

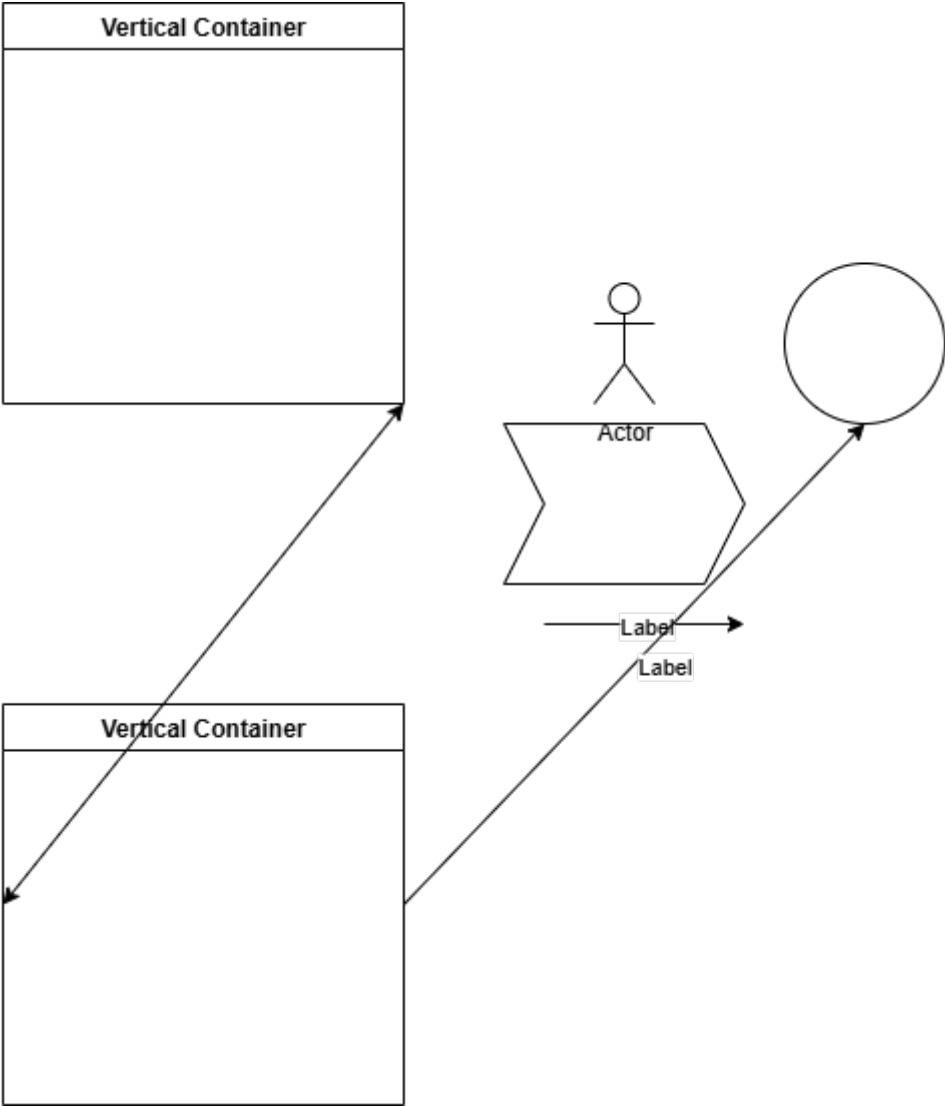
Proxmox Server und VMs

- [Proxmox Server Doku Konfiguration und Zweck](#)
- [Grafik Test](#)
- [Proxmox Markdown doku pro VM erstellen](#)
- [API-Token in Proxmox erstellen](#)

