

Proxmox Vergleich der Storage-Varianten vs. ZFS zu Fuß

Proxmox VE Storage im Vergleich: Ceph, ZFS, SAN/iSCSI und sichere ZFS-Replikation

Stand: 31. August 2026

PROXMOX GUI
ZFS • CEPH • SAN
VS. ZFS PER SCRIPT **LIVE • 31.08.2026**

```
root@storage:~# zpool status
pool: tank
state: ONLINE
scan: scrub repaired 0B in 00:00:12 with 0 errors on Mon Aug 31 10:00:00 2
config:
NAME                STATE  READ  WRITE  CKSUM
tank                ONLINE  0     0     0
raidz2-0            ONLINE  0     0     0
/dev/nvme0n1         ONLINE  0     0     0
/dev/nvme1n1         ONLINE  0     0     0
/dev/nvme2n1         ONLINE  0     0     0
/dev/nvme3n1         ONLINE  0     0     0
/dev/nvme4n1         ONLINE  0     0     0
errors: No known data errors

root@storage:~# zfs list -o name,used,avail,refer,mountpoint
NAME                USED    AVAIL    REFER  MOUNTPOINT
tank                2.35T   12.4T    256K   /tank
tank/vm             1.12T   12.4T    1.12T   /tank/vm
tank/backup          812G    12.4T    812G    /tank/backup
tank/data            449G    12.4T    449G    /tank/data

root@storage:~# zfs snapshot tank/vm@backup-20260831
root@storage:~# zfs send -R tank/vm@backup-20260831 | zfs receive -F backup/vm
root@storage:~#
```

[Hier der anschliessende YouTube Check zum Artikel](#)

Welche Storage-Lösung passt zu Proxmox VE: Ceph RBD, lokales ZFS oder ein SAN mit iSCSI und klassischem LVM? Dieser Praxisvergleich erklärt die wichtigsten Unterschiede bei Hochverfügbarkeit, Snapshots, Live-Migration, Thin Provisioning und Wiederherstellung. Anschließend zeigen wir, wie automatische ZFS-Snapshots, Pull-Replikation mit `zsync`, GUID-

Kontrolle mit `checkzfs.py` und der Miyagi-Workflow eine mehrstufige Backup- und Replikationsstrategie bilden.

Kurzantwort: Ceph, ZFS oder iSCSI?

- **Lokales ZFS** eignet sich besonders für einzelne Proxmox-Server und kleinere Cluster. Es bietet Prüfsummen, Snapshots, Clones und eine einfache Replikation zwischen ZFS-Systemen.
- **Ceph RBD** ist für verteilten, hochverfügbaren Shared Storage in größeren Proxmox-Clustern gedacht. Ceph repliziert Daten intern und vermeidet einen einzelnen Storage-Kopf.
- **iSCSI mit LVM thick** ist sinnvoll, wenn bereits ein professionelles SAN vorhanden ist. Der Storage ist gemeinsam nutzbar, die eigentliche Redundanz und Hochverfügbarkeit liefert jedoch das SAN.
- **ZFS per Script** ersetzt die Proxmox-Oberfläche nicht. Es erweitert sie um feinere Snapshot-Aufbewahrung, Pull-Replikation, unabhängige Kontrolle und automatisierte Wiederherstellungswege.

Inhalt

1. [Ceph, ZFS und iSCSI im Vergleich](#)
2. [Proxmox-Funktionen und Storage-Abhängigkeiten](#)
3. [Proxmox GUI versus ZFS-Kommandozeile](#)
4. [Automatische ZFS-Snapshots](#)
5. [Snapshots, Rollbacks, Clones und Snapdirs](#)
6. [Replace und RAIDZ-Expansion](#)
7. [Spares, L2ARC, SLOG und Special Devices](#)
8. [ZFS Pull-Replikation mit zsync](#)
9. [Replikation mit checkzfs.py kontrollieren](#)
10. [Trojanerresistenter Miyagi-Workflow](#)

