Proxmox VE

- Single-Node Installation (Deutsch)
  - Proxmox VE Einführung
  - Proxmox VE Installations- und Konfigurationshandbuch
  - Proxmox VE neu installieren, ohne VM/LXC-Daten zu verlieren (nur ZFS)
  - LXC-Festplattengröße in Proxmox erhöhen
  - Einrichtung eines sicheren SSH-Zugangs für Proxmox und LXC
  - Vorbereitung einer sicheren LXC-Vorlage (Proxmox)
  - Proxmox VE: Warnung „ignore invalid acl token“ nach dem Löschen von Access Tokens
  - Erstellen eines LXC-Container
  - SMB Share der NAS in einen Unpriviliged LXC Container bringen
  - Ultraleichte Linux-Optionen für Proxmox: DietPi und seine minimalistischen Alternativen
  - LXC Post-Install: User-Setup & SSH Hardening
- Proxmox v9 Cluster Installation in VirtualBox
  - Overview
  - Setting up the Proxmox Nodes
  - Building the Cluster
  - Proxmox VE 9 & Debian 13 Air-Gapped Update Guide

Single-Node Installation (Deutsch)

Deutsche Anleitung für eine Proxmox-9 Installation auf einem einzelnen Rechner mit 3 Festplatten.

Single-Node Installation (Deutsch)

Proxmox VE Einführung

Proxmox Virtual Environment (Proxmox VE) ist eine Open-Source-Plattform zur Servervirtualisierung, die hypervisorbasierte virtuelle Maschinen und containerbasierte Virtualisierung in einer gemeinsamen Verwaltungslösung vereint. Basierend auf **Debian Linux** kombiniert Proxmox die Technologien **KVM** und **LXC** und stellt Administratoren sowohl eine einheitliche Weboberfläche als auch Kommandozeilenwerkzeuge zur Verwaltung virtueller Infrastrukturen bereit. Aufgrund seiner Flexibilität sowie der integrierten Cluster- und Backup-Funktionen wird Proxmox VE häufig sowohl in Unternehmensumgebungen als auch in Homelabs eingesetzt.

Eckdaten

- **Erstveröffentlichung:** 2008
- **Entwickler:** Proxmox Server Solutions GmbH
- **Kerntechnologien:** KVM, LXC, Debian
- **Lizenz:** GNU AGPL v3
- **Aktuelle stabile Version:** Wird regelmäßig aktualisiert; größere Releases typischerweise alle 1–2 Jahre

---

Architektur und Funktionen

Proxmox VE wird direkt auf der Hardware installiert (**Bare-Metal-Betrieb**) und benötigt kein zusätzliches Host-Betriebssystem. Die Plattform unterstützt sowohl vollständige Virtualisierung mit **KVM** als auch leichtgewichtige Container mit **LXC**, wodurch ressourceneffiziente Deployments möglich sind.

Zu den zentralen Funktionen gehören:

- Webbasierte Verwaltung
- REST-API

- Integrierte Firewall
- Rollenbasierte Zugriffskontrolle (RBAC)
- Hochverfügbarkeits-Cluster (High Availability)

Storage und Backup

Proxmox VE unterstützt eine Vielzahl von Storage-Backends, darunter:

- Lokale Datenträger
- NFS
- iSCSI
- Ceph
- ZFS

Dies ermöglicht eine flexible Verwaltung der Daten von virtuellen Maschinen und Containern.

Das integrierte Backup-System, der **Proxmox Backup Server**, unterstützt inkrementelle und deduplizierte Sicherungen. Planung, Durchführung und Verifikation von Backups sind direkt in die Proxmox-Oberfläche integriert.

Clustering und Hochverfügbarkeit

Mehrere Proxmox-Nodes können zu einem Cluster zusammengeschlossen werden, der über das **Proxmox Cluster File System (pmxcfs)** verwaltet wird.

Dadurch werden folgende Funktionen ermöglicht:

- Gemeinsame Konfiguration über mehrere Nodes hinweg
- Live-Migration von Workloads
- Automatisches Failover bei Node-Ausfällen

Diese Funktionen unterstützen hohe Verfügbarkeit und gute Skalierbarkeit auch in produktiven Umgebungen.

Nutzung und Community

Durch das Open-Source-Modell und optionalen Subskriptions-Support hat sich rund um Proxmox VE eine große internationale Community gebildet. Die Plattform wird häufig von IT-Fachleuten, Bildungseinrichtungen sowie kleinen und mittelständischen Unternehmen eingesetzt, die eine kosteneffiziente Virtualisierungslösung mit Funktionsumfang auf Enterprise-Niveau suchen.

Proxmox VE bietet dabei viele Funktionen, die mit proprietären Lösungen wie VMware vSphere oder Microsoft Hyper-V vergleichbar sind.

Single-Node Installation (Deutsch)

Proxmox VE Installations- und Konfigurationshandbuch

Diese Anleitung beschreibt die Schritte zur Installation und Konfiguration von Proxmox VE, einer leistungsstarken Open-Source-Virtualisierungsplattform. Folge diesen Anweisungen sorgfältig, damit alles reibungslos klappt.

Lade die Proxmox VE ISO herunter

Geh auf die offizielle Proxmox-Website und lade das neueste **Proxmox VE ISO**-Image für deine Serverarchitektur herunter.

https://www.proxmox.com/de/

Erstelle einen bootfähigen USB-Stick

Nutze **BalenaEtcher** (oder ein anderes Tool, das du magst, z. B. Rufus oder `dd`), um die ISO-Datei auf einen USB-Stick zu flashen.

Tipp: Check nach dem Download die ISO-Prüfsumme, damit du sicher bist, dass die Datei heil ist.

Starte die Installation

- Boote den Computer vom USB-Stick.
- **Wichtig:** Wähle während der Einrichtung sorgfältig die **richtige Netzwerkschnittstelle** aus.
- Eine falsche Auswahl ist ein super häufiger Fehler, durch den du nach der Installation keinen Netzwerkzugriff mehr hast.

Falsche Schnittstelle korrigieren (ohne Neuinstallation)

Falls du die falsche Schnittstelle ausgewählt hast:

```
# Geh auf die lokale Konsole (Monitor+Tastatur oder IPMI/iLO).
nano /etc/network/interfaces
```

Passe die Zeilen „auto“ und „iface“ an deine gewünschte Netzwerkkarte an (z. B. „enp0s3“) und führ danach aus:

```
systemctl restart networking
```

Damit sparst du dir die komplette Neuinstallation.

Konfiguration nach der Installation

Enterprise-Repositorys deaktivieren

- Geh zu **Knoten ? Updates ? Repositorys**.
- **Deaktiviere** das Repository `pve-enterprise` (das braucht ein kostenpflichtiges Abo).

Repository ohne Abo aktivieren

Klick auf **Hinzufügen** ? Wähle **Ohne Abonnement** ? Trag ein:

```
deb https://download.proxmox.com/debian/pve bookworm pve-no-subscription
```

Bestätige und schließe.

Aktualisiere das System

- Geh auf die Registerkarte **Updates** und klick auf **Aktualisieren**.
- Sobald die Paketliste durch ist, klick auf **Upgrade**.
- Schau dir die Ausgabe an, bestätige und lass das System alle Updates einspielen.

Starte das System neu, wenn du dazu aufgefordert wirst, damit der neue Kernel und die neuen Dienste geladen werden.

Container-Vorlagen aktualisieren

Öffne die **Shell** (über die Weboberfläche oder per SSH) und führ folgenden Befehl aus:

```
pveam update
```

Damit holst du dir die neuesten LXC-Container-Vorlagen aus dem Proxmox-Repository, sodass du aktuelle Basis-Images (Ubuntu, Debian, Alpine usw.) nutzen kannst.

Du betreibst jetzt einen komplett aktualisierten, abonnementfreien Proxmox VE-Knoten, der bereit ist für VMs, LXCs und Homelab-Experimente.

Single-Node Installation (Deutsch)

Proxmox VE neu installieren, ohne VM/LXC-Daten zu verlieren (nur ZFS)

TODO – IN ARBEIT

> **Warnung:** Diese Methode gilt **nur, wenn VMs und LXCs auf einem separaten ZFS-Pool** liegen, der *nicht* dein Proxmox-Boot-Pool (`rpool`) ist.
> Wählst du während der Installation die falschen Festplatten aus, löschst du deine VM-/Container-Daten für immer.

> **Geltungsbereich:**
> Entwickelt für **Proxmox-Systeme mit einem einzigen Knoten**, die ZFS nutzen.
> Bei einem Cluster mit mehreren Knoten brauchst du zusätzliche Schritte für die Cluster-Wiederherstellung – die werden **hier nicht behandelt**.

Basisszenario

- **Host-Typ:** Einzelner Proxmox VE-Knoten mit ZFS
  - **Boot-Pool:** `rpool` – ZFS-Spiegel (2× SSD)
  - **Datenpool:** `zfs-data` – RAIDZ2 (4× NVMe) – hier liegen alle VM-/LXC-Festplatten
  - **Ziel:** Proxmox VE neu installieren **ohne die VM-Festplatten-Images sichern zu müssen**

Schritt 1: Wichtige Konfigurationsdateien sichern

Die VM- und LXC-**Metadaten** (also quasi die „Shells“) liegen hier:

```
/etc/pve
```

Diese Dateien enthalten Namen, Hardware, Festplatten und Ressourcen.

VM-Konfigurationsdateien

```
/etc/pve/qemu-server/*.conf
```

LXC-Konfigurationsdateien

```
/etc/pve/lxc/*.conf
```

Empfohlene zusätzliche Sicherungen

```
/etc/network/interfaces
/etc/hosts
/etc/pve/storage.cfg
```

> Tipp: Nutze `scp`, `rsync` oder WinSCP, um die Dateien an einen sicheren Ort zu kopieren.

Schritt 2: Alle Ressourcen sauber herunterfahren

Damit die Daten konsistent bleiben:

```
# Alle VMs stoppen
qm stop $(qm list | awk ,NR>1 {print $1}‘)

# Alle LXCs stoppen
pct stop $(pct list | awk ,NR>1 {print $1}‘)
```

Danach den Host ausschalten:

```
shutdown -h now
```

Schritt 3: Proxmox VE (sehr vorsichtig) neu installieren

1. Von der Proxmox VE-ISO booten
2. Bei der Festplattenauswahl:
   - Wähle **nur** die SSDs aus, die zu `rpool` gehören
   - Wähle **auf keinen Fall** die NVMe-Platten aus `zfs-data` aus
3. Rest der Installation ganz normal durchlaufen

> **Wichtig:** Falsche Festplatten auswählen = deine VM-Daten sind für immer weg.

Schritt 4: Den vorhandenen ZFS-Datenpool importieren

Nach dem ersten Boot:

```
zpool import
```

Wenn `zfs-data` automatisch da ist ? super, weiter.
Falls nicht:

```
zpool import -f zfs-data
```

Check danach:

```
zpool status zfs-data
zfs list
```

Schau, ob die Datensätze mit deinen VM-Festplatten da sind.

Schritt 5: Konfigurationsdateien zurückspielen

Kopier die gesicherten Dateien wieder zurück:

```
scp qemu/*.conf root@<new-proxmox-ip>:/etc/pve/qemu-server/
scp lxc/*.conf root@<new-proxmox-ip>:/etc/pve/lxc/
```

Die sollten sofort in der Weboberfläche auftauchen.

Schritt 6: Den ZFS-Pool wieder als Speicher hinzufügen

Option A: Web-GUI (empfohlen)

Rechenzentrum ? Speicher ? Hinzufügen ? ZFS

Auswählen: `zfs-data`

Inhalt: `Festplatten-Image, Container`

Option B: storage.cfg wiederherstellen

```
scp storage.cfg root@<new-proxmox-ip>:/etc/pve/storage.cfg
systemctl restart pve-cluster
```

Schritt 7: Prüfen, ob `/etc/pve` funktioniert (sehr wichtig)

Falls `/etc/pve` leer aussieht:

```
systemctl status pve-cluster
systemctl status corosync
journalctl -xe
```

Das Cluster-Dateisystem muss laufen, sonst kannst du die Konfigurationsdateien nicht nutzen.

Abschließende Überprüfung: Test-VM/LXC starten

```
qm start 100
pct start 103
```

Starten beide sauber ? System ist wieder einsatzbereit.

Hinweis zur Wiederherstellung

Falls du die Konfigurationssicherungen verloren hast, kannst du die Metadaten eventuell trotzdem retten, indem du schaust:

```
/zfs-data
```

oder:

```
zfs list
```

Die Datensatz-Namen entsprechen meist den VM-IDs.

Zusammenfassung

Wenn du die Konfigurationsdateien sicherst und den alten ZFS-Datenpool einfach wieder importierst, kannst du Proxmox VE neu installieren **ohne riesige VM-Disk-Images zu kopieren**. Das spart dir massiv Zeit, Bandbreite und Ausfallzeit. Die ganzen Prüfschritte machen den Prozess deutlich sicherer und wiederholbar.