Proxmox Server Doku

Konfiguration und Zweck

Grafik Test



Proxmox Markdown doku pro VM erstellen

title:: "ChatGPT" source:: "https://chatgpt.com/c/68b16db1-0144-8322-86c2-76a0c43f9e30" author:: [[ChatGPT]] published:: created:: 2025-08-29 description:: "ChatGPT hilft dir, Antworten zu finden, dich inspirieren zu lassen und produktiver zu sein. Du kannst es ganz einfach kostenlos ausprobieren. Stelle irgendeine Frage und ChatGPT kann dir beim Schreiben, Lernen, Ideensammeln und mehr helfen." tags:: clippings Ja — das lässt sich automatisch generieren. Unten bekommst du ein fertiges Python-Skript, das per Proxmox-API alle VMs im Cluster einsammelt und **für jede VM eine eigene Markdown-Datei** (plus eine `README.md` mit Index) erzeugt. Labels/Navigationspfade sind auf Deutsch.

☐☐ Generator: Proxmox → Markdown (pro VM)

Voraussetzungen

- Python 3.9+
- `pip install proxmoxer requests`
- Ein **API-Token** in Proxmox (Rechte mindestens: `Sys.Audit`, `VM.Audit` auf Cluster/Nodes), z. B. unter: **Rechteverwaltung** → **API Tokens** (Benutzer `root@pam` oder ein Service-User).
- Optional: QEMU Guest Agent in VMs, um IPs sauber zu ziehen.

Nutzung

```
export PMX_HOST="proxmox.dein.tld:8006"
export PMX_USER="root@pam"           # oder 'automation@pve'
export PMX_TOKEN_NAME="md-gen"      # Token-ID (unter dem Benutzer)
```

```
export PMX_TOKEN_VALUE="xxxxxxxx-xxxx-xxxx-xxxx" # Token-Secret
export VERIFY_SSL="true" # 'false' falls Self-Signed
python3 generate_proxmox_vm_docs.py --out ./vm-docs
```

- Ergebnis: ein Ordner `vm-docs/` mit `README.md` (Index) und je VM eine Datei wie `101-dc01.md`.

```
# generate_proxmox_vm_docs.py
# -*- coding: utf-8 -*-

import os
import argparse
import pathlib
import datetime

from typing import Dict, Any, List, Tuple

from proxmoxer import ProxmoxAPI
import requests

# -----
# Konfiguration & Helpers
# -----

def getenv_bool(name: str, default: bool) -> bool:
    v = os.getenv(name)
    if v is None:
        return default
    return str(v).strip().lower() in ("1", "true", "yes", "y")

def connect_proxmox() -> ProxmoxAPI:
    host = os.getenv("PMX_HOST")
    user = os.getenv("PMX_USER")
    token_name = os.getenv("PMX_TOKEN_NAME")
    token_value = os.getenv("PMX_TOKEN_VALUE")
    verify_ssl = getenv_bool("VERIFY_SSL", True)

    if not all([host, user, token_name, token_value]):
        raise SystemExit(
            "Umgebungsvariablen fehlen: PMX_HOST, PMX_USER, PMX_TOKEN_NAME, PMX_TOKEN_VALUE"
        )

    # proxmoxer erwartet host ohne https:// und ohne Pfad
```

```

host_clean = host.replace("https://", "").rstrip("/")
return ProxmoxAPI(
    host_clean,
    user=user,
    token_name=token_name,
    token_value=token_value,
    verify_ssl=verify_ssl,
)

def safe(val, default=""):
    return default if val is None else val

def bool_to_yesno(v) -> str:
    return "Ja" if str(v).lower() in ("1", "true", "yes") else "Nein"

def to_markdown_filename(vmid: int, name: str) -> str:
    # einfache, saubere Dateinamen
    safe_name = "".join(ch if ch.isalnum() or ch in ("-", "_") else "-" for ch in name.strip())
    safe_name = "-".join(filter(None, safe_name.split("-")))
    return f"{vmid}-{safe_name or 'vm'}.md"

def flatten_ips_from_agent(interfaces: List[Dict[str, Any]]) -> List[str]:
    ips = []
    for iface in interfaces or []:
        for addr in iface.get("ip-addresses", []):
            ip = addr.get("ip-address")
            if ip and not ip.startswith("fe80"): # Link-Local filtern
                ips.append(ip)
    # deduplizieren, Reihenfolge stabil
    seen = set()
    uniq = []
    for ip in ips:
        if ip not in seen:
            seen.add(ip)
            uniq.append(ip)
    return uniq

def read_guest_agent_ips(proxmox: ProxmoxAPI, node: str, vmid: int) -> Tuple[List[str], str]:
    # Versucht, IPs via QGA zu holen
    try:

```

```

res = proxmox.nodes(node).qemu(vmid).agent.get("network-get-interfaces")
if isinstance(res, dict) and "result" in res:
    ips = flatten_ips_from_agent(res["result"])
    return ips, ""
return [], ""
except requests.exceptions.HTTPError as e:
    # Häufiger Fehler: 501 QGA not running
    return [], f"QEMU Guest Agent nicht verfügbar ({e.response.status_code})."
except Exception as e:
    return [], f"QGA-Fehler: {e}"

def read_snapshots(proxmox: ProxmoxAPI, node: str, vmid: int) -> List[Dict[str, Any]]:
    try:
        return proxmox.nodes(node).qemu(vmid).snapshot.get()
    except Exception:
        return []

# -----
# Markdown-Generator
# -----
def vm_to_markdown(
    now: datetime.datetime,
    cluster_name: str,
    node: str,
    vm: Dict[str, Any],
    config: Dict[str, Any],
    status: Dict[str, Any],
    ips: List[str],
    agent_note: str,
    snapshots: List[Dict[str, Any]],
) -> str:
    name = safe(vm.get("name"), f"vm-{vm.get('vmid')}")
    vmid = vm.get("vmid")
    cpu = safe(config.get("cores"), status.get("cpus"))
    memory_mb = safe(config.get("memory"), status.get("maxmem", 0) // (1024 * 1024))
    disks_lines = []
    # Disk-Infos aus config: virtioN / scsiN / sataN etc.
    for key, val in sorted(config.items()):
        if any(key.startswith(prefix) for prefix in ("virtio", "scsi", "sata", "ide")) and isinstance(val, str):
            # Format z.B.: "local-lvm:vm-101-disk-0,size=50G"

```

```

    parts = val.split(",")
    size = next((p.split("=")[1] for p in parts if p.startswith("size=")), "n/a")
    storage = parts[0] if ":" in parts[0] else "n/a"
    disks_lines.append(f"- {key}: {size} ({storage})")
if not disks_lines:
    disks_lines.append("- n/a")

# Netz / Bridge / VLAN
net_lines = []
for key, val in sorted(config.items()):
    if key.startswith("net") and isinstance(val, str):
        # Beispiel: "virtio=DE:AD:BE:EF:00:01,bridge=vibr0,tag=10"
        parts = dict(p.split "=", 1) for p in val.split(",") if "=" in p
        typ = "virtio" # key 'net0', 'net1' trägt NIC-Index, der Typ steht in value vor '='
        mac = val.split(",")[0].split("=")[1] if "=" in val.split(",")[0] else ""
        bridge = parts.get("bridge", "n/a")
        vlan = parts.get("tag", parts.get("trunks", "n/a"))
        net_lines.append(f"- Typ: {typ}, MAC: {mac}, Bridge: {bridge}, VLAN: {vlan}")
if not net_lines:
    net_lines.append("- n/a")

# Laufende Dienste → Platzhalter (nicht per API ermittelbar)
ports_placeholder = "- (bitte ergänzen)"

# Snapshots
snap_lines = []
for s in snapshots:
    sname = s.get("name", "snapshot")
    stime = s.get("snaptime")
    ts = datetime.datetime.fromtimestamp(stime).strftime("%Y-%m-%d %H:%M") if stime else "n/a"
    snap_lines.append(f"- {sname} ({ts})")
if not snap_lines:
    snap_lines.append("- Keine Snapshots gefunden")

# Agent-Notiz
agent_hint = f"\n> Hinweis: {agent_note}\n" if agent_note else ""

# Status/Node
node_line = f"{node} (Cluster „{cluster_name}“)"