Proxmox Storage: Ceph RBD, lokales ZFS und iSCSI/LVM im Vergleich

Funktion oder Eigenschaft	Ceph RBD	Lokales ZFS	iSCSI + LVM thick
Gemeinsamer Cluster-Storage	Ja	Nein	Ja
Verteilte Speicherung	Ja	Nein	Nur wenn das SAN sie bereitstellt
Kein einzelner Storage-Kopf	Ja, bei korrektem Clusteraufbau	Nein	Abhängig vom SAN
Thin Provisioning	Ja	Optional mit <code>sparse</code>	Nein
Feste Platzreservierung	Steuerbar	Über ZFS-Reservation	Standard
Snapshots	Ja	Ja	Für VMs ab Proxmox VE 9 über Volume Chains
Linked Clones	Ja	Ja	Nein
Full Clones	Ja	Ja	Ja
VM- und LXC-Datenträger	Ja	Ja	Ja, über LVM
Live-Migration ohne Kopieren der VM-Disk	Ja	Nein	Ja
Migration mit Storage-Kopie	Ja	Ja	Ja
Storage-Replikation	Ceph-intern	Asynchron zwischen ZFS-Systemen	Üblicherweise SAN-intern
Selbstheilung	Verteilt im Ceph-Cluster	Lokal innerhalb des ZFS-Pools	Aufgabe des SAN
Prüfsummen gegen stille Datenfehler	Ceph-intern	Nativ durch ZFS	Abhängig vom SAN
Typischer Einsatz	Größerer HA-Cluster	Einzelserver oder kleiner Cluster	Vorhandenes zentrales SAN

Ceph RBD ist ein verteilter Block-Storage mit Thin Provisioning, Redundanz, Snapshots und Clones. Lokales ZFS bietet ebenfalls Snapshots und Clones, ist jedoch kein Shared Storage. Reines iSCSI stellt LUNs bereit, definiert aber keine allgemeine Speicherplatzverwaltung. Deshalb wird häufig eine große gemeinsame iSCSI-LUN mit klassischem LVM kombiniert.

Weitere Details enthält die [offizielle Proxmox-VE-Storage-Dokumentation](#).

Direktes iSCSI, iSCSI mit LVM und ZFS over iSCSI

1. **Direktes iSCSI:** Das SAN stellt fertige LUNs bereit. Der Storage ist shared, bietet über das Proxmox-iSCSI-Backend aber keine nativen Snapshot- oder Clone-Funktionen.
2. **iSCSI + klassisches LVM:** Eine große gemeinsame LUN enthält eine von Proxmox verwaltete Volume Group. Speicher wird thick zugeteilt; Proxmox übernimmt das clusterweite Locking.
3. **ZFS over iSCSI:** ZFS läuft auf einem entfernten Storage-System. Proxmox erzeugt dort per SSH ZVOLS und exportiert sie als iSCSI-LUNs. Ohne redundanten Storage-Kopf kann dieses System zum Single Point of Failure werden.

Welche Proxmox-Funktionen hängen vom Storage ab?

Proxmox-Funktion	Ceph	Lokales ZFS	iSCSI thick
KVM-VMs und LXC-Container	Ja	Ja	Ja
HA-Neustart auf einem anderen Node	Ja	Nur mit Replikat oder Storage-Wiederherstellung	Ja
Live-Migration	Ohne Disk-Kopie	Mit Disk-Kopie	Ohne Disk-Kopie
VM-Snapshot inklusive RAM-Zustand	Ja	Ja	Versions- und Konfigurationsabhängig
Templates und Linked Clones	Ja	Ja	Keine Linked Clones

Proxmox-Funktion	Ceph	Lokales ZFS	iSCSI thick
Backup mit Proxmox Backup Server	Ja	Ja	Ja
Online-Vergrößerung einer VM-Disk	Ja	Ja	Ja
Automatische Replikation	Ceph-intern	pvesr oder zfs send/receive	SAN-intern

Proxmox GUI versus ZFS-Kommandozeile

Die Proxmox-Oberfläche verwaltet VMs, Container, Cluster-Locks und Storage-Zuordnungen. Die ZFS-Kommandozeile administriert dagegen den darunterliegenden Pool. Beide Ebenen ergänzen sich, dürfen aber nicht miteinander verwechselt werden.

