Dokus und Anleitungen

Anleitungen für bestimmte Tasks

- SSH Alias anlegen
- Anlegen von SSH Schlüsseln
- SSH ID unter Windows kopieren
- AWX Installieren
- AWX direkt auf der Maschine
- Korrupte GPT Paratition reparieren
- Logs in Docker begrenzen
- Virtuelle Workstation mit Proxmox und mehreren Screens via X11
- Virtuelle Workstation mit Proxmox und mehreren Screens via Wayland
- SQL Server Logs
- BtrFS Installation Kubuntu 25.10
- Linux und Microsoft AD
    - AD Beitritt Ubuntu LTS
    - Zertifikat mit AD erzeugen
    - Linux System zu AD hinzufügen
    - AD Beitritt Open Suse 16

SSH Alias anlegen

Unter Windows kannst du für SSH-Server-Aliase die Datei `~/.ssh/config` verwenden – genauso wie unter Linux oder macOS. Damit kannst du z. B. einem GitLab-Server auf einem speziellen Port (wie 2022) einen Alias geben.

So richtest du den Alias ein:

1. Öffne den Windows-Dateiexplorer und gehe zu deinem Benutzerverzeichnis:

   ```
   C:\Users\<deinBenutzername>\.ssh
   ```

   Falls der Ordner `.ssh` noch nicht existiert, kannst du ihn anlegen.
2. Öffne (oder erstelle) darin die Datei `config` (ohne Dateiendung) mit einem Texteditor wie Notepad oder VSCode.
3. Füge folgenden Block hinzu (Passe die Werte an deine Umgebung an):

   ```
   Host gitlab
       HostName gitlab.example.com
       Port 2022
       User git
       IdentityFile ~/.ssh/id_rsa
   ```

   - **Host**: ist der Alias, den du später im `ssh`- oder `git`-Befehl verwenden kannst (z. B. `gitlab`).
   - **HostName**: ist der tatsächliche Hostname oder die IP-Adresse deines GitLab-Servers.
   - **Port**: der Port, auf dem der SSH-Server läuft (hier 2022).
   - **User**: der Benutzername für den Login (bei GitLab i.d.R. `git`).
   - **IdentityFile**: optional, gibt den privaten SSH-Schlüssel an, falls du nicht den Standardpfad verwendest.
4. Speichere die Datei.

Beispiel-Verwendung:

Statt so:

```
ssh -p 2022 git@gitlab.example.com
```

einfach so:

```
ssh gitlab
```

Oder beim Klonen eines Repositories:

```
git clone git@gitlab:username/repo.git
```

Tipp

Wenn dein SSH-Schlüssel einen anderen Namen hat (z. B. `id_gitlab_rsa`), gib ihn explizit bei `IdentityFile` an:

```
IdentityFile ~/.ssh/id_gitlab_rsa
```

Hinweis zu Pfadangaben unter Windows

`~` funktioniert normalerweise auch unter Windows, aber wenn du Probleme hast, ersetze `~` durch den vollständigen Pfad, z. B.:

```
IdentityFile C:/Users/DeinBenutzername/.ssh/id_rsa
```

Möchtest du, dass ich ein konkretes Beispiel mit deinen Daten schreibe?

Anlegen von SSH Schlüsseln

Um unter Windows sowohl einen ED25519- als auch einen RSA-SSH-Schlüssel zu erzeugen, gehst du am besten folgendermaßen vor:

---

1. Voraussetzungen

- **Windows 10 (ab Version 1803) oder Windows 11** – hier ist der OpenSSH-Client bereits integriert.
- **PowerShell** oder **Eingabeaufforderung**, idealerweise als Administrator gestartet (für Agent-Konfiguration).

---

2. Öffne PowerShell

1. Klicke auf **Start** → tippe **PowerShell** → **Rechtsklick** → **Als Administrator ausführen**.
2. Wechsel (falls gewünscht) in dein Benutzer-.ssh-Verzeichnis:

```
cd $HOME\.ssh
```

> **Hinweis:** Falls das Verzeichnis noch nicht existiert, leg es an mit `mkdir $HOME\.ssh`.

---

3. Erzeugen eines ED25519-Schlüssels

1. Führe den Befehl aus:

```
ssh-keygen -t ed25519 -C "dein.comment@beispiel.de"
```

2. Du wirst gefragt nach:
   - **Dateiname** (Standard: `%USERPROFILE%\.ssh\id_ed25519`) → einfach mit **Enter** bestätigen, falls du den Standard willst.
   - **Passphrase** (optional): Tipp eine starke Passphrase ein oder leer lassen → Enter.

Dadurch erhältst du:

- **Privat-Key:** `%USERPROFILE%\.ssh\id_ed25519`
- **Public-Key:** `%USERPROFILE%\.ssh\id_ed25519.pub`

---

4. Erzeugen eines RSA-Schlüssels

1. Führe den Befehl aus:

```
ssh-keygen -t rsa -b 4096 -C "dein.comment@beispiel.de"
```

2. Abfragen:
   - **Dateiname** (Standard: `%USERPROFILE%\.ssh\id_rsa`) → Enter.
   - **Passphrase** (empfohlen): Eingeben oder leer lassen → Enter.