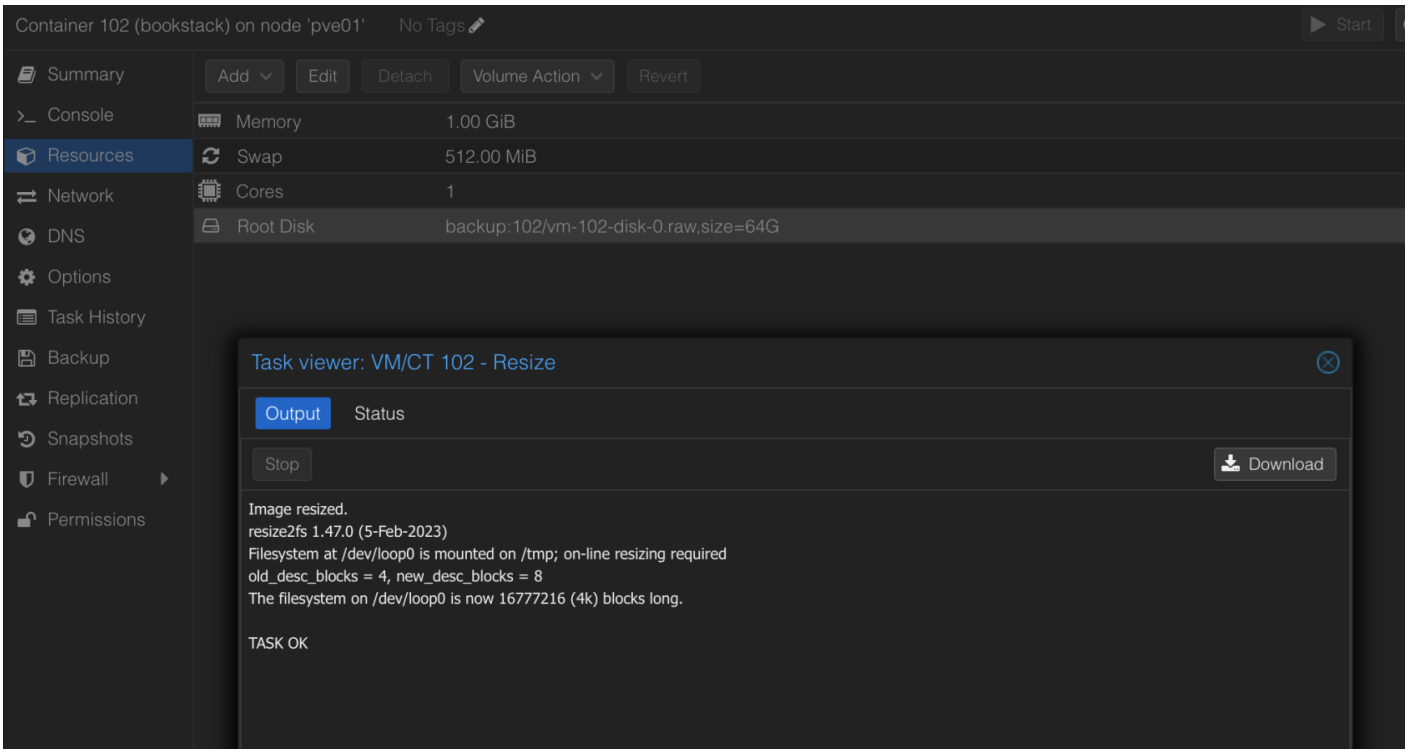
LXC-Festplattengröße in Proxmox erhöhen

Ich habe kürzlich bemerkt, dass mein BookStack LXC-Container auf Proxmox nur 8 GB Speicher zugewiesen hatte.
Zum Glück ist es ziemlich einfach, die Festplattengröße nachträglich zu vergrößern.

Die Festplattengröße über die Proxmox-Weboberfläche vergrößern
Die einfachste Methode geht direkt über die Proxmox VE Oberfläche:

1. Melde dich in der Proxmox-Oberfläche an.
2. Öffne deinen Container und geh auf **Ressourcen**.
3. Wähle die **Root Disk** (Wurzel-Festplatte) aus.
4. Im Dropdown-Menü **Volume Action** wählst du **Resize** (Größe ändern).
5. Gib ein, wie viel **zusätzlichen** Speicherplatz du hinzufügen möchtest
   (du gibst also nur den **Zuwachs** an, nicht die endgültige Gesamtgröße).

Wichtig:
Die Festplattengröße kannst du hier **nicht verkleinern**.



Die Festplattengröße über die Kommandozeile erhöhen

Du kannst die Festplatte auch direkt auf dem Proxmox-Host vergrößern:

1. SSH dich auf deinen Proxmox-Server.
2. Führe den Befehl `pct resize` mit der neuen Größe aus:

```
pct resize <vmid> <disk> <size>
```

Bei mir habe ich den Root-Disk meines BookStack-Containers (ID 105) auf **64 GB** erhöht:

```
pct resize 105 rootfs 32G
```

Wichtig: Du gibst hier die **Gesamtgröße** an, die der Disk danach haben soll (nicht den Zuwachs). Im Beispiel oben wird der Disk also auf insgesamt 64 GB gesetzt.

Nach dem Resize meldest du dich im Container an und prüfst die neue Größe mit:

```
df -h
```

Beide Methoden (Web-UI und Kommandozeile) funktionieren zuverlässig. Die Weboberfläche zeigt die aktualisierte Festplattengröße sofort an.

Einrichtung eines sicheren SSH-Zugangs für Proxmox und LXC

Das Ziel ist es, ein einheitliches, schlüsselbasiertes Authentifizierungsmodell sowohl für den Proxmox-Host als auch für seine Container aufzubauen:

- keine SSH-Root-Anmeldung
- keine Passwortauthentifizierung
- Zugriff nur über einen `admin`-Benutzer mit einem SSH-Schlüssel
- bequemer Zugriff über die SSH-Konfiguration

Dieser Ansatz reduziert die Angriffsfläche und macht alles deutlich einfacher zu verwalten.

Teil 1: Bereite deinen lokalen Rechner vor

Generiere ein SSH-Schlüsselpaar

Falls du noch keins hast:

```
ssh-keygen -t ed25519 -C "admin"
```

Speichere es sicher, zum Beispiel hier:

```
/home/deinname/.ssh/admin_key
```

Optional: Füge den Schlüssel zum Agenten hinzu

Wenn der Schlüssel eine Passphrase hat:

```
ssh-add /home/deinname/.ssh/admin_key
```

Teil 2: Sichere den Proxmox-Host

Erstelle einen Administratorbenutzer

Auf dem Proxmox-Host (über Web-Shell oder SSH):

```
adduser admin
usermod -aG sudo admin
```

Stelle den SSH-Schlüssel bereit

```
ssh-copy-id -i /home/deinname/.ssh/admin_key.pub admin@<Proxmox-Hostname>
```

Teste den Zugriff:

```
ssh -i /home/deinname/.ssh/admin_key admin@<Proxmox-Hostname>
```

Deaktiviere die unsicheren Authentifizierungsmethoden

Bearbeite `/etc/ssh/sshd_config`:

```
PermitRootLogin no
PasswordAuthentication no
ChallengeResponseAuthentication no
PubkeyAuthentication yes
```

Wende die Änderungen an:

```
systemctl restart sshd
```

Der Host akzeptiert jetzt nur noch schlüsselbasierte Anmeldungen für den `admin`-Benutzer.

Teil 3: Sichere deine LXC-Container

Viele Vorlagen erlauben SSH-Zugriff als Root – teilweise sogar mit Passwort. Wende dieselben Sicherheitsmaßnahmen in jedem Container an.

Erstelle einen Administratorbenutzer

Gehe in den Container:

```
pct enter <CTID>
```

Lege den Benutzer an und gib ihm sudo-Rechte:

```
adduser admin
usermod -aG sudo admin
```

Stelle deinen Schlüssel im Container bereit

Auf deinem lokalen Rechner:

```
ssh-copy-id -i /home/deinname/.ssh/admin_key.pub admin@<lxc-ip-oder-
hostname>
```

Teste:

```
ssh admin@<lxc-ip-oder-hostname>
```

Deaktiviere die unsicheren Methoden im Container

Bearbeite `/etc/ssh/sshd_config`:

```
PermitRootLogin no
PasswordAuthentication no
ChallengeResponseAuthentication no
PubkeyAuthentication yes
```

Starte neu:

```
sudo systemctl restart sshd
```

Der Container hat jetzt exakt dieselbe Sicherheitsstufe wie der Host.

Teil 4: Mach dir das Leben mit der lokalen SSH-Konfiguration leichter

Erstelle oder bearbeite die Datei:

```
~/.ssh/config
```

Beispiel-Inhalt:

```
Host proxmox
    HostName <Proxmox-Hostname>
    User admin
    IdentityFile /home/deinname/.ssh/admin_key

Host lxc-admin
    HostName <lxc-ip-oder-hostname>
    User admin
    IdentityFile /home/deinname/.ssh/admin_key
```

Schütze die Datei:

```
chmod 600 ~/.ssh/config
```

Ab jetzt reicht:

```
ssh proxmox
ssh lxc-admin
```

Ergebnis

- Proxmox-Host und alle Container nutzen dieselbe sichere Anmeldemethode.
- Root-Login ist überall deaktiviert.
- Passwort-Authentifizierung ist überall aus.
- Ein einziger Schlüssel + ein einziger Benutzer (`admin`) für den administrativen Zugriff.
- Super einfache Verbindung über sprechende Host-Namen in der SSH-Konfiguration.

Vorbereitung einer sicheren LXC-Vorlage (Proxmox)

Dieses Kapitel beschreibt, wie du eine **gehärtete LXC-Vorlage** erstellst, die du in Proxmox klonen kannst. Jeder Container, der aus dieser Vorlage erstellt wird, bietet:

- SSH-Zugang ausschließlich über einen `admin`-Benutzer
- Nutzung deines lokalen SSH-Public-Keys
- Verbot von direktem Root-Login
- Deaktivierung der Passwort-Authentifizierung

Dieser Ansatz sorgt für konsistente und sichere Deployments ohne manuelle Nachbearbeitungsschritte.

Schritt 1: Einen Basis-Container erstellen

Erstelle einen normalen LXC-Container aus deinem bevorzugten Image, zum Beispiel:

```
pveam update
pveam download local debian-12-standard_*.tar.zst
```

Danach deploy den Container:

```
pct create <CTID> local:vztmpl/debian-12-standard_*.tar.zst
```

Starte ihn:

```
pct start <CTID>
pct enter <CTID>
```

Schritt 2: Den Admin-Benutzer in der Vorlage anlegen

Innerhalb des Containers:

```
adduser admin
usermod -aG sudo admin
```

Dieser Benutzer wird der einzige SSH-Einstiegspunkt.

Schritt 3: Deinen SSH-Public-Key installieren

Auf deiner **lokalen Maschine** stelle sicher, dass du einen Key hast:

```
ssh-keygen -t ed25519 -C "admin@lxc"
```

Kopiere ihn in den Container:

```
ssh-copy-id admin@<container-ip>
```

Teste den Login:

```
ssh admin@<container-ip>
```

Du solltest dich ohne Passwort einloggen können.

Schritt 4: Die SSH-Konfiguration härten

Innerhalb des Containers:

```
sudo nano /etc/ssh/sshd_config
```

Wende folgendes an:

```
- PermitRootLogin yes
+ PermitRootLogin no

- PasswordAuthentication yes
+ PasswordAuthentication no

- ChallengeResponseAuthentication yes
+ ChallengeResponseAuthentication no
```

(Optional):

```
PubkeyAuthentication yes
```

Starte neu:

```
sudo systemctl restart sshd
```

Schritt 5: Aufräumen und für die Vorlage vorbereiten

Damit keine temporären Artefakte weitervererbt werden:

Innerhalb des Containers:

```
history -c
rm -rf /tmp/*
apt clean
```

Nicht löschen:

- `/home/admin/.ssh/authorized_keys`
- den `admin`-Benutzer
- die Änderungen an der SSH-Konfiguration

Diese Dinge werden für die Funktionalität der Vorlage benötigt.

Schritt 6: Herunterfahren und in eine Vorlage umwandeln

Verlasse den Container und stoppe ihn:

```
pct stop <CTID>
```

Wandle ihn in eine Proxmox-Vorlage um:

```
pct template <CTID>
```

Der Container ist jetzt als wiederverwendbare Vorlage gespeichert.

Ergebnis

Jeder zukünftige Container, den du aus dieser Vorlage klonst, bietet automatisch:

- sicheren SSH-Zugang
- keinen Root-Login
- erzwungene Schlüssel-basierte Authentifizierung
- konsistente sudo-Konfiguration

Keine weiteren SSH-Härtungsschritte mehr nötig.

Beispiel zum Klonen

```
pct clone <TEMPLATE-ID> 120 --hostname web01 --storage local-lvm
pct start 120
```

Danach sofort verbinden:

```
ssh admin@web01
```

Proxmox VE: Warnung „ignore invalid acl token“ nach dem Löschen von Access Tokens

Zusammenfassung

Nach dem Löschen eines Access Tokens oder eines Benutzers in der Proxmox-GUI kann beim Zugriff auf Container oder VMs (z. B. mittels `pct enter` oder `qm enter`) folgende Warnmeldung erscheinen:

```
user config - ignore invalid acl token 'user@realm!tokenname'
```

Diese Meldung ist harmlos, weist jedoch auf **verwaiste Token-Einträge** in der Proxmox-Konfiguration hin und sollte bereinigt werden.

Der Text hier benutzt `homepage@pam!homepage-192-168-10-110` als Beispielname.