Aufgabe	Proxmox VE	ZFS-Kommandozeile
Pools anzeigen	Node → Disks → ZFS	<code>zpool list</code>
Poolzustand prüfen	ZFS-Übersicht	<code>zpool status -v</code>
Speicherbelegung anzeigen	Storage Summary	<code>zfs list -o space</code>
I/O beobachten	PVE-Metriken	<code>zpool iostat -v 2</code>
Dataset anlegen	Storage-Konfiguration	<code>zfs create tank/vmdata</code>
ZVOL anlegen	VM-Disk hinzufügen	<code>zfs create -V 100G tank/vmdata/test</code>
Kompression aktivieren	Meist CLI	<code>zfs set compression=lz4 tank/vmdata</code>
VM-Snapshot erstellen	<code>qm snapshot 100 snap1</code>	<code>zfs snapshot tank/vmdata/vm-100-disk-0@snap1</code>
Snapshot zurückrollen	PVE-Snapshot-Menü	<code>zfs rollback ...@snap1</code>
Clone erzeugen	<code>qm clone 100 200</code>	<code>zfs clone ...@snap1 ...</code>
VM-Disk vergrößern	<code>qm resize 100 scsi0 +20G</code>	<code>zfs set volsize=120G ...</code>
Replikation	PVE Replication Job	<code>zfs send ... ssh ... zfs receive ...</code>
Datenintegrität prüfen	Tasks und Status	<code>zpool scrub tank</code>
Defekte Platte ersetzen	Disks/ZFS beziehungsweise CLI	<code>zpool replace tank alt neu</code>

Aufgabe	Proxmox VE	ZFS-Kommandozeile
TRIM starten	Zeitplan oder CLI	<code>zpool trim tank</code>

Praxisregel für Proxmox-VMs auf ZFS

Proxmox-eigene VM-Datenträger sollten nicht unkontrolliert direkt mit `zfs snapshot`, `zfs clone`, `zfs rollback`, `zfs rename` oder `zfs destroy` verändert werden. Proxmox kennt diese Änderungen sonst nicht. Für VM-Operationen sind deshalb Weboberfläche, API, `qm`, `pct` und `pvesm` die erste Wahl. Die ZFS-CLI eignet sich besonders für Poolzustand, Scrubs, Plattentausch, Kompression, Kapazitätsanalyse und geplante externe Replikation.

Automatische ZFS-Snapshots für Proxmox einrichten

Für die relevanten ZFS-Datasets und ZVOLS verwenden wir eine gestaffelte Snapshot-Aufbewahrung. Sie bietet engmaschige Wiederherstellungspunkte für aktuelle Fehler und ein gröberes Raster für ältere Stände.

Intervall	Aufbewahrung	Maximale Anzahl	Zweck
Alle 15 Minuten	3 Stunden	12	Schnelle Hilfe bei aktuellen Fehlern
Stündlich	96 Stunden	96	Vier Tage inklusive langer Wochenenden
Täglich	14 Tage	14	Zwei Wochen Tageshistorie
Wöchentlich	6 Wochen	6	Mittelfristige Wiederherstellung
Monatlich	3 Monate	3	Spät erkannte Fehler

Das ergibt maximal ungefähr **131 automatische Snapshots pro Dataset oder ZVOL**. Durch zeitliche Überschneidungen und das Verhalten der verwendeten Snapshot-Engine kann die tatsächliche Zahl abweichen.

Wie viel Speicher benötigen ZFS-Snapshots?