Erzeugt:

- **Privat-Key:** `%USERPROFILE%\.ssh\id_rsa`
- **Public-Key:** `%USERPROFILE%\.ssh\id_rsa.pub`

---

5. SSH-Agent starten und Schlüssel laden (optional)

Damit du nicht bei jeder SSH-Verbindung dein Passwort eingeben musst, kannst du den **ssh-agent** nutzen:

1. Starte den Agent:

```
Start-Service ssh-agent
```

2. Konfiguriere ihn so, dass er immer startet:

```
Set-Service -Name ssh-agent -StartupType Automatic
```

3. Lade deine Schlüssel in den Agenten:

```
ssh-add $HOME\.ssh\id_ed25519
ssh-add $HOME\.ssh\id_rsa
```

   Du wirst nach den Passphrasen gefragt (falls gesetzt).

---

6. Public Key auf dem Server hinterlegen

1. Zeige den Public Key an:

```
type $HOME\.ssh\id_ed25519.pub
# bzw.
type $HOME\.ssh\id_rsa.pub
```

2. Kopiere die Ausgabe und füge sie in `~/.ssh/authorized_keys` auf deinem Ziel-Server ein.

Zusammenfassung der Befehle

```
# 1. .ssh-Verzeichnis anlegen (falls nötig)
mkdir $HOME\.ssh

# 2. ED25519-Schlüssel
ssh-keygen -t ed25519 -C "dein.comment@beispiel.de"

# 3. RSA-Schlüssel (4096 Bit)
ssh-keygen -t rsa -b 4096 -C "dein.comment@beispiel.de"

# 4. SSH-Agent starten & Schlüssel hinzufügen
Start-Service ssh-agent
Set-Service -Name ssh-agent -StartupType Automatic
ssh-add $HOME\.ssh\id_ed25519
ssh-add $HOME\.ssh\id_rsa

# 5. Public Key anzeigen
```

```
type $HOME\.ssh\id_ed25519.pub
type $HOME\.ssh\id_rsa.pub
```

Damit hast du unter Windows erfolgreich sowohl einen ED25519- als auch einen RSA-SSH-Schlüssel erstellt und bereitgestellt. Viel Erfolg beim Einrichten deiner SSH-Verbindungen!

SSH ID unter Windows kopieren

Du hast unter Windows mehrere Möglichkeiten, dir das Linux-Tool **ssh-copy-id** zu “klonen” bzw. dessen Funktionalität nachzubauen:

---

1. Über Git Bash oder WSL direkt das Original nutzen

1. **Git for Windows** installieren (git-scm.com).
2. Git Bash öffnen – dort ist `ssh-copy-id` in der Regel bereits enthalten:

   ```
   ssh-copy-id -i ~/.ssh/id_ed25519.pub user@host
   ```

   Das kopiert deinen Public Key automatisch nach `~/.ssh/authorized_keys` auf dem Zielserver.
3. **Oder WSL (Windows Subsystem for Linux)**:
   - In PowerShell als Admin:

     ```
     wsl --install
     ```

   - Danach unter Ubuntu/etc. per

     ```
     sudo apt update && sudo apt install openssh-client
     ssh-copy-id -i ~/.ssh/id_ed25519.pub user@host
     ```

---

2. Eigenes Powershell-Skript als „ssh-copy-id“ nachbauen

Wenn du keine Unix-Umgebung installieren willst, kannst du in deinem PowerShell-Profil (`$PROFILE`) eine Funktion anlegen:

```
function Copy-SshKey {
    [CmdletBinding()]
    param(
        [Parameter(Mandatory)]
        [string]$UserAtHost,                    # z.B. "alice@server.example.com"
        [string]$KeyFile = "$HOME\.ssh\id_rsa.pub" # Pfad zum Public Key
    )

    # Key einlesen
    $pubKey = Get-Content $KeyFile -Raw

    # Auf dem Remote-Host das .ssh-Verzeichnis anlegen und Key anhängen
    ssh $UserAtHost "mkdir -p ~/.ssh && echo '$pubKey' >> ~/.ssh/authorized_keys"
}
```

1. Öffne PowerShell und lade dein Profil neu mit:

   ```
   . $PROFILE
   ```

2. Führe dann aus:

   ```
   Copy-SshKey -UserAtHost "alice@server.example.com"
   ```

---

3. Das offizielle Skript aus GitHub klonen

Es gibt ein reines Shell-Skript von Kevin Burke, das du einfach unter Windows in Git Bash oder WSL nutzen kannst:

1. Verzeichnis anlegen und klonen:

   ```
   mkdir C:\tools\ssh-copy-id
   cd C:\tools\ssh-copy-id
   git clone https://github.com/kevinburke/ssh-copy-id.git .
   ```

2. (Optional) `C:\tools\ssh-copy-id` in deine PATH-Umgebungsvariable aufnehmen, damit `ssh-copy-id` überall verfügbar ist.
3. In Git Bash oder WSL aufrufen:

```
ssh-copy-id -i ~/.ssh/id_ed25519.pub alice@server.example.com
```

Zusammenfassung

- **Einfachster Weg:** Git Bash oder WSL installieren und das vorhandene `ssh-copy-id` nutzen.
- **Windows-Native:** Mit einer kleinen PowerShell-Funktion das Key-Kopieren automatisieren.
- **Skript-Klon:** Das offizielle Shell-Skript aus GitHub klonen und in PATH legen.