Symptome

- Die Warnung erscheint bei:
  - `pct enter <CTID>`
  - `qm enter <VMID>`
  - teilweise auch bei API-Zugriffen
- Die betroffenen Container oder VMs funktionieren weiterhin normal
- `pvesh get /access/acl` zeigt **keine** fehlerhaften ACL-Einträge an

Ursache

Proxmox speichert Benutzer, Tokens und ACLs zentral in der Cluster-Konfigurationsdatei:

```
/etc/pve/user.cfg
```

Wichtiger Punkt (GUI-Verhalten)

> **Beim Löschen eines Users oder Access Tokens in der GUI werden zugehörige ACL-Einträge oder Token-Definitionen nicht automatisch vollständig entfernt.**

Das führt zu folgendem Zustand:

- Der User oder Token wurde in der GUI gelöscht
- In `user.cfg` existiert jedoch weiterhin ein `token:`-Eintrag
- Beim Laden der Benutzerkonfiguration erkennt Proxmox den Token nicht mehr korrekt
- Ergebnis: die Warnung „ignore invalid acl token“

Die Meldung ist irreführend, da sie nicht zwingend ein ACL-Problem ist, sondern ein **Konfigurationsartefakt**.

Diagnose

ACLs prüfen

```
pvesh get /access/acl
```

Wenn dort **kein** Eintrag mit dem genannten Token erscheint, liegt das Problem sehr wahrscheinlich in `user.cfg`.

Token-Einträge suchen

```
grep '^token:' /etc/pve/user.cfg
```

Oder gezielt:

```
grep 'homepage@pam' /etc/pve/user.cfg
```

Lösung (empfohlen)

1. Benutzerkonfiguration bearbeiten

```
nano /etc/pve/user.cfg
```

2. Verwaisten Token entfernen

Einen Eintrag dieser Art **vollständig löschen**:

```
token:homepage@pam!homepage-192-168-10-110:privsep=1:
```

?? **Nur den betroffenen `token:`-Eintrag entfernen**, keine anderen Benutzer-, Rollen- oder ACL-Zeilen.

---

3. Dienste neu laden

```
systemctl restart pve-cluster pvedaemon pveproxy
```

Die Änderung wird clusterweit sofort wirksam (pmxcfs).

---

4. Verifikation

```
pct enter <CTID>
```

Die Warnmeldung sollte nicht mehr erscheinen.

Best Practices / Prävention

- Tokens **vor** dem Löschen eines Users entfernen
- ACL-Zuweisungen vor dem Token-Löschen prüfen

- Nach umfangreichen Änderungen an Usern/Tokens:

```
grep '^token:' /etc/pve/user.cfg
```

- Bei Automatisierung bevorzugt API / `pvesh` nutzen statt ausschließlich die GUI

Hinweis

Das manuelle Bearbeiten von `/etc/pve/user.cfg` ist in diesem Fall **zulässig und sicher**, da es sich um eine vom Cluster-Dateisystem verwaltete Konfigurationsdatei handelt. Änderungen werden sofort synchronisiert.

Erstellen eines LXC-Container

Wir nutzen als Beispiel die ID **110**.

- **Allgemein:** Vergib ID, Hostname und Passwort. Optional kannst du deinen öffentlichen SSH-Key hinzufügen.
- **Template:** Ubuntu 24.04
- **Festplatten:** Mindestens 8 GB. Empfohlen: 32 GB oder 48 GB, wenn du viele Dateien oder Bilder speichern willst.
- **CPU:** 1 Kern reicht; 2 Kerne sind bei höherer Last empfehlenswert.
- **Arbeitsspeicher:** Mindestens 512 MB RAM + 512 MB Swap. Wenn möglich, erhöhe den RAM auf 1024 MB.
- **Netzwerk:**
  - IPv4: Statische IP, z. B. `192.168.100.110/24`
  - Gateway: Das Gateway deines Hosts, z. B. `192.168.100.1`
  - IPv6: Deaktiviert
- **DNS:** Nutze die Einstellungen des Hosts oder passe sie bei Bedarf an.
- **Bestätigen:** Schau dir alle Einstellungen nochmal genau an. Aktiviere **Nach Erstellung starten** und klick auf **Fertigstellen**.

Nach der Erstellung gehst du auf den Reiter **Konsole** und meldest dich als `root` an.
Falls nur ein Cursor zu sehen ist, klick einfach ins Konsolenfenster und drück Enter – dann erscheint die Eingabeaufforderung.
Melde dich mit `root` und deinem Passwort an.

System aktualisieren und upgraden:

```
apt update -qqy && apt upgrade -y
```

Nach Abschluss des Upgrades einmal neu starten.

Einen administrativen Benutzer anlegen und sudo-Rechte geben:

```
adduser admin
usermod -aG sudo admin
```

Ab jetzt kannst du entweder hier weitermachen oder in ein externes Terminal wechseln.

PostgreSQL Installation

Diesen Schritt kannst du als `admin` oder `root` machen. Beispiel als `admin`:

```
sudo apt update
sudo apt install postgresql postgresql-contrib -y
# Zum postgres-Benutzer wechseln
sudo -u postgres psql
# Innerhalb von psql:
CREATE DATABASE wikijs;
CREATE USER wikiuser WITH PASSWORD 'your_secure_password';
GRANT ALL PRIVILEGES ON DATABASE wikijs TO wikiuser;
\q
```

Node Installation

Folge den Anweisungen von der Node.js-Website:

```
sudo apt install curl -y
curl -fsSL https://deb.nodesource.com/setup_20.x | sudo bash -
sudo apt install nodejs -y
node -v
npm -v
# npm auf die im Terminal angezeigte Version bringen
sudo npm install -g npm@11.6.2
```

Wiki.js System-Benutzer anlegen

Erstelle einen dedizierten System-Benutzer für den Wiki.js-Dienst:

```
sudo useradd -r -s /usr/sbin/nologin wikijs
sudo mkdir -p /var/www/wiki
sudo chown wikijs:wikijs /var/www/wiki
```

Wiki.js Installation

Basierend auf der offiziellen Dokumentation: https://docs.requarks.io/install/linux

Installation

1. Neueste Wiki.js-Version herunterladen:

```
wget
https://github.com/Requarks/wiki/releases/latest/download/wiki-
js.tar.gz
```

2. Dateien entpacken:

```
mkdir wiki
tar xzf wiki-js.tar.gz -C ./wiki
cd ./wiki
```

3. Beispiel-Konfiguration umbenennen:

```
mv config.sample.yml config.yml
```

4. Konfiguration bearbeiten:

```
nano config.yml
```

5. *(Nur bei SQLite-Installationen)*:

```
npm rebuild sqlite3
```

6. Wiki.js starten:

```
node server
```

7. Warte auf die Setup-Meldung und öffne die angegebene URL in deinem Browser.

8. Den Setup-Wizard durchlaufen.

Als Systemdienst (systemd) einrichten

1. Service-Datei erstellen:

```
sudo nano /etc/systemd/system/wiki.service
```

2. Folgenden Inhalt einfügen (Pfade bei Bedarf anpassen):

```
[Unit]
Description=Wiki.js
After=network.target

[Service]
Type=simple
ExecStart=/usr/bin/node server
Restart=always
User=nobody
Environment=NODE_ENV=production
WorkingDirectory=/var/wiki

[Install]
WantedBy=multi-user.target
```

3. Speichern und schließen.

4. systemd neu laden:

```
sudo systemctl daemon-reload
```

5. Dienst starten:

```
sudo systemctl start wiki
```

6. Autostart aktivieren:

```
sudo systemctl enable wiki
```

Logs kannst du so anschauen:

```
journalctl -u wiki
```

SMB Share der NAS in einen Unpriviliged LXC Container bringen

Nachdem ich zahlreiche Projekte mit Proxmox umgesetzt habe, ist es für mich die vielseitigste Virtualisierungsplattform überhaupt. Dank der großen Anzahl an Community-Skripten und der klaren Ausrichtung auf Home-Lab-Setups lässt sich mit Proxmox und etwas Geduld nahezu alles realisieren – von virtualisierten Entwicklungsumgebungen und produktiven Alltags-VMs bis hin zu vollständigen Hackintosh-Installationen.

Ein NAS habe ich bislang bewusst nicht auf meinen PVE-Systemen betrieben und Storage-Server stets von meinen Proxmox-Workstations getrennt gehalten. Nach mehreren Gesprächen mit einigen Menschen im IRC und XMPP ob es möglich sei einen virtualisierten NAS-Ansatz zu betreiben, kamen wir zu dem Schluss, dass es zwar geht, aber dennoch keine optimale Sache sei und es zum testen und ausprobieren ok, aber für Produktivsysteme nicht geeignet ist. Allerdings fanden wir bei der Diskussion eine elegante Möglichkeit, Netzwerkfreigaben aus dieser VM mit unprivilegierten LXCs zu verbinden – ohne den sonst üblichen, aufwendigen Terminal-Einsatz. Und darum geht es hier.

Warum macht man das?

Nützlich, wenn ausschließlich unprivilegierte LXCs zum Einsatz kommen

Zwischen meiner ausgeprägten Neigung, möglichst viele Serverkomponenten anzuschaffen, betreibe ich sowohl ein primäres als auch ein sekundäres NAS, wobei Letzteres für Storage-Experimente genutzt wird. Wer zum Testen kein separates NAS sein eigenen nennt, kann unter Proxmox auch ein OpenMediaVault oder ein TrueNAS in einer VM einrichten. Damit funktioniert es genauso gut.

Überblick

Im Hinblick auf die Netzwerkfreigaben sieht mein Setup wie folgt aus: Auf meinem Test-NAS läuft eine SMB-Freigabe, die zunächst auf dem PVE-Host eingebunden wird. Anschließend wird per Terminal ein Mountpunkt auf den Node für den LXC eingerichtet, sodass die Freigabe dort wie ein lokales Laufwerk erscheint. Dieser Umweg ist notwendig, da unprivilegierte Container häufig

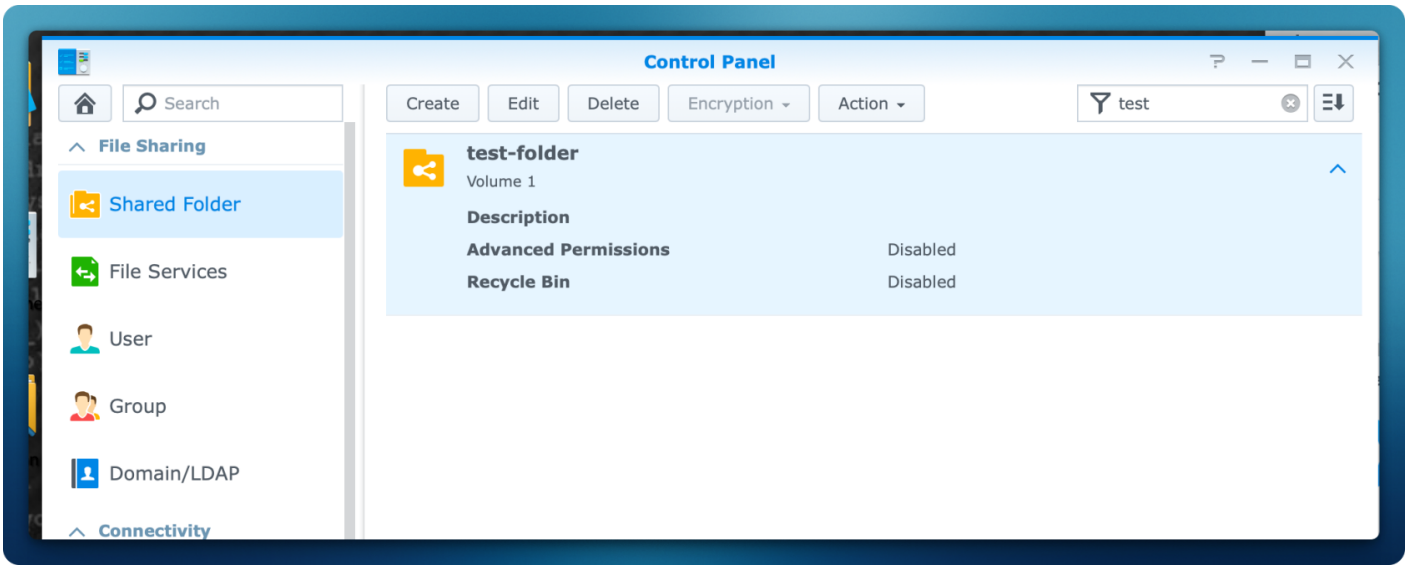
Berechtigungsprobleme beim direkten Einbinden von Netzwerkfreigaben verursachen. Aus Sicherheitsgründen sind unprivilegierte LXCs zwar vorzuziehen, sie erschweren jedoch genau dieses Szenario. Der beschriebene Ansatz umgeht diese Einschränkung und ermöglicht die Nutzung von SMB-Freigaben in sicheren, unprivilegierten Containern, ohne sich mit komplexen UID-/GID-Zuordnungen befassen zu müssen.