Ein ZFS-Snapshot ist keine vollständige Kopie. Beim Erstellen wird lediglich ein neuer Referenzpunkt im Blockbaum angelegt. Zusätzlicher Platz wird benötigt, sobald Daten im Live-Dataset überschrieben oder gelöscht werden und der Snapshot die alten Blöcke weiterhin referenziert. Der Speicherbedarf hängt deshalb von der Änderungsrate ab, nicht allein von der Anzahl der Snapshots.

```
zfs list -o name,used,available,usedbysnapshots
zfs list -t snapshot -o name,creation,used
zpool list
```

Snapshots auf demselben Pool sind kein vollständiges Backup. Fällt der gesamte Pool aus oder wird er zerstört, sind auch seine Snapshots verloren. Für längere Aufbewahrung und getrennte Kopien verwenden wir ZFS-Replikation und Proxmox Backup Server.

ZFS-Snapshots, Rollbacks, Clones und Snapdirs erklärt

ZFS-Snapshot erstellen

```
zfs snapshot rpool/data@vor-update
zfs snapshot -r rpool/data@vor-update
```

Snapshots entstehen nahezu sofort. Ein nativer Snapshot außerhalb von Proxmox ist jedoch zunächst nur crash-konsistent. Für Datenbanken oder VMs mit mehreren Disks können QEMU Guest Agent, Freeze/Thaw oder eine anwendungsspezifische Vorbereitung erforderlich sein.

Einzelne Dateien aus einem Snapshot wiederherstellen

```
cp /daten/.zfs/snapshot/daily-2026-08-31/datei /daten/datei
```

Eine einzelne Datei zu kopieren ist meist sicherer als ein Rollback des gesamten Datasets, weil alle anderen aktuellen Änderungen erhalten bleiben.

Snapdir sichtbar machen

```
zfs set snapdir=visible pool/daten
```

- `hidden`: `.zfs/snapshot` ist vorhanden, erscheint aber nicht im normalen Verzeichnislisting.
- `visible`: Benutzer können den Snapshot-Pfad sehen und ältere Dateien lesen.
- `disabled`: Zugriff über das Snapshot-Verzeichnis ist deaktiviert.

`snapdir` gilt für ZFS-Dateisysteme. Proxmox-VM-Disks sind üblicherweise ZVOLs und besitzen kein durchsuchbares Dateiverzeichnis. Für ZVOL-Snapshot-Geräte existiert stattdessen

`snapdev`. Siehe [OpenZFS-Dokumentation zu Snapshots, Clones und Snapdir](#).

ZFS-Rollback

```
zfs rollback pool/daten@vor-update
```

Ein Rollback verwirft alle Änderungen nach dem gewählten Snapshot. Standardmäßig ist nur der neueste Snapshot zulässig. `zfs rollback -r` kann neuere Snapshots löschen; `-R` kann zusätzlich abhängige Clones betreffen. Vor einem Rollback sollte der aktuelle Zustand deshalb separat gesichert werden.

ZFS-Clone

```
zfs clone pool/daten@vor-update pool/test
```

Ein Clone ist eine sofort beschreibbare Kopie eines Snapshots. Anfangs teilen sich Snapshot und Clone die unveränderten Blöcke. Neue Schreibvorgänge benötigen zusätzlichen Platz. Typische Anwendungen sind Tests, Updates, Datenrettung und isolierte Kopien. Der Clone bleibt von seinem Ursprungssnapshot abhängig, bis die Beziehung gegebenenfalls mit `zfs promote` umgekehrt wird.

ZFS-Platte schnell ersetzen und RAIDZ erweitern

Defekte ZFS-Platte mit Replace austauschen

```
zpool replace tank alte-platte neue-platte
zpool status tank
```

ZFS rekonstruiert die benötigten Blöcke aus den verbliebenen Geräten und schreibt sie auf die neue Platte. Ist die alte Platte noch lesbar, sollte die neue möglichst zuerst angeschlossen und der Replace kontrolliert gestartet werden. So bleibt die vorhandene Redundanz länger erhalten. Bei Mirrors kann `zpool replace -s` eine sequenzielle Rekonstruktion starten; für RAIDZ ist diese Option nicht verfügbar. Details stehen in der [OpenZFS-Referenz zu zpool replace](#).