```

Markdown

md = f"### VM-Dokumentation: {name}"

Stand: {now.strftime("%Y-%m-%d %H:%M")}

Allgemeine Informationen

- **Hostname:** {name}
- **VM-ID (Proxmox):** {vmid}
- **Rolle / Zweck:** *(bitte ergänzen)*
- **Betriebssystem:** {safe(config.get("ostype"), "unbekannt")}
- **Version / Build:** *(bitte ergänzen)*
- **IP-Adresse(n):** {" ".join(ips) if ips else "(nicht ermittelbar)"}
- **MAC-Adresse:** *(siehe Netzwerkadapter)*
- **Standort (Cluster/Node):** {node_line}{agent_hint}

Ressourcen

- **vCPU:** {cpu}
- **RAM (GB):** {round(int(memory_mb)/1024, 2) if memory_mb else "n/a"}
- **Disk(s):**
{chr(10).join(" " + line for line in disks_lines)}
- **Netzwerkadapter:**
{chr(10).join(" " + line for line in net_lines)}

Software & Dienste

- **Installierte Hauptsoftware:** *(bitte ergänzen)*
- **Laufende Dienste / Ports:**
{ports_placeholder}

Backups & Snapshots

- **Backup-Strategie:** *(z. B. täglich, wöchentlich – bitte ergänzen)*
- **Letztes Backup:** *(bitte ergänzen / aus Backup-Tool)*
- **Snapshots:**
{chr(10).join(" " + line for line in snap_lines)}

Monitoring & Logging

- **Überwachung durch:** *(z. B. Wazuh, Prometheus – bitte ergänzen)*
- **Log-Speicherort:** *(z. B. Syslog, Wazuh, Pfade – bitte ergänzen)*

Verantwortlichkeiten

- **Verantwortlicher Admin:** Frank Ahorn
- **Owner / Fachabteilung:** IT

```

## Notizen / Besonderheiten
- *(z. B. Startreihenfolge, Affinität/Pinning, Wartungsfenster)*
"""
    return md

# -----
# Hauptlogik
# -----
def main():
    parser = argparse.ArgumentParser(description="Erzeugt pro VM Markdown-Dokumente aus dem Proxmox-Cluster.")
    parser.add_argument("--out", default="./vm-docs", help="Ausgabeordner (default: ./vm-docs)")
    parser.add_argument("--include-stopped", action="store_true", help="Auch gestoppte VMs dokumentieren")
    parser.add_argument("--cluster-name", default="", help="Clusternamen überschreiben (optional)")
    args = parser.parse_args()

    out_dir = pathlib.Path(args.out)
    out_dir.mkdir(parents=True, exist_ok=True)

    proxmox = connect_proxmox()
    now = datetime.datetime.now()

    # Clusternamen
    cluster_name = args.cluster_name or ""
    try:
        cinfo = proxmox.cluster.status.get()
        # Ein Eintrag mit 'type' == 'cluster' enthält meist 'name'
        for item in cinfo:
            if item.get("type") == "cluster" and item.get("name"):
                cluster_name = item["name"]
                break
    except Exception:
        pass
    cluster_name = cluster_name or "Proxmox-Cluster"

    index_lines = [
        f"# VM-Dokumentation - Index ({cluster_name})",
        "",
        f"*Stand: {now.strftime('%Y-%m-%d %H:%M')}*",

```

```

    "",
    "| VM-ID | Name | Node | Datei | Status | vCPU | RAM (GiB) |",
    "|-----|-----|-----|-----|-----|-----|-----:|",
]

vm_count = 0

nodes = proxmox.nodes.get()
for n in nodes:
    node = n["node"]
    # nur QEMU-VMs (keine Container)
    vms = proxmox.nodes(node).qemu.get()
    for vm in vms:
        vmid = vm["vmid"]
        # Status lesen
        try:
            status = proxmox.nodes(node).qemu(vmid).status.current.get()
        except Exception:
            # Minimal-Status fallback
            status = {}

        if not args.include_stopped and status.get("status") != "running":
            continue

        # Config lesen
        try:
            config = proxmox.nodes(node).qemu(vmid).config.get()
        except Exception:
            config = {}

        # IPs via QGA (wenn möglich)
        ips, agent_note = read_guest_agent_ips(proxmox, node, vmid)

        # Snapshots
        snapshots = read_snapshots(proxmox, node, vmid)

        # Markdown rendern
        md = vm_to_markdown(
            now=now,
            cluster_name=cluster_name,