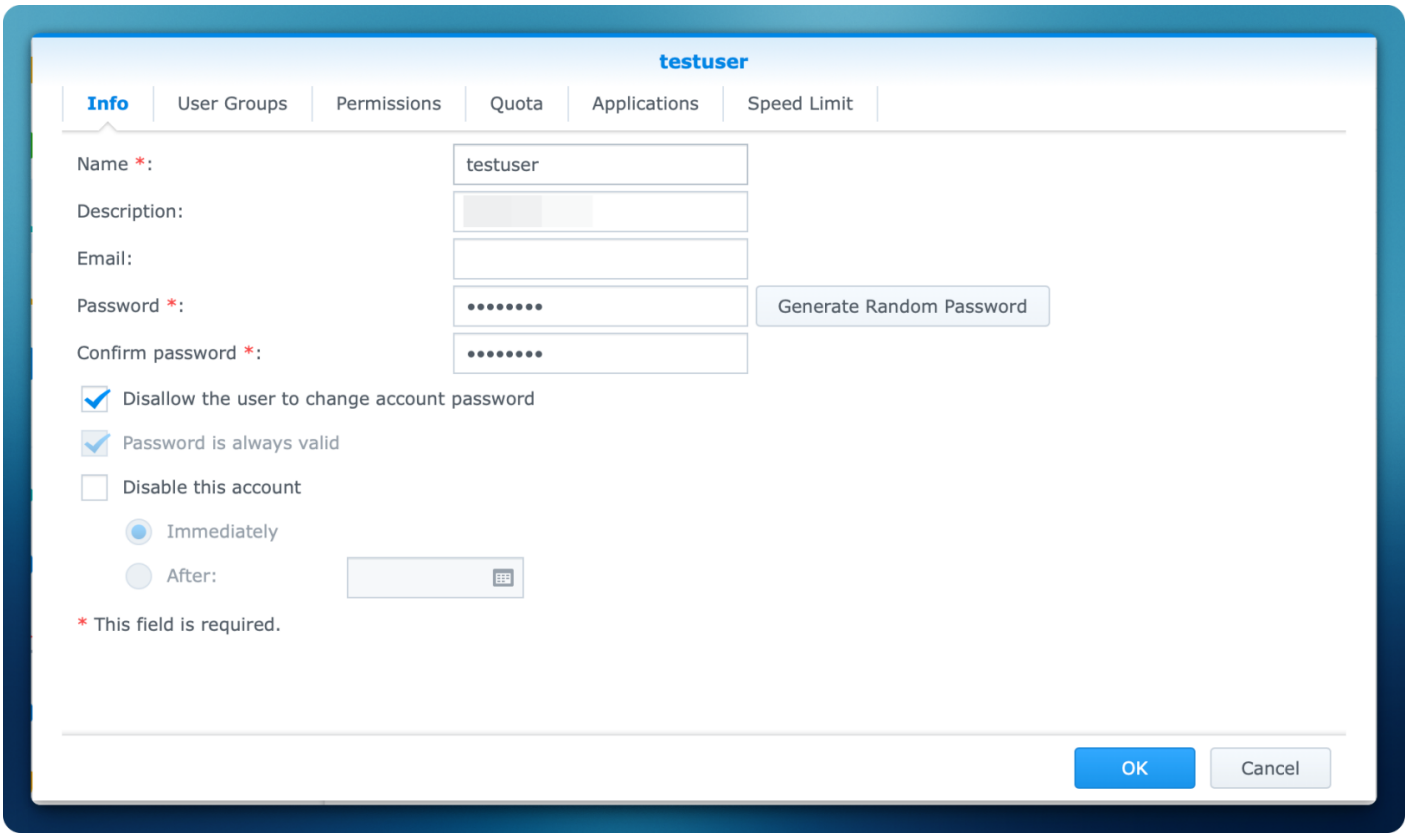
Annahmen

Auserdem gehen wir davon aus, dass auf dem NAS ein Nutzer namens "**Testuser**" mit Passwort existiert und die Freigabe unter `smb://192.168.10.20/test-folder` erreichbar ist.

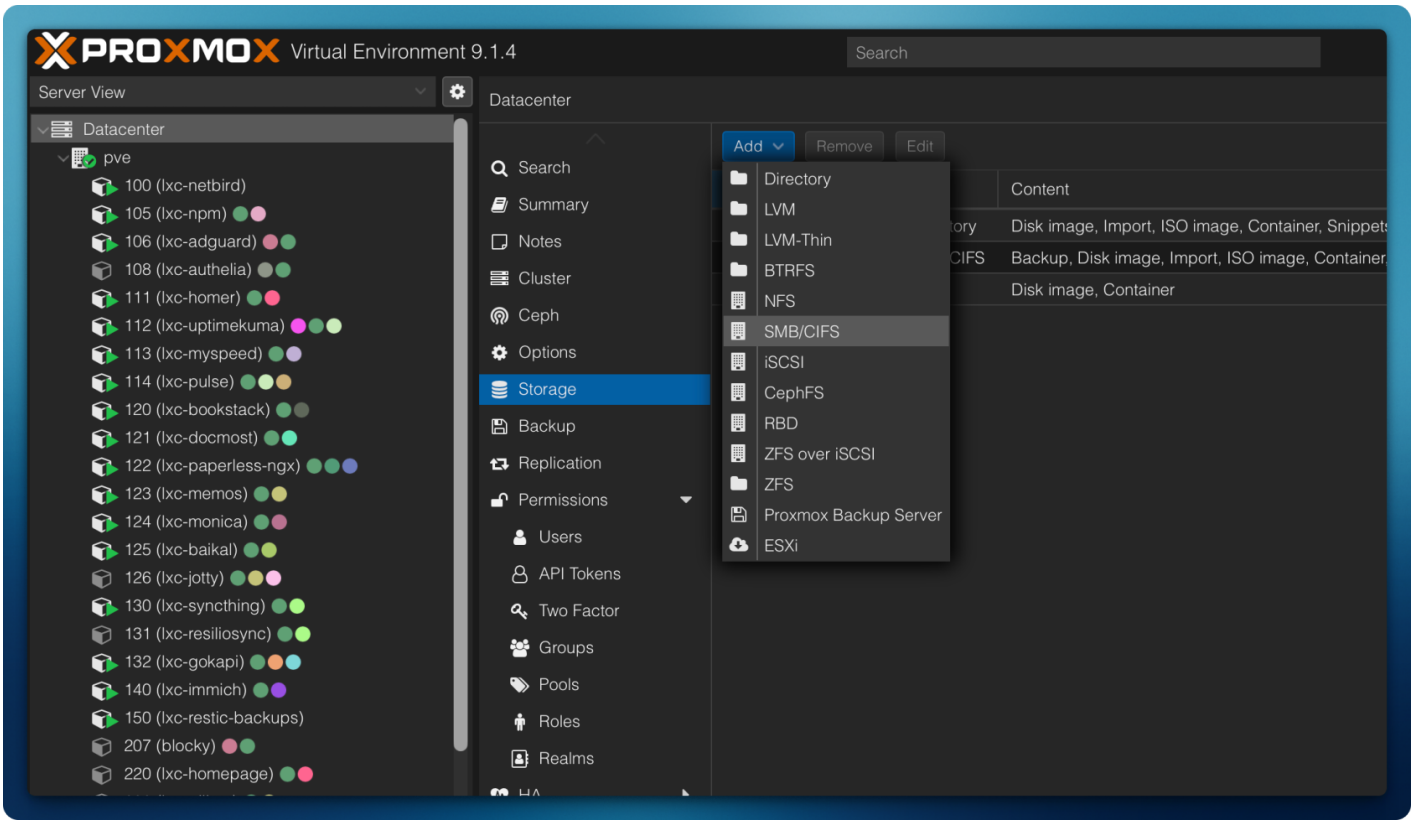
Außerdem verfügen wir über einen beliebigen LXC Container den wir zum Testen verwenden wollen. Ich habe hierzu einfach einen neuen LXC mit einer ID von 500 erstellt mit minimal CPU und RAM Angaben.

Synology Nutzer und Share



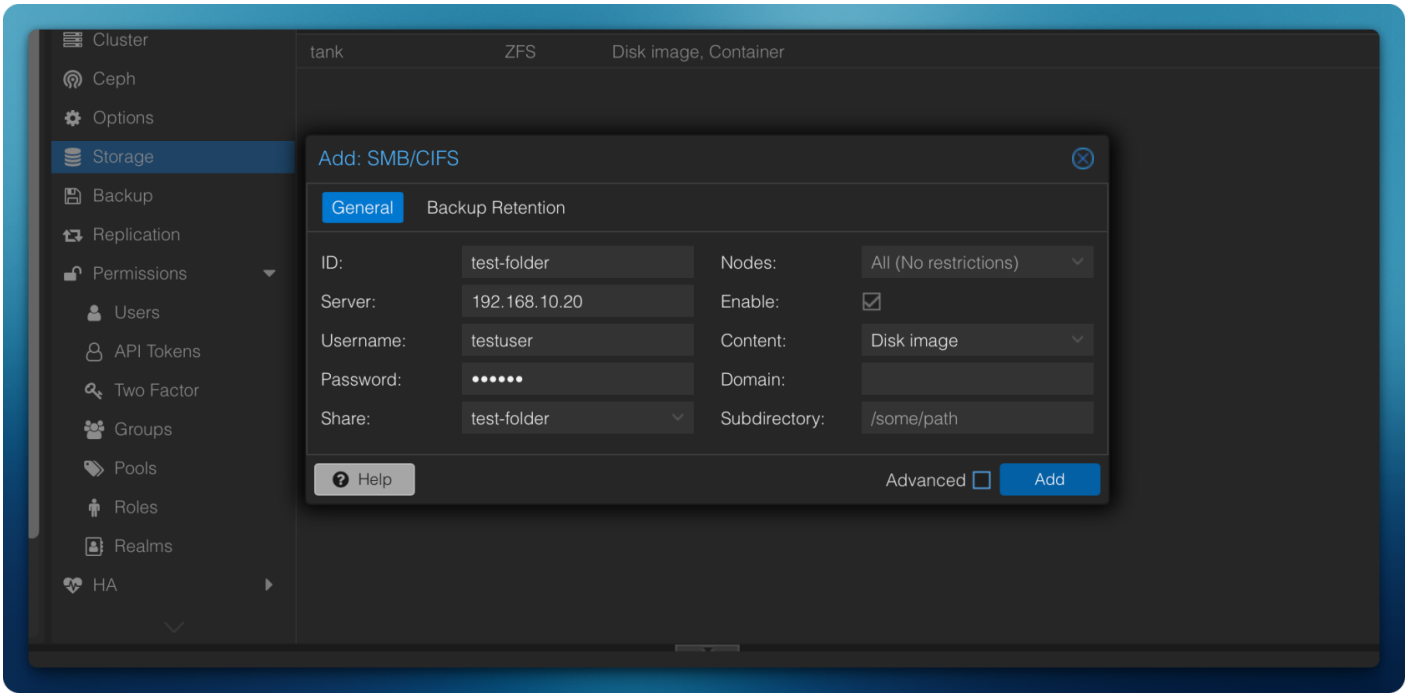


Einbinden des Share in Proxmox Node

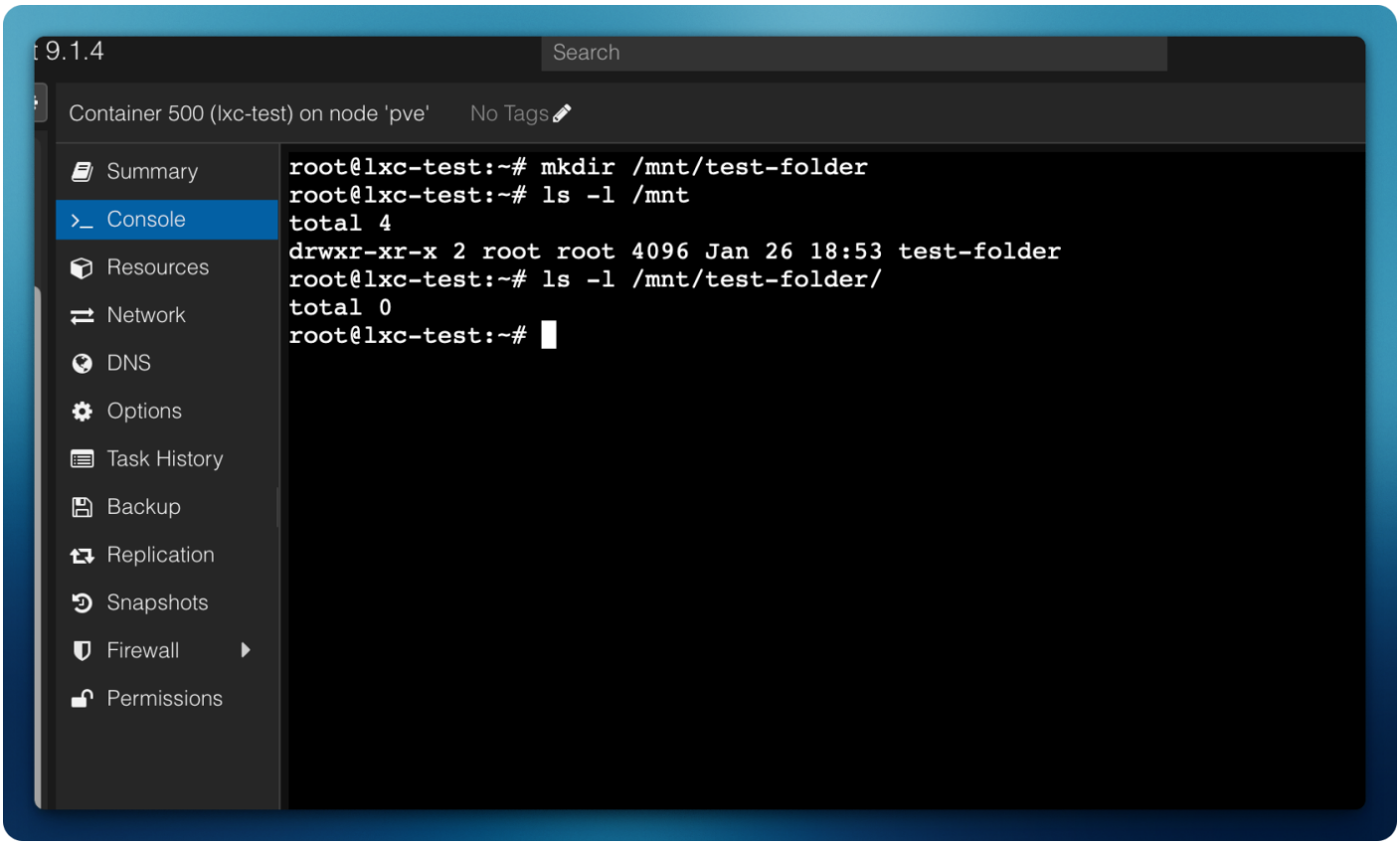


Nachdem die SMB-Freigabe auf der Synology eingerichtet war, wechselte ich zu meinem Proxmox-Host. Vor dem Anlegen des LXCs öffnete ich in der Proxmox-Weboberfläche den