RAIDZ-Expansion: RAIDZ um eine Platte erweitern

```
zpool attach tank raidz2-0 neue-platte
```

Mit aktuellen OpenZFS-Versionen kann ein bestehendes RAIDZ-vdev schrittweise um einzelne Platten verbreitert werden. Die Feature-Flag `raidz_expansion` muss vorhanden und aktiviert sein. Die neue Platte muss mindestens so groß wie die kleinste vorhandene Platte sein.

- RAIDZ1, RAIDZ2 oder RAIDZ3 behalten ihr jeweiliges Paritätsniveau.
- Die Anzahl tolerierbarer Plattenausfälle steigt durch die Expansion nicht.
- ZFS liest die belegten Daten und verteilt sie über die verbreiterte Gruppe.
- `zpool status` zeigt den Fortschritt.
- Bei einem Plattenfehler pausiert die Expansion, bis das vdev wieder gesund ist.
- Alte Blöcke behalten ihr ursprüngliches Daten-Paritäts-Verhältnis; neue Blöcke nutzen die breitere Geometrie vollständig.

Weitere technische Hintergründe bietet die [OpenZFS-Dokumentation zur Änderung des Pool-Layouts](#).

ZFS Spares, L2ARC, SLOG und Special Devices richtig einsetzen

Gerätetyp	Aufgabe	Was passiert bei Verlust?
Hot Spare	Ersatzplatte bereithalten	Die normale Pool-Redundanz entscheidet
Cache/L2ARC	Häufig gelesene Daten zusätzlich cachen	Unkritisch, da nur Kopien gespeichert werden
Log/SLOG	Synchrone Schreibvorgänge beschleunigen	Risiko für die jüngsten noch nicht übernommenen Sync-Schreibvorgänge
Special vdev	Metadaten und optional kleine Blöcke dauerhaft speichern	Der Verlust kann den gesamten Pool zerstören

Hot Spare

```
zpool add tank spare /dev/disk/by-id/...
```

Ein Hot Spare verkürzt die Zeit im degradierten Zustand, erhöht aber nicht die Fehlertoleranz. Ein RAIDZ2 bleibt trotz Spare ein RAIDZ2. Die ausgefallene Platte muss dauerhaft ersetzt

werden.

L2ARC als ZFS-Lesecache

```
zpool add tank cache /dev/nvme0n1
```

Der primäre ZFS-Cache ist der ARC im RAM. L2ARC auf SSD oder NVMe ist nur die zweite Stufe. Er lohnt sich vor allem bei wiederkehrenden zufälligen Lesezugriffen und einem Arbeitsdatensatz, der nicht vollständig in den RAM passt. L2ARC beschleunigt keine Schreibvorgänge; mehr RAM ist häufig die bessere erste Maßnahme.

SLOG für synchrone Schreibvorgänge

```
zpool add tank log mirror /dev/nvme0n1 /dev/nvme1n1
```

Ein SLOG ist kein allgemeiner Schreibcache. Es beschleunigt nur synchrone Schreibvorgänge, etwa bei NFS, Datenbanken oder VM-Storage mit häufigem `fsync()`. Geeignete Geräte benötigen geringe Latenz und Power-Loss-Protection. Wichtige SLOGs sollten gespiegelt werden. `sync=disabled` ist kein pauschaler Tuningtrick, weil dabei zugesagte Schreibsicherheit verloren geht.

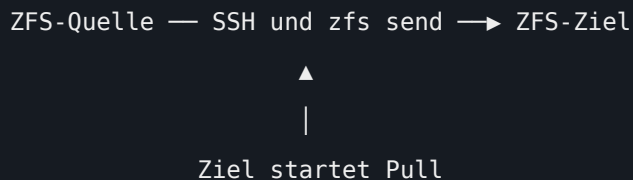
Special vdev für Metadaten und kleine Blöcke

```
zpool add tank special mirror /dev/nvme0n1 /dev/nvme1n1  
zfs set special_small_blocks=32K tank/daten
```

Ein Special vdev speichert Metadaten, indirekte Blöcke, gegebenenfalls Dedup-Tabellen und optional kleine Datenblöcke dauerhaft. Es ist kein Cache. Deshalb muss es mindestens so redundant wie der normale Pool aufgebaut werden. Geht das vollständige Special vdev verloren, kann der gesamte Pool verloren sein. Siehe [OpenZFS-Dokumentation zu Special vdevs](#).