```

```

node=node,
vm=vm,
config=config,
status=status,
ips=ips,
agent_note=agent_note,
snapshots=snapshots,
)

# Datei schreiben
fname = to_markdown_filename(vmid, vm.get("name") or str(vmid))
(out_dir / fname).write_text(md, encoding="utf-8")

# Indexzeile
cpu = config.get("cores", status.get("cpus", "n/a"))
mem_gib = round((config.get("memory") or status.get("maxmem", 0) // (1024 * 1024)) / 1024, 2) if
(config.get("memory") or status.get("maxmem")) else "n/a"
index_lines.append(
    f"| {vmid} | {vm.get('name', '')} | {node} | [{fname}][.{fname}] | {status.get('status','n/a')} | {cpu}
| {mem_gib} |"
)
vm_count += 1

# Index schreiben
(out_dir / "README.md").write_text("\n".join(index_lines) + "\n", encoding="utf-8")

print(f"Fertig. {vm_count} VM-Dokumente in {out_dir} erzeugt.")

if __name__ == "__main__":
    main()

```

Was das Skript sammelt

- **Allgemeines:** Name, VM-ID, Node, Clustername, OS-Typ (aus `ostype`)
- **Ressourcen:** vCPU, RAM, Disks (Größe/Storage aus `virtio*/scsi*/sata*`)
- **Netzwerk:** MAC/Bridge/VLAN pro Adapter
- **IP-Adressen:** wenn QEMU Guest Agent verfügbar
- **Snapshots:** Liste mit Zeitstempel

- **Index:** Übersichtstabelle mit Links zu allen Blättern

“ Backups zieht Proxmox nicht per VM-API direkt aus, da sie meist über `vzdump` /Jobs laufen – im Blatt bleibt dafür ein Platzhalter (du kannst hier später Daten aus deinem Backup-Tool eintragen).

Optional

- **Gestoppte VMs einschließen:** `--include-stopped`
- **Clusternamen überschreiben:** `--cluster-name "prod-cluster"`

Wenn du willst, passe ich dir das Skript noch an (z. B. **LXC-Container** zusätzlich, **Backup-Logs** einlesen, oder **GLPI-Felder** automatisch vorbereiten).

☐☐ API-Token in Proxmox erstellen

title:: "ChatGPT" source:: "https://chatgpt.com/c/68b16db1-0144-8322-86c2-76a0c43f9e30"
author:: [[ChatGPT]] published:: created:: 2025-08-29 description:: "ChatGPT hilft dir, Antworten zu finden, dich inspirieren zu lassen und produktiver zu sein. Du kannst es ganz einfach kostenlos ausprobieren. Stelle irgendeine Frage und ChatGPT kann dir beim Schreiben, Lernen, Ideensammeln und mehr helfen." tags:: clippings

1. Anmelden

- Melde dich auf der **Proxmox WebGUI** an (`https://<dein-host>:8006`).

2. Benutzer auswählen

- Links im Baum: **Rechenzentrum** → **Berechtigungen** → **Benutzer**.
- Wähle den Benutzer, unter dem du arbeiten willst, z. B. `root@pam`.
- ☐☐ Für Automatisierungen empfiehlt es sich, einen separaten Benutzer anzulegen, z. B. `automation@pve`.

3. Token anlegen

- Klick auf den Benutzer → Tab **API Token** → **Hinzufügen**.
- Gib ein:
- **Token-ID** (frei wählbarer Name, z. B. `md-gen`).
- Häkchen bei **Unverfallbar** setzen, wenn das Token nicht automatisch ablaufen soll.
- Optional: Ablaufdatum, wenn du es zeitlich begrenzen willst.
- Bestätigen mit **Hinzufügen**.