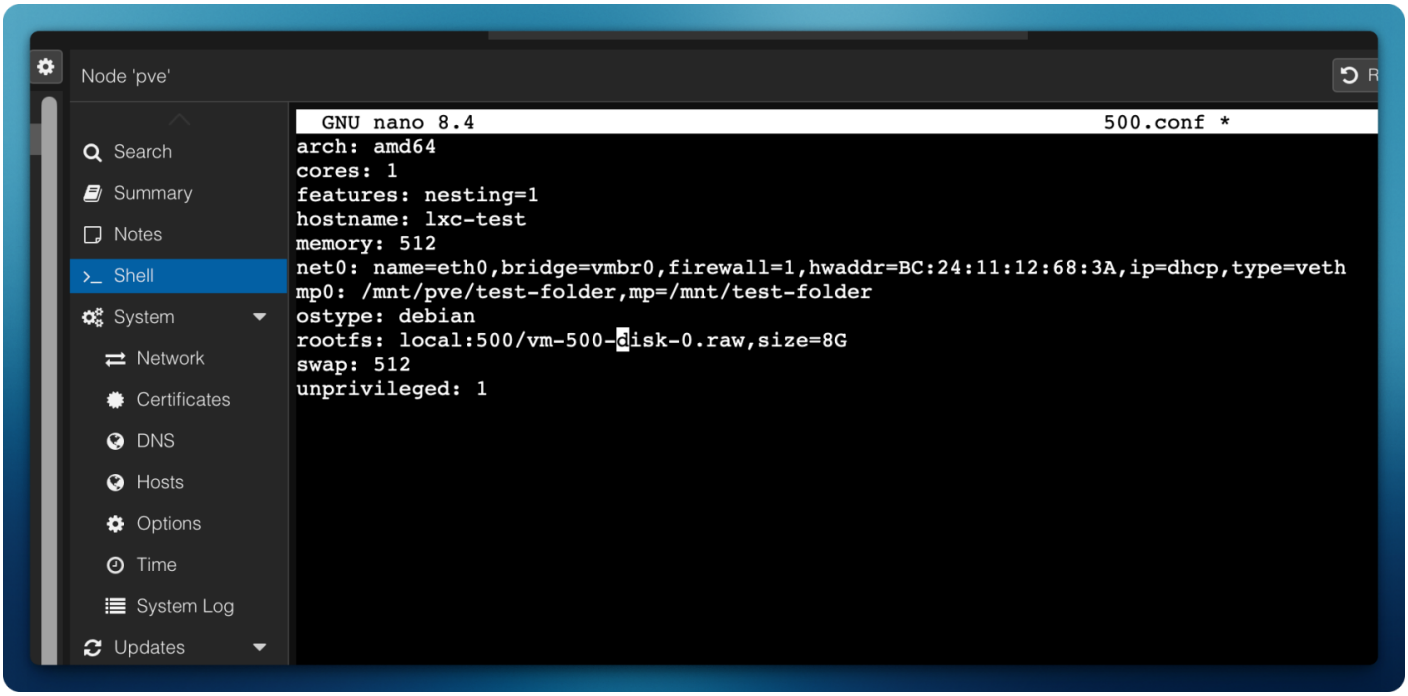
Datastore, ging auf den Reiter **Storage** und wählte über **Add ? CIFS/SMB** eine neue Netzwerkfreigabe aus. Als ID genügt ein beliebiger Name, als Server trug ich die IP-Adresse der Synology ein und hinterlegte Benutzername sowie Passwort des SMB-Kontos. Anschließend wurde die Freigabe automatisch erkannt und hinzugefügt.



Als Nächstes erstellte ich ein Verzeichnis, das als Mountpunkt im LXC dienen sollte. Dafür wechselte ich in die Shell meines **debianbasierten unprivilegierten Container** den ich zu Testzwecken mit der **ID 500** angelegt hatte. Anstatt zunächst Anwendungen zu konfigurieren, wechselte ich in der Proxmox-Oberfläche direkt in die **Console** des Containers und legte mit `mkdir /mnt/test-folder` das entsprechende Verzeichnis an.

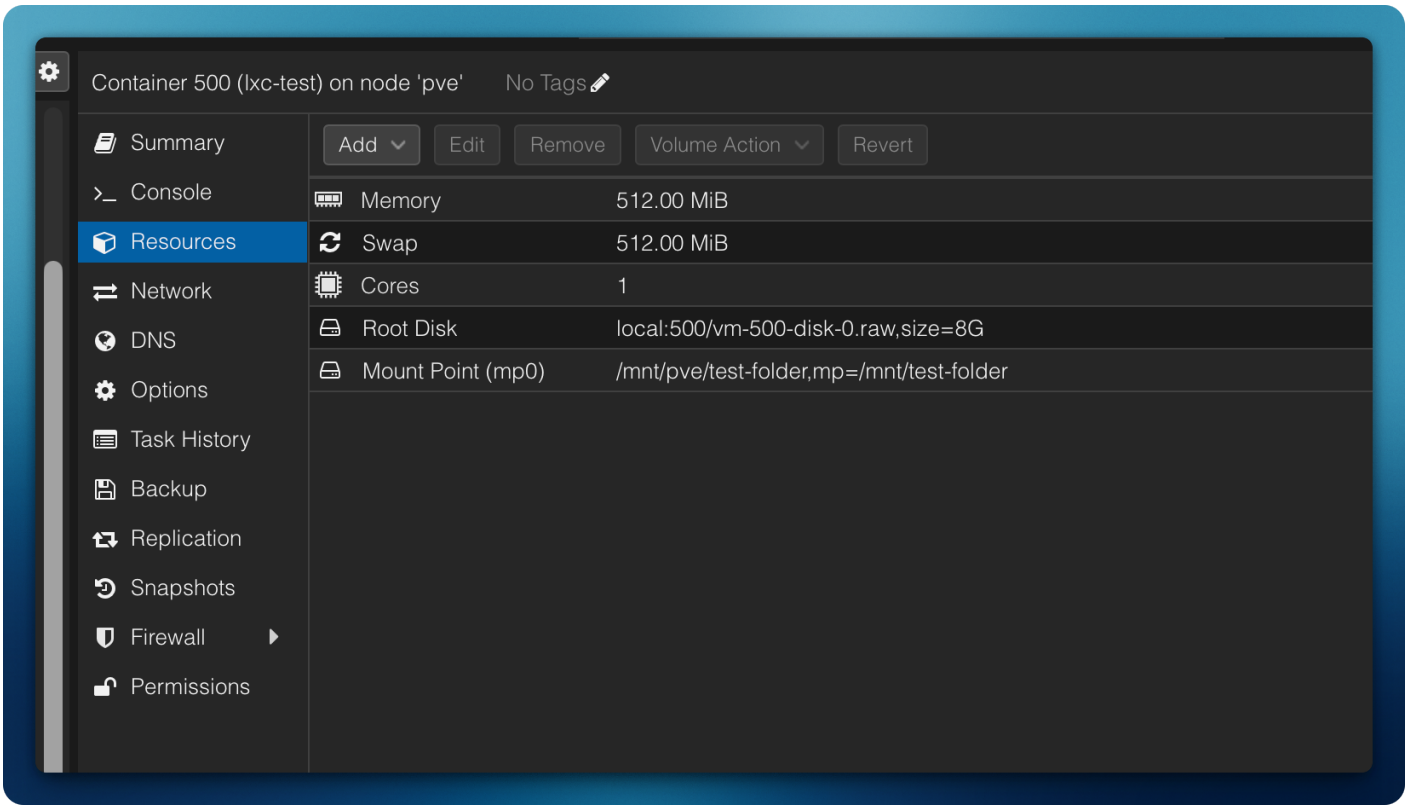


Zum Schluss wechselte ich auf dem Proxmox-Host in die Shell, navigierte nach `/etc/pve/lxc`, öffnete die Konfigurationsdatei des entsprechenden Containers (`500.conf`) und ergänzte dort die erforderliche Mount-Konfiguration, um die SMB-Freigabe im LXC verfügbar zu machen.



Hier verweist die Variable **test-folder** auf die ID, die ich der SMB-Freigabe in Proxmox zugewiesen hatte. Auch ohne einen Neustart sollte es funktionieren, zur Sicherheit aber, ist es sinnvoll einen

Neustart des Containers durchzuführen und dann zu schauen, ob dar eingebundene Verzeichnis im LXC sichtbar wird. In meinem Fall hat es geklappt.



Test der Verbindung und erweitern der Rechte

Zur Überprüfung lud ich eine Datei auf die Synology NAS hoch, die anschließend unmittelbar im Container erschien. Damit war bestätigt, dass die Einbindung der SMB-Freigabe korrekt funktionierte.

Das tut es auch soweit so gut. Innerhalb des LXC kann ich nun **lesend** auf die Dateien zugreifen. Schreiben jedoch nicht, da die Rechte nicht weitergereicht werden.

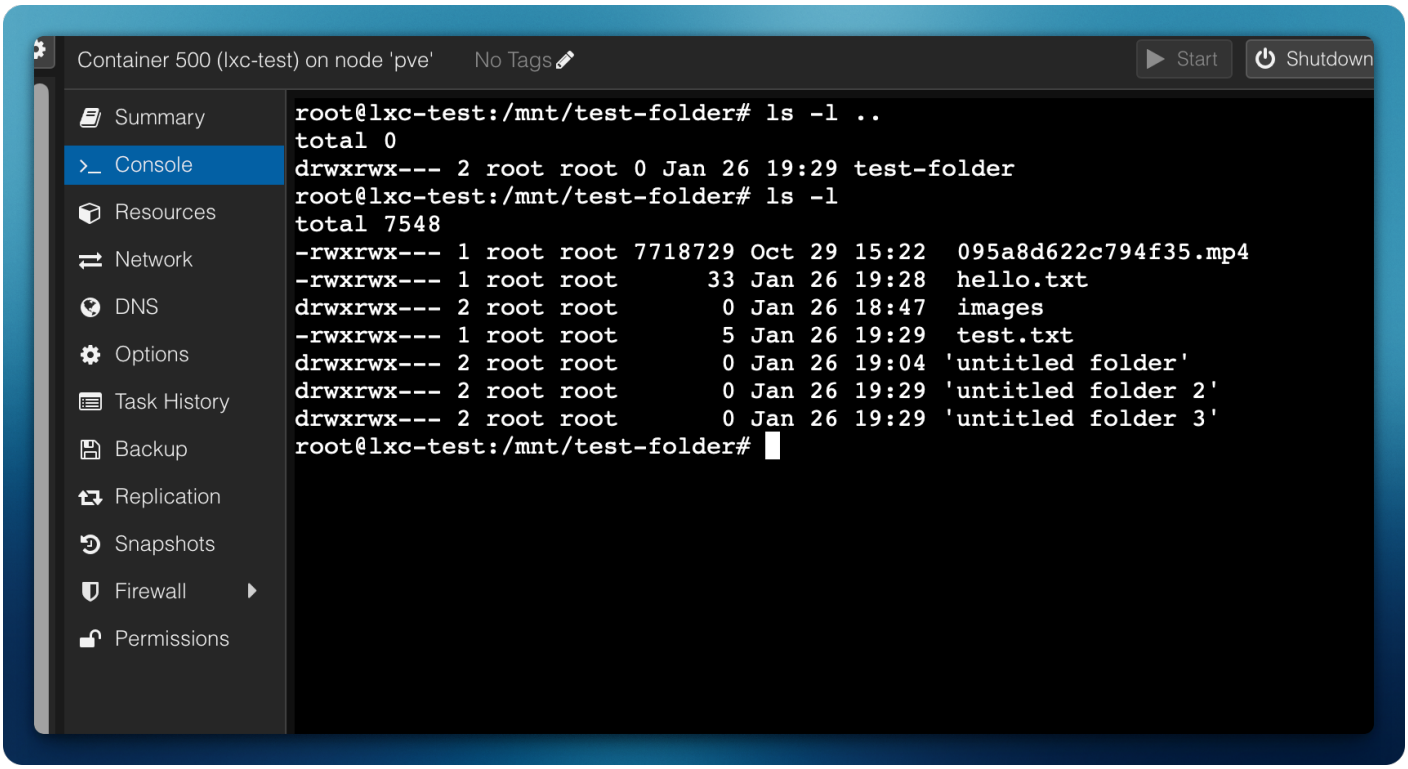
Um es dem lokalen `root` Nutzer des LXC zu ermöglichen auch Daten zu schreiben, ist noch ein kleiner Handgriff von Nöten.