ZFS Pull-Replikation mit bashclub zsync

`zsync` läuft auf dem Zielsystem und holt markierte ZFS-Dateisysteme und ZVOLS über SSH von der Quelle. Das Ziel initiiert also die Replikation:



Dieses Pull-Modell reduziert die Rechte der Quelle auf dem Sicherungsziel. Ein kompromittierter Proxmox-Server soll weder ältere Replikate löschen noch die Retention auf dem Ziel verändern können.

Datasets mit dem ZFS-Attribut bashclub:zsync markieren

```
zfs set bashclub:zsync=all rpool/data
```

- `all`: Dataset und untergeordnete Datasets beziehungsweise ZVOLS einbeziehen
- `subvols`: nur untergeordnete Datasets und ZVOLS einbeziehen
- `exclude`: den markierten Zweig ausschließen

Der Attributname kann geändert werden. Dadurch lassen sich verschiedene Replikationsziele mit unterschiedlichen Tags steuern.

Ablauf der zsync-Replikation

1. `zsync` startet auf dem Zielsystem.
2. Das Ziel verbindet sich per SSH mit der ZFS-Quelle.
3. Die ZFS-Attribute bestimmen die zu replizierenden Datasets und ZVOLS.
4. Optional wird vor der Replikation ein Snapshot erzeugt.
5. Quelle und Ziel suchen den neuesten gemeinsamen Snapshot.

6. Der erste Lauf sendet einen vollständigen Stream; Folgeläufe übertragen nur geänderte Blöcke.
7. Das Ziel erzeugt den vollständigen Dataset-Pfad automatisch.
8. `com.sun:auto-snapshot=false` verhindert ungewollte lokale Auto-Snapshots auf dem Replikationsziel.
9. Der Mountpoint wird übernommen und `canmount=noauto` verhindert ein unbeabsichtigtes Mounten.
10. `snapshot_filter` bestimmt, welche vorhandenen Snapshots mitgenommen werden.
11. Raw Replication kann verschlüsselte Blöcke ohne Entschlüsselung auf dem Ziel übertragen.

```
source=root@pve01
target=backup/repl/pve01
tag=bashclub:zsync
snapshot_filter=hourly|daily|weekly|monthly
min_keep=3
zfs_auto_snapshot_keep=14
zfs_auto_snapshot_label=zsync
```

Installation, Tags und Snapshot-Filter beschreibt die [deutsche bashclub-zsync-Dokumentation](#). Das frühere GitHub-Repository verweist inzwischen auf das GitLab-Projekt.

ZFS-Replikation per Snapshot-GUID mit checkzfs.py überwachen

Eine erfolgreiche Script-Ausgabe beweist noch nicht, dass ein aktuelles und brauchbares Replikat existiert. `checkzfs.py` vergleicht deshalb die Snapshots auf Quelle und Ziel anhand ihrer ZFS-GUID. Gleiche Snapshot-Namen reichen als Nachweis nicht aus.

Die Prüfung kann unter anderem folgende Fehler erkennen:

- kein Replikat gefunden,

- letzter gemeinsamer Snapshot zu alt,
- Snapshot auf dem Ziel nicht vorhanden,
- Rollback auf einen älteren Snapshot,
- zu viele Snapshots,
- unerwünschte Auto-Snapshots auf dem Replikationsziel,
- SSH- oder Abfragefehler.

Wunschalter der Replikation festlegen

```
threshold: 1500,2000
```

Im Beispiel wird nach 1.500 Minuten gewarnt und nach 2.000 Minuten ein kritischer Zustand gemeldet. Zusätzlich lassen sich maximale Snapshotzahlen, Dataset-Filter, Snapshot-Filter, SSH-Key und Servicename definieren.

```
source: pve01
remote: backup01
filter: rpool/data
replicafilter: backup/repl/pve01
snapshotfilter: zsync|daily|weekly|monthly
threshold: 1500,2000
maxsnapshots: 140,160
prefix: REPLICA
```

Die Ausgabe eignet sich unter anderem für Tabellen, JSON, CSV, E-Mail und lokale Checkmk-Checks. Der [Quellcode von checkzfs.py](#) zeigt die verfügbaren Parameter und Prüfungen.

