint=none", "-o",
                    "com.sun:auto-snapshot=false", args.dataset], args.dry_run)
        if not storage:
            change(["pvesm", "add", "zfspool", args.storage, "--pool", args.dataset,
                    "--content", "images", "--sparse", "1", "--nodes", args.node],
args.dry_run)
        for d in disks:
            change(["zfs", "clone", "-o", "refreservation=none", "-o", "readonly=off",
                    "-o", "com.sun:auto-snapshot=false",
f"{d.source}@{args.snapshot}",
                    d.target], args.dry_run)
            if not args.dry_run:
                created.append(d.target)
        change(command, args.dry_run)
        for d in disks:
            volume = f"{args.storage}:{d.target.rsplit('/', 1)[1]}"
            specification = ",".join([volume] + d.options)
            change(["qm", "set", str(args.target), f"--{d.slot}", specification],
args.dry_run)
        if not args.dry_run:
            print(read("qm", "config", str(args.target)))
            print(f"{'Vorschau abgeschlossen' if args.dry_run else 'Fertig'}: VM
{args.target} bleibt gestoppt, ohne Netzwerk.")
            print(f"Nach Prüfung starten: qm start {args.target}")
            print("Clones binden ihre Ursprungssnapshots. Nach der Rettung Recovery-VM und
Disks aufräumen.")
    except (Exception, KeyboardInterrupt):
        if not args.dry_run:
            print("ABBRUCH: Bereits erstellte Ressourcen bleiben zur Prüfung
erhalten.", file=sys.stderr)
            print(f"Ziel-VM prüfen: qm config {args.target}", file=sys.stderr)
            for target in created:
                print(f"Erstellter Clone: {target}", file=sys.stderr)
        raise
    finally:

```

```

        for snap in held:
            try:
                change(["zfs", "release", tag, snap], False)
            except (OSError, subprocess.CalledProcessError):
                print(f"Hold manuell lösen: zfs release {shlex.quote(tag)} {shlex.quote(snap)}", file=sys.stderr)

def arguments():
    parser = argparse.ArgumentParser(description=__doc__,
formatter_class=argparse.RawDescriptionHelpFormatter)
    parser.add_argument("source", type=int, help="Quell-VMID")
    parser.add_argument("target", type=int, help="Freie Ziel-VMID")
    parser.add_argument("snapshot", help="Snapshot-Name ohne Dataset und ohne @")
    parser.add_argument("--name", help="Name der neuen VM")
    parser.add_argument("--dataset", default="rpool/clone", help="ZFS-Zielbereich (Standard: rpool/clone)")
    parser.add_argument("--storage", default="zfs-clone", help="Proxmox-Storage-ID (Standard: zfs-clone)")
    parser.add_argument("--node", default=socket.gethostname().split(".")[0], help="Lokaler Proxmox-Knotenname")
    parser.add_argument("--dry-run", action="store_true", help="Nur prüfen und Befehle anzeigen; keine Änderungen")
    args = parser.parse_args()
    if not (100 <= args.source <= 999999999 and 100 <= args.target <= 999999999) or args.source == args.target:
        parser.error("Unterschiedliche VMIDs zwischen 100 und 999999999 angeben.")
    if not re.fullmatch(r"[A-Za-z0-9_.-]+", args.snapshot):
        parser.error("Snapshot-Name ohne @, / oder Leerzeichen angeben.")
    if not re.fullmatch(r"[A-Za-z][A-Za-z0-9_-]*(?:/[A-Za-z0-9_.-]+)", args.dataset):
        parser.error("Ziel muss ein Dataset unterhalb eines Pools sein, z. B. rpool/clone.")
    if not re.fullmatch(r"[A-Za-z][A-Za-z0-9_-]*", args.storage):
        parser.error("Ungültige Storage-ID.")
    args.name = args.name or f"recovery-vm{args.source}-{args.target}"
    if not re.fullmatch(r"[A-Za-z0-9][A-Za-z0-9_-]*", args.name):
        parser.error("VM-Name ohne Leerzeichen oder Sonderzeichen angeben.")
    return args

```

```
if __name__ == "__main__":  
    try:  
        main(arguments())  
    except (CloneError, subprocess.CalledProcessError, OSError, ValueError) as error:  
        print(f"FEHLER: {error}", file=sys.stderr)  
        sys.exit(1)  
    except KeyboardInterrupt:  
        print("Abgebrochen.", file=sys.stderr)  
        sys.exit(130)
```

Stand: 8. Oktober 2026 · Betriebsstrategie: ZFS Auto Snapshot mit fünf Aufbewahrungsstufen.

Version #3

Erstellt: 2026-10-08 14:41:05 UTC von Christian Zengel (sysops GmbH)

Zuletzt aktualisiert: 2026-10-08 16:10:06 UTC von Christian Zengel (sysops GmbH)