Und zwar müssen wir auf dem Proxmox Node dazu einmal die `/etc/pve/storage.cfg` öffnen und folgende Zeile hinzüfugen

```
options uid=100000,gid=100000,file_mode=0770,dir_mode=0770
```

```
cifs: test-folder
        path /mnt/pve/test-folder
        server 192.168.10.20
        share test-folder
        content images
        options uid=100000,gid=100000,file_mode=0770,dir_mode=0770
        prune-backups keep-all=1
        username testuser
```

Wenn wir das gemacht haben, speichern wir die Datei und starten die LXC neu. Danach sollte der Ordner nun Root gehören, und die korrekten Berechtigungen haben.



Ultraleichte Linux-Optionen für Proxmox: DietPi und seine minimalistischen Alternativen

DietPi ist eine extrem optimierte Debian-basierte Distribution, die speziell für minimalen Ressourcenverbrauch entwickelt wurde. Dadurch eignet sie sich hervorragend als Basis für Proxmox LXC-Container und leichtgewichtige VMs. Mit ihrem winzigen Footprint, kaum laufenden Hintergrundprozessen und einem RAM-optimierten Logging-System liefert sie beeindruckende Performance – selbst auf eher bescheidener Hardware.

Das integrierte *DietPi-Software*-System macht die Sache noch einfacher: Beliebte Anwendungen wie Pi-hole, Docker, Home Assistant oder Nextcloud installierst und konfigurierst du mit nur wenigen Auswahlen – fertig vorkonfiguriert, ohne stundenlanges manuelles Herumgefummel.

DietPi steht aber nicht allein da. Es gibt mehrere andere minimalistische Linux-Distributionen, die ähnliche Ziele verfolgen:

- **Alpine Linux** – musl-basiert, extrem klein, weit verbreitet in Containern, ideal für Security-fokussierte Setups.
- **Tiny Core Linux** – absolut minimalistisch, startet mit nur wenigen Megabyte, perfekt für richtig knappe Umgebungen.
- **Debian Minimal / netinst** – saubere Basisinstallation ohne Schnickschnack, super zum Bauen eigener Server-Setups.
- **Ubuntu Server Minimal** – abgespeckte Ubuntu-Variante mit weniger Paketen und schneller Cloud-Integration.
- **Armbian Minimal** – optimierte Debian-/Ubuntu-Builds für ARM-Geräte, speziell für SBCs wie Raspberry-Pi-Alternativen getunt.

Alle diese Varianten bieten unterschiedliche Kompromisse zwischen Größe, Features und Bequemlichkeit. DietPi hebt sich vor allem durch seine Automatisierung und Optimierungs-Tools ab – du bekommst schnell eine einsatzbereite Umgebung, ohne dass der Ressourcenverbrauch

explodiert.

Für Proxmox-Nutzer, die maximale Effizienz ohne unnötige Komplexität wollen, bieten DietPi und seine minimalistischen Alternativen eine richtig starke und flexible Basis.

LXC Post-Install: User-Setup & SSH Hardening

Dieses Post-Install-Skript wird als `root` in einem frischen LXC-Container ausgeführt. Es automatisiert die grundlegende Einrichtung und Absicherung des Systems. Folgende Schritte werden durchgeführt:

- Erstellung eines neuen Standard-Benutzers.
- Zuweisung zur `sudo`-Gruppe und Einrichtung von passwortlosem `sudo`.
- Hinterlegung des öffentlichen SSH-Schlüssels (Public Key) für den neuen Benutzer.
- Entfernung aller hinterlegten SSH-Schlüssel für den `root`-Benutzer.
- Härtung der `/etc/ssh/sshd_config` (Deaktivierung von Root-Login und Passwort-Authentifizierung).

---

1. Vorbereitung (Host-System)

Lege das Post-Install-Skript (z.B. `post_install_ssh.sh`) und deinen Public Key (z.B. `ADMIN.pub`) in einem gemeinsamen Verzeichnis auf deinem Rechner ab.

Wechsle im Terminal in dieses Verzeichnis und starte einen temporären Python-Webserver, um die Dateien im Netzwerk bereitzustellen:

```
python -m http.server 8080
```

> **Hinweis:** Notiere dir die IP-Adresse dieses Rechners (im folgenden Beispiel `10.10.10.10`), da diese im nächsten Schritt benötigt wird.

---

2. Ausführung (LXC-Container)

Verbinde dich als `root` mit dem neuen LXC-Container.

Führe den folgenden Befehl aus. Er lädt das Skript herunter, macht es ausführbar und übergibt den Inhalt deines Public Keys direkt als Argument an das Skript.

(Passe die IP-Adresse `10.10.10.10` entsprechend an!)

```
curl -fsS
[http://10.10.10.10:8080/post_install_ssh.sh](http://10.10.10.10:8080/post_install_ssh.sh) -o /tmp/bootstrap.sh \
  && chmod +x /tmp/bootstrap.sh \
  && /tmp/bootstrap.sh "$(curl -fsS
[http://10.10.10.10:8080/ADMIN.pub](http://10.10.10.10:8080/ADMIN.pub))
"
```

---

3. Das Post-Install-Skript (`post_install_ssh.sh`)

> **Wichtig:** Passe den Wert `USER_NAME="ADMIN"` an deinen gewünschten Benutzernamen an, bevor du das Skript über den Webserver bereitstellst.

```
#!/usr/bin/env bash
set -euo pipefail

USER_NAME="ADMIN"
PUBKEY="${1:-}"
SSHD_CONFIG="/etc/ssh/sshd_config"

echo "==== SSH Bootstrap Starting ===="

########################################
# Ensure user exists
########################################
if id "${USER_NAME}" &>/dev/null; then
    echo "User ${USER_NAME} already exists."
else
    echo "Creating user ${USER_NAME}..."
    useradd -m -s /bin/bash "${USER_NAME}"
fi
```

```
#########################################
# Ensure sudo privileges
#########################################
if ! id -nG "${USER_NAME}" | grep -qw sudo; then
    echo "Adding ${USER_NAME} to sudo group..."
    usermod -aG sudo "${USER_NAME}"
fi

#########################################
# Configure passwordless sudo
#########################################
echo "Installing sudo if it is not installed"
apt install sudo -y -q

SUDO_FILE="/etc/sudoers.d/${USER_NAME}"

echo "Configuring passwordless sudo for ${USER_NAME}..."
echo "${USER_NAME} ALL=(ALL) NOPASSWD: ALL" > "${SUDO_FILE}"
chmod 440 "${SUDO_FILE}"

# Validate sudoers syntax before proceeding
visudo -cf "${SUDO_FILE}"

#########################################
# Configure SSH key
#########################################
SSH_DIR="/home/${USER_NAME}/.ssh"
AUTHORIZED_KEYS="${SSH_DIR}/authorized_keys"

mkdir -p "${SSH_DIR}"
chmod 700 "${SSH_DIR}"
touch "${AUTHORIZED_KEYS}"
chmod 600 "${AUTHORIZED_KEYS}"
chown -R "${USER_NAME}:${USER_NAME}" "${SSH_DIR}"

if [[ -n "${PUBKEY}" ]]; then
    if ! grep -qxF "${PUBKEY}" "${AUTHORIZED_KEYS}"; then
        echo "Installing public key..."
        echo "${PUBKEY}" >> "${AUTHORIZED_KEYS}"
    fi
else
    echo "WARNING: No public key supplied."
fi

#########################################
# Remove all root SSH authorized keys
#########################################
```

```
ROOT_SSH_DIR="/root/.ssh"
ROOT_AUTH_KEYS="${ROOT_SSH_DIR}/authorized_keys"
ROOT_AUTH_KEYS2="${ROOT_SSH_DIR}/authorized_keys2"

echo "Removing root SSH authorized keys..."

if [[ -d "${ROOT_SSH_DIR}" ]]; then
    rm -f "${ROOT_AUTH_KEYS}" "${ROOT_AUTH_KEYS2}"
    chmod 700 "${ROOT_SSH_DIR}" 2>/dev/null || true
    chown root:root "${ROOT_SSH_DIR}" 2>/dev/null || true
    echo "Root authorized_keys removed."
else
    echo "No root .ssh directory present."
fi

########################################
# Backup sshd_config
########################################
BACKUP="${SSHD_CONFIG}.$(date +%F-%H%M%S).bak"
cp "${SSHD_CONFIG}" "${BACKUP}"
echo "Backup created at ${BACKUP}"

########################################
# Idempotent config helper
########################################
set_config() {
    local key="$1"
    local value="$2"

    if grep -qiE "^\s*#?\s*${key}\b" "${SSHD_CONFIG}"; then
        sed -ri "s|^\s*#?\s*${key}\b.*|${key} ${value}|I"
"${SSHD_CONFIG}"
    else
        echo "${key} ${value}" >> "${SSHD_CONFIG}"
    fi
}

########################################
# Apply SSH hardening
########################################
echo "Applying SSH hardening..."

set_config "PermitRootLogin" "no"
set_config "PasswordAuthentication" "no"
set_config "KbdInteractiveAuthentication" "no"
set_config "ChallengeResponseAuthentication" "no"
set_config "PubkeyAuthentication" "yes"
```

```
set_config "AllowUsers" "${USER_NAME}"

########################################
# Validate before restart
########################################
echo "Validating sshd config..."
sshd -t

########################################
# Restart SSH
########################################
echo "Restarting SSH service..."
if command -v systemctl &>/dev/null; then
    systemctl restart sshd 2>/dev/null || systemctl restart ssh
else
    service ssh restart
fi

echo "==== SSH Hardening Complete ===="
echo "? Root SSH disabled"
echo "? Password auth disabled"
echo "? Only ${USER_NAME} allowed"
```

---

4. Beispiel-Ausgabe

Bei einer erfolgreichen Ausführung sollte die Ausgabe im Terminal deines LXC-Containers in etwa so aussehen:

```
==== SSH Bootstrap Starting ====
Creating user ADMIN...
Adding ADMIN to sudo group...
Installing sudo if it is not installed
Reading package lists...
Building dependency tree...
Reading state information...
sudo is already the newest version (1.9.16p2-3+deb13u1).
Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 0
Configuring passwordless sudo for ADMIN...
/etc/sudoers.d/ADMIN: parsed OK
Installing public key...
Removing root SSH authorized keys...
Root authorized_keys removed.
Backup created at /etc/ssh/sshd_config.2026-03-18-191349.bak
```

```
Applying SSH hardening...
Validating sshd config...
Restarting SSH service...
==== SSH Hardening Complete ====
? Root SSH disabled
? Password auth disabled
? Only ADMIN allowed
```

---

5. Login nach der Einrichtung

Sobald das Skript erfolgreich durchgelaufen ist, wurde der `root`-Login über SSH deaktiviert. Du kannst dich nun von deinem Client-Rechner aus (auf dem der passende Private Key liegt) sicher mit dem neu erstellten Benutzer anmelden:

```
ssh ADMIN@<IP-ADRESSE-DES-LXC>
```

Da das Skript passwortloses `sudo` eingerichtet hat, kannst du administrative Befehle nun einfach mit vorangestelltem `sudo` ausführen, ohne ein Passwort eingeben zu müssen (z.B. `sudo apt update`).

Proxmox v9 Cluster Installation in VirtualBox

Installation of Proxmox v9 Cluster within VirtualBox for testing and learning

Overview

This documenation describes a real-world virtualization lab by running Proxmox VE inside a VirtualBox VM, enabling a fully local environment for learning DevOps, infrastructure engineering, virtualization, and cloud concepts.

Proxmox & clustering — why it matters

Proxmox VE is a Debian-based, open-source hypervisor that runs KVM virtual machines and LXC containers, all managed through a web UI.

Benefits of clustering

- Single control plane: manage multiple Proxmox nodes from one dashboard.
- Live migration: move VMs between nodes with minimal downtime.
- High availability (HA): automatically restart VMs on healthy nodes if a node fails.
- Replication: scheduled syncing of VM data across nodes for fast recovery.
- Scalability: add nodes to increase capacity without reorganizing your setup.
- Better resource utilization: distribute CPU, memory, and storage load across the cluster.

When to use it

- Learning DevOps, infrastructure, or cloud concepts.
- Testing HA, migration, and replication workflows.
- Running multi-node labs that mirror production operations.

Key trade-offs

- Network and storage design become more important (latency, bandwidth, shared storage).
- Cluster management adds operational complexity and requires monitoring.
- Some features (e.g., HA, efficient replication) need reliable networking and proper fencing/qpinger setup.

Quick checklist to get started

1. Ensure time sync (NTP) and reliable networking between nodes.
2. Use separate networks for management, replication (migration), and cluster communication.
3. Configure fencing/qdevice or quorum helpers for safety in failure scenarios.
4. Test live migration and replication in your lab before trusting production workloads.

Key components

- Three computers or VMs with at least 6 GB RAM and 100 GB storage space per machine
    - Storage can be lower as we will be using dynamically allocated virtual disks
- VirtualBox — host hypervisor that runs the Proxmox VM
- Proxmox VE ISO — installed as the nested hypervisor inside VirtualBox
- Ubuntu Server ISO — guest OS installed in a VM managed by Proxmox
- Debian Server ISO — guest OS install in a VM managed by Proxmox
- VBoxManage — CLI for creating and configuring the outer VirtualBox VM
- Proxmox Web UI — dashboard used to manage the inner guest VMs

2-Nods vs 3-Node Cluster

A 2-node cluster is simpler to set up and fine for learning, but it lacks a proper quorum (the voting system that keeps a cluster running when a node fails). That means HA isn’t reliable without extra workarounds to prevent the cluster from stalling.