Wichtig: Ein GUID-Vergleich bestätigt die gemeinsame Snapshot-Identität. Er ersetzt weder den regelmäßigen ZFS-Scrub noch einen getesteten Restore.

Trojanerresistentes Proxmox-Backup mit dem Miyagi-

Workflow

Der Miyagi-Workflow verbindet zwei unterschiedliche Sicherungsmethoden auf einem getrennten System:

- **Proxmox Backup Server:** deduplizierte und längerfristige VM- und LXC-Backups
- **Native ZFS-Replikation:** unmittelbar vorhandene Datasets und ZVOLS für einen schnellen Wiederanlauf

Das Ziel sind mindestens drei Datenkopien, zwei Standorte, zwei Sicherungsmethoden, automatische Kontrolle und eine möglichst kurze Netzwerk-Erreichbarkeit des Sicherungsservers.

Typischer Ablauf des Miyagi-Servers

Miyagi-Server startet

- |
- ├─ Updates und Zustandsprüfung
- ├─ ZFS-Replikation per Pull
- ├─ PBS-Backup koordinieren
- ├─ Verify, Wartung und Monitoring
- ├─ GUID und Alter der Replikate prüfen
- └─ Sicherungsserver herunterfahren

Das ZFS-Replikat kann einen schnellen Wiederanlauf ermöglichen, weil Datasets und ZVOLS bereits auf dem Ziel liegen. Das PBS-Backup bietet die längere, deduplizierte Historie, benötigt für die produktive Nutzung aber einen Restore.

Wann ist der Workflow gegen Ransomware und Trojaner widerstandsfähig?

„Trojanersicher“ ist keine einzelne Produkteigenschaft. Die Schutzwirkung entsteht durch technische Trennung und minimale Berechtigungen:

- Das Ziel startet die Pull-Replikation.

- Die Quelle besitzt keine administrativen Schreib- oder Löschrechte auf dem Ziel.
- SSH-Zugriffe werden auf notwendige Befehle und Datasets begrenzt.
- Der PBS-API-Key erhält nur die erforderlichen Backup- und Restore-Rechte.
- SSH-Passwortlogin wird deaktiviert und interaktive Konten werden mit 2FA geschützt.
- Snapshot-Löschung und Retention werden ausschließlich vom Ziel kontrolliert.
- Der Miyagi-Server hat häufig nur definierte Routen und kein Default Gateway.
- Außerhalb des Sicherungsfensters ist der Server ausgeschaltet oder wirksam isoliert.
- Scrubs, Verify-Jobs und Wiederherstellungstests werden regelmäßig durchgeführt.
- Mindestens eine Kopie befindet sich an einem anderen Standort.

Der dokumentierte Workflow kombiniert Backups, startfähige ZFS-Replikate, Monitoring, Wartung und automatisches Herunterfahren. Die vollständige Einrichtung beschreibt die Anleitung [**Backup und Replikation für Proxmox VE mit dem Miyagi-Workflow**](#).

Welche Proxmox-Storage-Lösung ist die richtige?

- **Ein bis zwei Nodes:** Lokales ZFS ist meist die einfachste und robusteste Lösung.
- **Drei oder mehr geeignete Nodes mit echtem Storage-HA:** Ceph RBD ist der typische Ansatz.
- **Vorhandenes professionelles SAN:** iSCSI mit klassischem LVM nutzt die bestehende zentrale Infrastruktur.
- **Thin Provisioning und schnelle Linked Clones:** Ceph oder ZFS.
- **Vorhersehbare feste Platzbelegung:** iSCSI mit LVM thick.
- **Schneller lokaler Restore plus getrennte Kopie:** ZFS-Snapshots mit Pull-Replikation und PBS kombinieren.