4. Rechte vergeben

- Das Token selbst hat keine Rechte, die musst du separat vergeben.
- Gehe zu: **Rechenzentrum** → **Berechtigungen** → **Berechtigungen** → **Hinzufügen** → **API Token Berechtigung**.
- Wähle:
- **Pfad:** / (oder gezielt /vms wenn du nur VM-Zugriff brauchst).
- **API Token:** deinen Benutzer + das neu erstellte Token.
- **Rolle:** mindestens PVEVM.Audit und Sys.Audit (damit kann man Configs und Status lesen). Falls du später auch Snapshots oder Backups auslesen willst: VM.Backup / Datastore.Audit ergänzen.
- Mit **Hinzufügen** speichern.

5. Token-Secret notieren

- Nach dem Erstellen wird dir **Token-ID** und **Secret** angezeigt.
- ⚠ Das Secret siehst du **nur einmal!** Gleich kopieren und sicher ablegen.

📄 Ergebnis

- Dein API-Token besteht aus:
- Benutzer: root@pam oder z. B. automation@pve
- Token-ID: md-gen
- Token-Secret: ein langer Schlüssel (UUID-ähnlich)

Im Script/Environment trägst du es so ein:

```
export PMX_HOST="proxmox.dein.tld:8006"
export PMX_USER="root@pam"
export PMX_TOKEN_NAME="md-gen"
export PMX_TOKEN_VALUE="xxxxxxxx-xxxx-xxxx-xxxx"
export VERIFY_SSL="true"
```

📄 Soll ich dir auch gleich ein Beispiel geben, wie du einen **eigenen Service-User** anlegst (anstatt root@pam zu benutzen)? Das ist meist die sicherere Variante.

Du:

ja bitte

ChatGPT:

Perfekt `[]` - einen **eigenen Service-User** zu verwenden ist deutlich sicherer als `root@pam`.

Hier die Schritte:

`[]` Eigenen Proxmox Service-User mit API-Token erstellen

1. Benutzergruppe für Automation anlegen

1. In der WebGUI: **Rechenzentrum** → **Berechtigungen** → **Gruppen**.
2. **Gruppe hinzufügen** - z. B. Name `automation`.

2. Benutzer anlegen

1. Gehe zu **Rechenzentrum** → **Berechtigungen** → **Benutzer**.
2. **Hinzufügen**:

- **Benutzername:** `automation`
- **Realm:** `pve` (damit ist es ein interner Proxmox-User, kein PAM/LDAP).
- **Kommentar:** „Automation Account für API-Zugriff“
- Passwort kannst du setzen, ist aber für Token-Nutzung nicht zwingend.

3. Benutzer zur Gruppe `automation` hinzufügen.

3. Rechte für die Gruppe vergeben

1. Gehe zu **Rechenzentrum** → **Berechtigungen** → **Berechtigungen**.
2. **Hinzufügen** → **Gruppe Berechtigung**:

- **Pfad:** `/` (für kompletten Zugriff lesen - alternativ `/vms` oder `/storage` enger einschränken).
- **Gruppe:** `automation`.

- **Rolle:** mindestens:
- `PVEVM.Audit` (VM-Infos lesen)
- `Sys.Audit` (Cluster/Node Infos lesen)
- Optional: `Datastore.Audit` (wenn du Storage/Backup-Infos brauchst)

☐☐ Damit kann der Account **lesen**, keine VM starten/stoppen.

4. API-Token für den User erzeugen

1. Wieder zu **Rechenzentrum** → **Berechtigungen** → **Benutzer**.
2. Wähle den User `automation@pve`.
3. Tab **API Token** → **Hinzufügen**:

- **Token-ID:** z. B. `md-gen`.
- Häkchen bei **Unverfallbar** setzen (außer du willst ein Ablaufdatum).

4. Nach dem Erstellen bekommst du:

- **Token-ID:** `automation@pve!md-gen`
- **Token-Secret:** einmalig sichtbar → sofort kopieren!