A 3-node cluster includes quorum by default: two of three nodes can keep the cluster running if one goes down. It’s more stable, supports real HA testing, and scales better—though it requires one more machine and a bit more setup.

We’ll use a 3-node cluster because it’s the best balance of stability and realistic, hands-on experience.

Lab architecture (my current setup)

- AMD Ryzen 7 7800X3D

- 32 GB RAM
- 1 TB NVMe storage

Overview of what will be built

- Server Template inside VirtualBox to clone
- Installation Proxmox VE inside a VirtualBox VM clones
- Configured networking so the Proxmox Web UI is reachable from the host browser.
- Uploaded the Ubuntu Server ISO into Proxmox and created a guest VM.
- Launched and installed Ubuntu inside Proxmox.
- Resolved nested-virtualization KVM errors by disabling KVM for the guest.

Network configuration

- Adapter 1: Bridge — outbound internet access and management network.
- Adapter 2: Host-Only — host ? Proxmox cluster communication (coresync) network.
- Adapter 3: Host-Only — host ? Proxmox replication (migration) network.

I have choosen the following IP-Addresses for management, coresync and replication:

- Node 1:
  - nic0 `192.168.0.201` (hostname `prox01.local`)
  - nic1 `172.20.1.201`
  - nic2 `172.20.2.201`
- Node 2:
  - nic0 `192.168.0.202` (hostname `prox02.local`)
  - nic1 `172.20.1.201`
  - nic2 `172.20.2.201`
- Node 3:
  - nic0 `192.168.0.203` (hostname `prox03.local`)

- nic1 `172.20.1.201`
- nic2 `172.20.2.201`

You should adapt the IP address of nic0 (the main interface of the Proxmox node) so that it is in the same subnet as your PC (for example, both in 192.168.0.0/24). This allows direct communication without additional configuration. If they are in different subnets, a router with the correct routing rules is required — otherwise, your PC will not be able to reach the Proxmox node.

Once you have downloaded all the ISO images, continue on the next page with the installation and setup of the Proxmox nodes and networking.

Setting up the Proxmox Nodes

On the first page of this chapter, we defined our requirements and made some key decisions. Now, let's set up the Proxmox nodes in VirtualBox and install Proxmox VE on them.

To ensure cluster stability and prevent traffic saturation, your test environment requires a minimum of three separate network interfaces:

- **Management Interface:** For host access and configuration.
- **Cluster Communication (Heartbeat):** Isolated traffic for node health and HA voting.
- **VM Migration & Storage Replication:** High-bandwidth link for data synchronization.

> **Note:** While dedicating a network segment for NAS storage is a best practice in production, it is excluded from the scope of this virtualized test exercise.

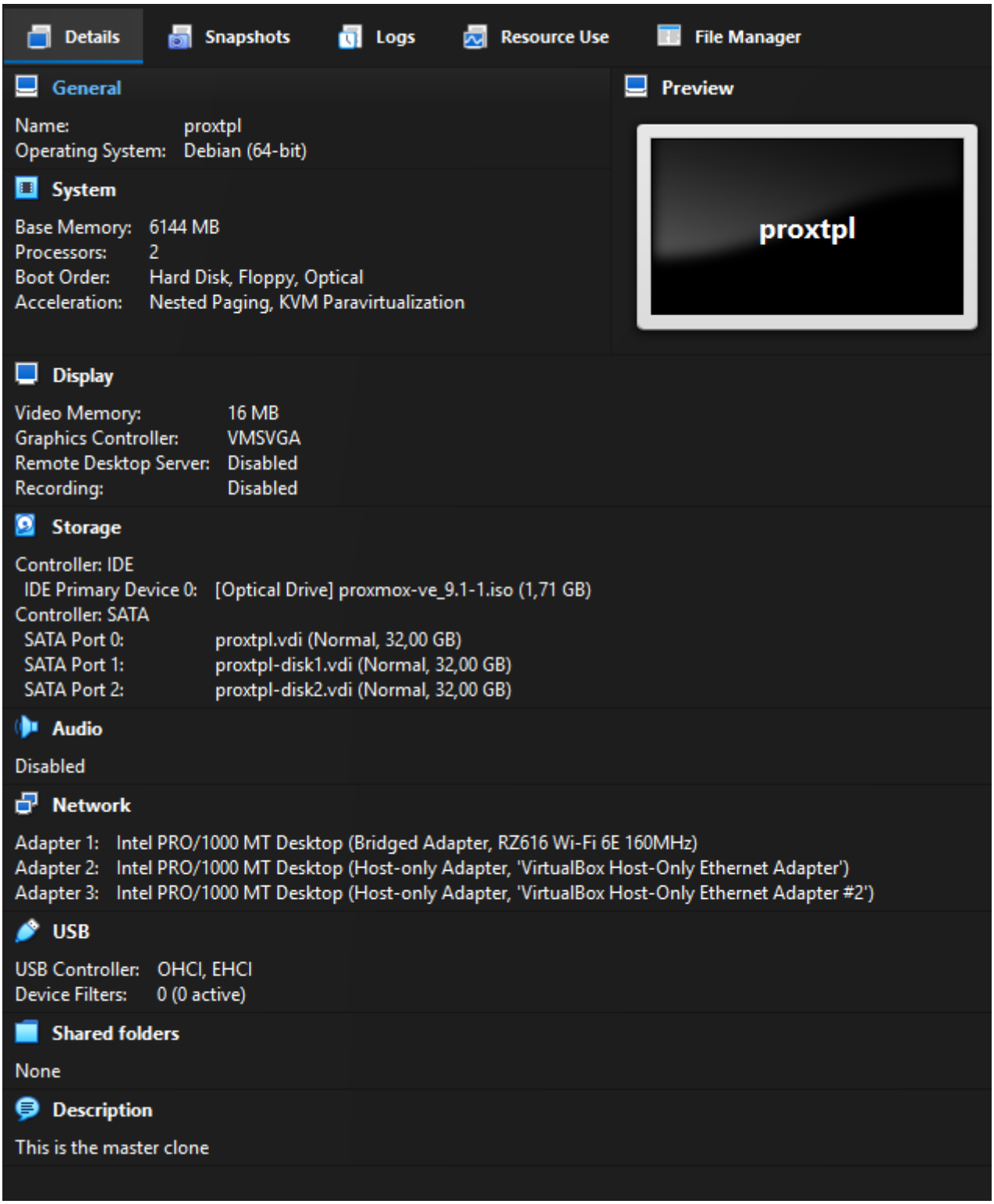
Creating the Master Virtual Machine (Hardware Mode)

To avoid missing critical settings or having to restart the process, it is recommended to build a single VirtualBox Virtual Machine (VBVM) to act as a template. You can name this master image `proxtpl` so that it remains at the bottom of your VM list. **Ensure that you never actually start the master image; it must remain powered off.**

- **Name:** `proxtpl`
- **Type:** Linux, Debian (64-bit)
- **RAM:** 2 GB minimum (4 GB or 6 GB is recommended if your host hardware allows).
- **Storage (SATA0):** Create a 32 GB boot drive for the Proxmox installation.

- **Storage (SATA1 & SATA2):** Create two 32 GB dynamically allocated disks; these will be used for the ZFS storage pool. You may use 64 GB if you have sufficient space.
- **Boot Order:** Adjust the order to set **Hard Disk** first and **Optical** second.
- **Optical Drive:** Mount the Proxmox ISO.
- **Network Adapters:** Attach 3 adapters:
  - **nic0:** BRIDGED (Management interface)
  - **nic1:** HOST-ONLY (Cluster interface / Core sync) — *Disable DHCP*
  - **nic2:** HOST-ONLY (Replication / Migration interface) — *Disable DHCP*

Once configured, your setup should look like this:



Once the master is configured, create three clones and name them `prox01`, `prox02`, and `prox03`.

Installing Proxmox VE (Software Mode)

With your three clones created, start them up to begin the installation.

- **Ignore KVM Virtualization errors:** You will likely see an error stating that KVM Virtualization is not detected. This is expected, as VirtualBox may not pass VT-x/AMD-V instructions through to the guest. This is not an issue for this lab, as **the test cluster will run Linux Containers (LXC)** instead of full VMs.
- **Verify hostnames:** Ensure you correctly name each node (e.g., `prox01.local`, `prox02.local`, `prox03.local`) during setup. Renaming a node after a cluster is established is a complex and difficult procedure.
- **Select the 32GB boot drive** for the installation target.
- **Configure sequential IP addresses:** Base these on the hostnames to simplify management (e.g., `prox01` – 192.168.10.201, `prox02` – 192.168.10.202, `prox03` – 192.168.10.203).
- **Network Settings:** Set the Netmask to `255.255.255.0` or use CIDR notation (e.g., `192.168.10.201/24`).
- **Gateway:** Set this to your network's gateway, most likely `192.168.10.1`.
- **DNS Server:** Set this to your local DNS configuration, also likely `192.168.10.1`.
- **Finalize:** After installation, shut down the hosts and **reboot them in headless mode**. This is where the boot order change from earlier comes in handy.

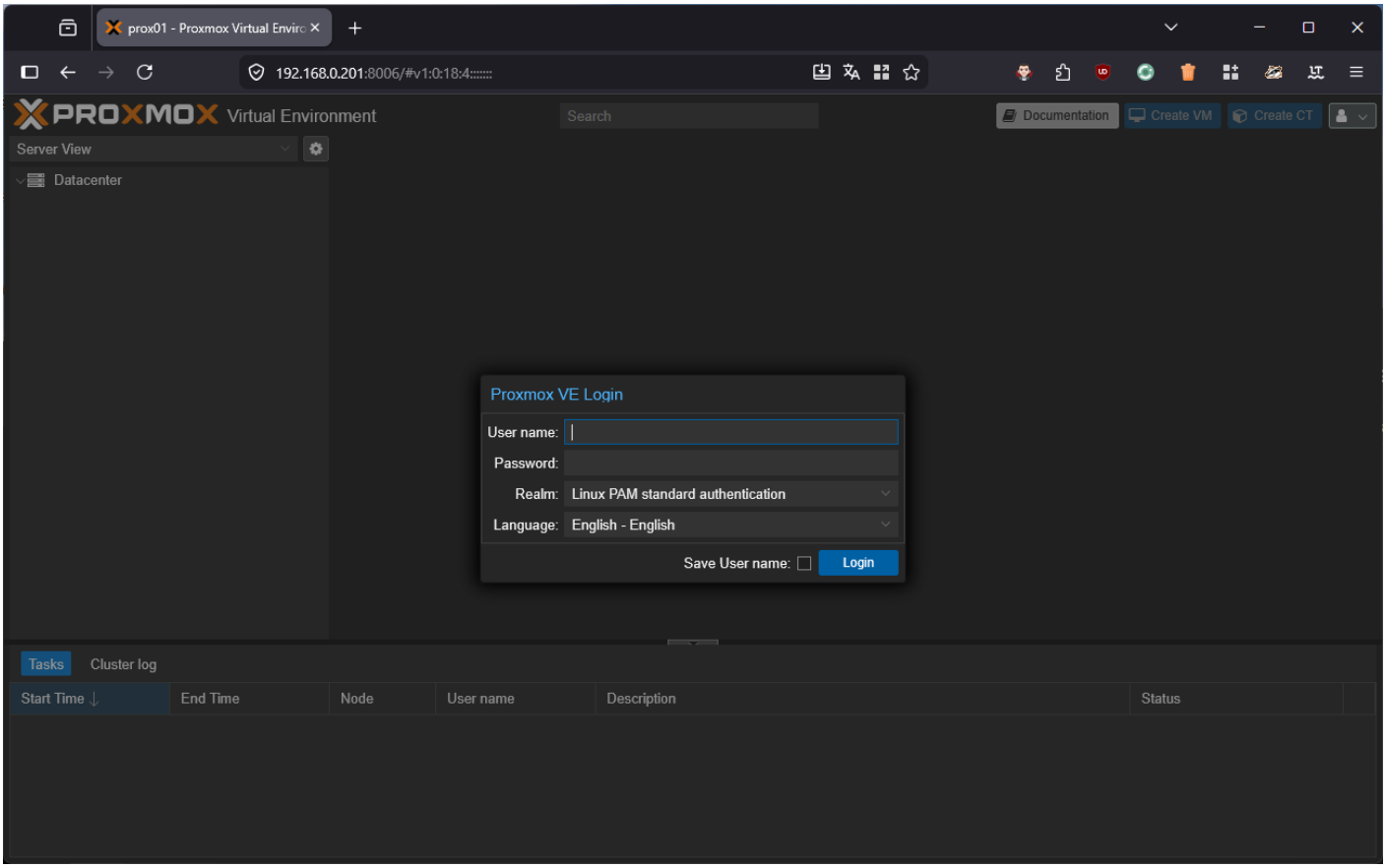
Unfold to see predefined IP Configuration

Remember the IP Configuration from the Overview-Page:

- Node 1:
  - nic0 `192.168.0.201` (hostname `prox01.local`)
  - nic1 `172.20.1.201`
  - nic2 `172.20.2.201`
- Node 2:
  - nic0 `192.168.0.202` (hostname `prox02.local`)
  - nic1 `172.20.1.201`
  - nic2 `172.20.2.201`
- Node 3:

- nic0 `192.168.0.203` (hostname `prox03.local`)
- nic1 `172.20.1.201`
- nic2 `172.20.2.201`

After the reboot, access the web interface on **port 8006** (HTTPS). You will likely be prompted to accept a self-signed certificate. Once you accept the security risk, you should see the Proxmox dashboard.