Häufige Fragen zu Proxmox, ZFS, Ceph und Replikation

Ist ein ZFS-Snapshot bereits ein Backup?

Nein. Ein Snapshot liegt normalerweise auf demselben ZFS-Pool wie die produktiven Daten. Er schützt gut vor versehentlichem Löschen oder unerwünschten Änderungen, aber nicht vor dem vollständigen Verlust oder der Zerstörung des Pools.

Was ist für Proxmox besser: Ceph oder ZFS?

Für einzelne Server und kleine Cluster ist lokales ZFS meist einfacher. Ceph eignet sich für mehrere Nodes, verteilten Shared Storage und hohe Verfügbarkeit ohne einzelnen Storage-Kopf. Ceph benötigt dafür mehr Hardware, Netzwerkleistung und Betriebsaufwand.

Kann Proxmox mit lokalem ZFS live migrieren?

Ja, eine laufende VM kann migriert werden. Da lokales ZFS kein Shared Storage ist, muss der VM-Datenträger dabei jedoch auf den Ziel-Node kopiert werden. Bei Ceph oder gemeinsamem iSCSI-Storage entfällt diese Storage-Kopie in der Regel.

Was ist der Unterschied zwischen ZFS-Replikation und Proxmox Backup Server?

ZFS-Replikation überträgt native Datasets, ZVOLS und Snapshots und ermöglicht dadurch einen schnellen Wiederanlauf. Proxmox Backup Server erstellt deduplizierte Backups mit längerer Historie und integriertem Restore für VMs und Container. Beide Verfahren ergänzen sich.

Warum ist Pull-Replikation sicherer als Push-Replikation?

Bei Pull-Replikation startet das Ziel die Verbindung und holt Daten von der Quelle. Die Quelle benötigt dadurch keine administrativen Rechte auf dem Sicherungsziel. Wird sie kompromittiert, kann sie ältere Ziel-Snapshots nicht ohne Weiteres löschen.

Wozu vergleicht checkzfs.py Snapshot-GUIDs?

Ein Snapshot-Name kann auf zwei Systemen identisch sein, ohne denselben Snapshot zu bezeichnen. Die GUID ermöglicht eine eindeutige Zuordnung und zeigt, ob Quelle und Ziel tatsächlich einen gemeinsamen replizierten Snapshot besitzen.

Beschleunigt ein ZFS-SLOG alle Schreibvorgänge?

Nein. Ein SLOG hilft nur bei synchronen Schreibvorgängen. Asynchrone Writes werden dadurch nicht grundsätzlich schneller. Ein geeignetes SLOG benötigt geringe Latenz und Power-Loss-Protection.

Kann der Verlust eines Special vdev den ZFS-Pool zerstören?

Ja. Ein Special vdev speichert bestimmte Blöcke dauerhaft und nicht nur als Cache. Es muss deshalb mindestens so redundant wie die normalen Daten-vdevs aufgebaut werden.

Fazit: Proxmox-Storage mit überprüfbarer Backup-Strategie

Ceph, lokales ZFS und iSCSI erfüllen unterschiedliche Aufgaben. Entscheidend ist nicht nur die Storage-Technik, sondern der vollständige Wiederherstellungsweg. Automatische ZFS-Snapshots liefern schnelle lokale Zeitpunkte, `zsync` erzeugt getrennte native Replikate, `checkzfs.py` kontrolliert deren Aktualität und der Miyagi-Workflow kombiniert ZFS-Replikation mit Proxmox Backup Server, Monitoring und zeitweiser Offline-Sicherung.

So entsteht aus einzelnen Storage-Funktionen eine mehrstufige Strategie gegen Hardwareausfälle, Fehlbedienung, spät erkannte Datenfehler und kompromittierte Produktivsysteme.

Version #5

Erstellt: 2026-08-31 13:35:48 UTC von Christian Zengel (sysops GmbH)

Zuletzt aktualisiert: 2026-09-07 16:30:46 UTC von Christian Zengel (sysops GmbH)