5. Token ins Script eintragen

In deiner Shell/Umgebung:

```
export PMX_HOST="proxmox.dein.tld:8006"
export PMX_USER="automation@pve"
export PMX_TOKEN_NAME="md-gen"
export PMX_TOKEN_VALUE="xxxxxxxx-xxxx-xxxx-xxxx"
export VERIFY_SSL="true"
```

☐☐ Vorteil

- Kein Root-Zugriff nötig

- Rechte sauber begrenzt (nur Lesen)
- Token kann jederzeit widerrufen oder neu erstellt werden

☐ Soll ich dir ein **Rollenprofil** bauen (z. B. `VM.DokuLesen`), das nur die **minimal nötigen Rechte** fürs Dokumentations-Skript hat? Dann kannst du es an den Automation-User binden, ohne zu viel freizugeben.

Du:

Ja bitte

ChatGPT:

Super — hier ist ein **minimal sicheres Rollenprofil** für dein Doku-Skript. Es erlaubt reines Lesen von Cluster/VM/Storage-Infos und (optional) QEMU-Guest-Agent-Abfragen für IPs.

☐ Rolle „VM.DokuLesen“

Enthaltene Privilegien

- **Sys.Audit** → Cluster/Nodes lesen
- **PVEVM.Audit** → VM-Konfig & Status lesen (inkl. Snapshots)
- **Datastore.Audit** → Storage/Volumes lesen (nur lesend)
- **VM.Monitor** (*optional, empfohlen*) → nötig für **Guest-Agent** API (`network-get-interfaces`), um IPs auszulesen

“ Wenn du **keine** IPs via Guest-Agent brauchst, lass **VM.Monitor** weg.

Variante A: Klickpfad (GUI)

1. Rechenzentrum → Berechtigungen → Rollen → Hinzufügen

- **Name:** `VM.DokuLesen`
- **Rechte anhängen:** `Sys.Audit`, `PVEVM.Audit`, `Datastore.Audit`, optional `VM.Monitor`
- **Speichern**

2. Rechenzentrum → Berechtigungen → Berechtigungen → Hinzufügen → Gruppe Berechtigung

- **Pfad:** / (oder enger, z. B. /vms und /storage)
- **Gruppe:** automation
- **Rolle:** VM.DokuLesen
- **Hinzufügen**

(Du hast die Gruppe/den User automation@pve ja bereits erstellt.)

Variante B: CLI (schnell & reproduzierbar)

```
# Rolle anlegen (mit Guest-Agent-Recht)
pveum roleadd VM.DokuLesen -privs "Sys.Audit,PVEVM.Audit,Datastore.Audit,VM.Monitor"

# Falls ohne Guest-Agent:
# pveum roleadd VM.DokuLesen -privs "Sys.Audit,PVEVM.Audit,Datastore.Audit"

# Gruppe (falls noch nicht vorhanden)
pveum groupadd automation -comment "Automation Gruppe"

# User (falls noch nicht vorhanden)
# pveum useradd automation@pve -password <GEHEIM> -comment "Automation User"

# Rolle der Gruppe auf / zuweisen (Read-only, clusterweit)
pveum aclmod / -group automation -role VM.DokuLesen
```

Sicherheits-Hinweise

- **Least Privilege:** Weisen, wenn möglich, nur auf die benötigten Teilpfade zu (z. B. nur bestimmter Node/Pool/Storage).
- **Tokens statt Passwort:** Für das Skript ausschließlich **API-Token** von automation@pve verwenden.
- **Rotation:** Token mit Ablaufdatum erstellen oder regelmäßig neu ausstellen.

- **Audit:** Änderungen an Rechten protokollieren (z. B. Änderungs-Ticket).
-

Wenn du willst, gebe ich dir noch eine „**Härtungs-Checkliste**“ (Self-Signed TLS, Token-Ablauf, engere ACL-Pfadwahl pro Pool/Node, Logging), die du einfach abarbeiten kannst.