Post-Installation

Once your Proxmox nodes have successfully booted, there are several essential post-installation tasks to complete, such as managing repositories and removing subscription warnings. The most efficient way to handle this is by using the Proxmox VE Community Post-Install Script. This interactive script automates the removal of the "No Valid Subscription" nag-screen, configures the correct non-subscription repositories, and optimizes system settings.

```
bash -c "$(curl -fsSL https://raw.githubusercontent.com/community-scripts/ProxmoxVE/main/tools/pve/post-pve-install.sh)"
```

These are my recommended choices:

- disable pve-enterprise repository
- disable ceph enterprise repository
- enable/keep pve-no-subscription repository
- select NO to the pve-test repository
- select YES to disable the subscription nag
- select NO when asked to disable high availability
- select NO when asked to update (we will set up the ZFS Storage first)
- select NO when asked to reboot (we will set up the ZFS Storage first)

Install on an Air-Gapped environment

For an **air-gapped (offline) environment**, the standard script cannot reach external Proxmox servers. To resolve this, you can host a local version of the script within your network.

1. Host the Script Locally

Download the script to a machine that has network access to your Proxmox nodes. Navigate to the folder containing the script and start a temporary web server using Python:

```
# Start a local web server on port 8000
python3 -m http.server 8000
```

2. Execute on the Proxmox Node

Log into your Proxmox node's console and run the following command to pull and execute the script from your local server.

> **Note:** Replace `192.168.0.20` with the actual IP address of the machine running the Python server.

```
bash -c "$(curl -L http://192.168.0.10:8000/post-pve-install.sh)"
```

This method allows you to maintain a consistent configuration across all nodes in your cluster without requiring direct internet access for each individual machine.

Setting up ZFS Storage

Updating and final reboot

About the Network Interface Design

A successful Proxmox cluster requires careful network segregation to prevent traffic saturation. At a minimum, you will need **three separate network interfaces** for your VirtualBox test cluster:

1. **Management Interface:** This connects to your internal network and serves as the primary way to access and manage the Proxmox hosts.
2. **Cluster Communication Heartbeat:** This interface acts as the lifeline for the Proxmox hosts to communicate. It **must be placed on its own isolated network segment** (or physically isolated switch) because constant cluster "chatter" and High Availability (HA) voting can easily overwhelm a shared network link.
3. **VM Migration & Storage Replication:** A dedicated interface is needed for migrating machines and handling Storage Replication, which copies data across nodes every 15 minutes. Combining this heavy file synchronization traffic with standard cluster communication can quickly saturate a single network connection, potentially leading to false-positive node failures.

Building the Cluster

Setting IP ADdress

I did my best to simplify the network design:

There are 3 PVE hosts with corresponding management IP’s:

- prox1 – 192.168.0.201
- prox2 – 192.168.0.202
- prox3 – 192.168.0.203

Each PVE host has 3 network adapters:

- Adapter 1: A Bridged Adapter that connects to the [physical] internal network.
- Adapter 2: Host only Adapter #2 that will serve as the [virtual] isolated cluster network.
- Adapter 3: Host only Adapter #3 that will serve as the [virtual] dedicated migration network.

Each network adapter plugs into a different [virtual] network segment with a different ip range:

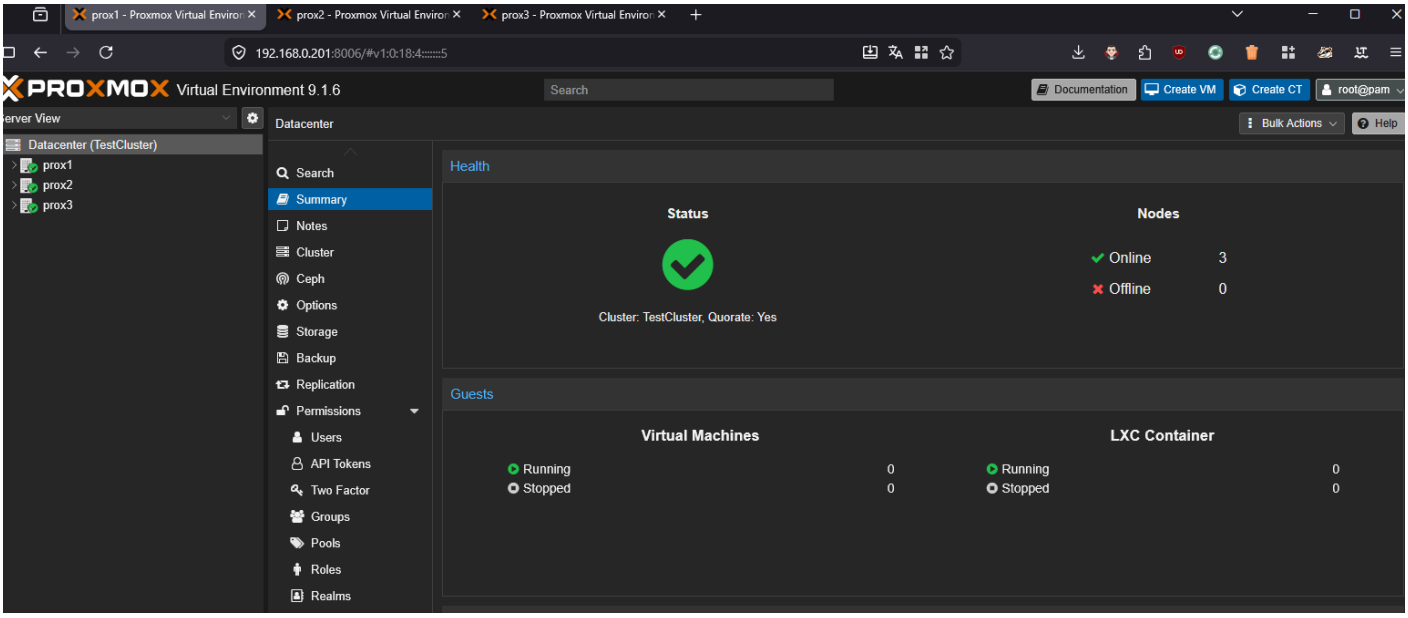
- Adapter 1 (vmbr0) (nic0) – 192.168.0.0/24
- Adapter 2 (nic1) – 192.168.101.0/24
- Adapter 3 (nic2) – 192.168.102.0/24

Each PVE hosts’ IP on each network roughly corresponds to its hostname:

- prox1 – 192.168.0.201, 192.168.101.1, 192.168.102.1
- prox2 – 192.168.0.202, 192.168.101.2, 192.168.102.2
- rox3 – 192.168.0.203, 192.168.101.3, 192.168.102.3

Prox 1 Example

```
auto lo
iface lo inet loopback

iface nic0 inet manual

auto vmbr0
iface vmbr0 inet static
        address 192.168.0.201/24
        gateway 192.168.0.1
        bridge-ports nic0
        bridge-stp off
        bridge-fd 0

# cluster network
auto nic1
iface nic1 inet manual
        address 192.168.101.1/24

# migration network
auto nic2
iface nic2 inet manual
        address 192.168.102.1/24

source /etc/network/interfaces.d/*
```



Proxmox VE 9 & Debian 13 Air-Gapped Update Guide

This guide describes how to set up an intermediate APT cache on an internet-connected machine (via VirtualBox) and transfer that cache to an air-gapped environment to update Proxmox VE 9 installations.

Phase 1: VirtualBox Setup (Internet-Facing Host)

On your internet-connected machine, you need two virtual entities: the **Cache Server** and a **Template Proxmox VM** to "pull" the initial data.

1.1 Debian 13 (Trixie) Cache VM

1. **Create VM:** 2 vCPUs, 2GB RAM, 50GB+ Disk (depending on how many packages you cache).
2. **Networking:** Use **Bridged Adapter** to ensure it has its own IP on your local network.
3. **OS Installation:** Install a minimal Debian 13 (Netinst). Ensure `SSH server` and `Standard system utilities` are selected.

I did set it up with the hostname `prox-cache` in my lan with the domain `lan.zn80.net` with the IP `192.168.0.240/24`.

When you get asked what to install in addition to the basis system, deselct the `Desktop Environment` and select only `WebServer`, `SSH Server` and `Standard System Utilities`.

After starting the system, enable the `sshd.service`

```
systemctl enable sshd.service
systemctl start sshd.service
```

Also add a static IP address by editing the `/etc/network/interfaces` file:

```
# The primary network interface
allow-hotplug enp0s3
iface enp0s3 inet static
        address 192.168.0.240/24
        gateway 192.168.0.1
```

This should be it for now. Continue with installing the Proxmox-Feeder.

1.2 Proxmox VE 9 "Feeder" VM

To populate the cache, you need a machine that requests the specific Proxmox 9 packages.

1. **Create VM:** 2 vCPUs, 4GB RAM, 20GB Disk.
2. **OS Installation:** Install Proxmox VE 9 (or Debian 13 + PVE 9 packages).
3. **Networking:** Ensure it can reach the Debian 13 Cache VM.

Id did set it up with the hostname `prox-feeder` in my lan with the domain `lan.zn80.net` with the IP `192.168.0.241/24`.

After installing the proxmox feeder vm we need to configure the in the next steps. Do not update the initial installation yet.

Phase 2: Setting up APT-Cacher-NG (Cache Server)

Before installing the cache, we will optimize the Debian mirror selection to ensure the fastest download speeds.

2.1 Optimization: Selecting the Fastest Mirror

Install `netselect-apt` to automatically determine the best mirror for Debian 13.

```
# Install netselect-apt
sudo apt update
sudo apt install netselect-apt -y

# Find the fastest mirror for Debian 13 (Trixie)
# This creates a 'sources.list' file in the current directory
sudo netselect-apt trixie
```

```
# Backup existing sources and apply the new optimized list
sudo mv /etc/apt/sources.list /etc/apt/sources.list.bak
sudo mv sources.list /etc/apt/sources.list
sudo apt update
```

2.2 Install and Configure APT-Cacher-NG

Now, install the caching service which will use the fast mirrors selected above.

Installation

```
sudo apt install apt-cacher-ng -y
```

Configuration

Edit the configuration to ensure it allows Proxmox repositories:

```
sudo nano /etc/apt-cacher-ng/acng.conf
```

Ensure the following line is active to allow HTTPS tunneling if necessary:
`PassThroughPattern: .*`

Restart Service

```
sudo systemctl restart apt-cacher-ng
```

Phase 3: Populating the Cache

On your **Proxmox VE 9 "Feeder" VM**, tell APT to use the Cache Server.

1. Create a proxy configuration file:

```
echo 'Acquire::http::Proxy "http://<IP-OF-CACHE-SERVER>:3142";' | sudo
tee /etc/apt/apt.conf.d/00proxy
```

2. Run the updates to pull data into the cache:

```
apt update
apt dist-upgrade -y
```

```
Now, all downloaded `.deb` files are stored on the Debian 13 Cache VM
in `/var/cache/apt-cacher-ng`.
```

Phase 4: Exporting the Cache to the Air-Gapped System

Since the target system is air-gapped, we must physically move the data.

4.1 On the Internet-Connected Cache VM:

Compress the cache data:

```
sudo tar -cvzf pve-cache-export.tar.gz /var/cache/apt-cacher-ng
```

Copy `pve-cache-export.tar.gz` to a USB drive or mobile storage.

4.2 On the Air-Gapped Target System:

You need a machine (or LXC container) in the air-gapped network to act as the **Local Cache Server**.

1. Install a Debian 13 LXC or VM on your air-gapped Proxmox.
2. Install `apt-cacher-ng` (you might need to install this manually via `.deb` files once if the container isn't prepared).
3. Import the data:

```
# Extract the data to the correct location
sudo tar -xvzf /path/to/usb/pve-cache-export.tar.gz -C /
sudo chown -R apt-cacher-ng:apt-cacher-ng /var/cache/apt-cacher-ng
sudo systemctl restart apt-cacher-ng
```

Phase 5: Configuring Air-Gapped Proxmox Clients

Now, configure all your air-gapped Proxmox 9 nodes to use the internal cache server.

5.1 Set the Proxy

Edit `/etc/apt/apt.conf.d/00proxy` on **every** node:

```text
Acquire::http::Proxy "http://<INTERNAL-CACHE-LXC-IP>:3142";
```

5.2 Update Repository Sources

Ensure your `/etc/apt/sources.list` and `/etc/apt/sources.list.d/pve-enterprise.list` point to standard URLs. Even though there is no internet, `apt-cacher-ng` will trick APT into thinking it's talking to the real servers, while actually serving the files from the local disk.

Example for Proxmox 9 (No-Subscription):

```
deb [http://download.proxmox.com/debian/pve](http://download.proxmox.com/debian/pve) trixie pve-no-subscription
```

5.3 Run Update

```
apt update
apt dist-upgrade
```

Troubleshooting & Maintenance

- **Maintenance:** To update the air-gapped system again, repeat Phase 3 (on the internet host) and Phase 4 (transfer).
- **Disk Space:** Monitor `/var/cache/apt-cacher-ng`. You can use the web interface at `http://<cache-ip>:3142/acng-report.html` to manage the expiration of old packages.

Soll ich dir noch spezifische Konfigurationsparameter für die Proxmox Enterprise Repositories heraussuchen, falls du diese über den Cache spiegeln möchtest?