Wähle die Methode, die am besten zu deinem Workflow passt!

AWX Installieren

Hier eine erprobte Schritt-für-Schritt-Anleitung zur Installation der Open-Source-Version von AWX (aktuell v17.1.0) mit Docker Compose – ganz ohne Kubernetes, offiziell zwar nur für Tests gedacht, in der Praxis aber voll funktionsfähig.

---

1. System-Abhängigkeiten installieren

Unter Ubuntu/Debian

```
sudo apt update
sudo apt install -y git gcc g++ python3-pip ansible curl
pip3 install docker docker-compose
```

(mpolinowski.github.io, docs.zpesystems.com)

Unter RHEL/CentOS 8

```
sudo dnf install -y git gcc gcc-c++ python3-pip ansible curl
sudo alternatives --set python /usr/bin/python3
pip3 install docker docker-compose
```

(mpolinowski.github.io, docs.zpesystems.com)

> Damit sind Docker, Docker Compose, Ansible und die nötigen Build-Tools auf dem Host.

---

2. AWX-Quellcode klonen & zum Installer wechseln

```
git clone -b 17.1.0 https://github.com/ansible/awx.git
cd awx/installer
```

(docs.zpesystems.com, mpolinowski.github.io)

---

3. Secret-Key erzeugen

Der Installer benötigt einen 30-stelligen Schlüssel für die interne Verschlüsselung.
Wahlweise mit OpenSSL:

```
openssl rand -base64 30 > secret_key.txt
```

(mpolinowski.github.io)

Oder mit pwgen:

```
pwgen -N1 -s30 > secret_key.txt
```

(docs.zpesystems.com)

---

4. Inventardatei anpassen

Öffne `installer/inventory` in deinem Lieblings-Editor und setze mindestens diese Werte im Block `[all:vars]`:

```
# Web-Zugriff
host_port=80
host_port_ssl=443

# Admin-Account
```

```
admin_user=admin
admin_password=MeinSicheresPasswort

# Secret Key (aus secret_key.txt)
secret_key=<Inhalt von secret_key.txt>

# (Optional) Pfade für persistente Daten
postgres_data_dir="/opt/awx/pgdocker"
project_data_dir="/opt/awx/projects"
docker_compose_dir="/opt/awx/awxcompose"
```

(docs.zpesystems.com, mpolinowski.github.io)

> Achte darauf, dass die Verzeichnisse existieren und Docker-Prozesse darauf schreiben dürfen (z.B. `sudo mkdir -p /opt/awx/{pgdocker,projects,awxcompose}`).

---

5. Installation per Ansible-Playbook starten

```
ansible-playbook -i inventory install.yml
```

(docs.zpesystems.com, mpolinowski.github.io)

Der Playbook-Lauf erzeugt via Docker Compose Container für PostgreSQL, Redis, das AWX-Web-UI und den Task-Worker.

---

6. Kontrolle & Erster Zugriff

1. **Status prüfen**

   ```
   docker ps
   ```

2. **Installations-Logs beobachten**

```
docker logs -f awx_task
```

Warte, bis dort die Datenbank-Migrations-Tasks erfolgreich durchgelaufen sind.

3. **Web-UI öffnen**
   Rufe im Browser auf:

   ```
   http://<Deine-Server-IP>:80
   ```

   Login mit `admin / MeinSicheresPasswort`.

   (docs.zpesystems.com, mpolinowski.github.io)

Fertig! Du hast nun AWX 17.1.0 vollständig in Docker-Containern laufen, ganz ohne Kubernetes.

AWX direkt auf der Maschine

Hier eine komplette, kopierfertige Anleitung für die native Installation von AWX 17.1.0 auf Ubuntu 24.04 ohne Docker, mit bestehendem Nginx Proxy Manager als Reverse-Proxy. Einfach der Reihe nach abarbeiten:

---

1. System-Pakete installieren

```
sudo apt update
sudo apt install -y \
  git build-essential libssl-dev libffi-dev python3-venv python3-pip \
  libpq-dev postgresql postgresql-contrib redis-server rabbitmq-server \
  nginx curl pwgen
```

---

2. PostgreSQL-Datenbank & RabbitMQ anlegen

a) PostgreSQL

```
sudo -u postgres psql <<'EOF'
CREATE USER awx WITH PASSWORD 'AWX_db_passwort';
CREATE DATABASE awx OWNER awx ENCODING 'UTF8';
\q
EOF
```

b) RabbitMQ

```
sudo rabbitmqctl add_user awx MQ_passwort
sudo rabbitmqctl set_user_tags awx administrator
```

```
sudo rabbitmqctl set_permissions -p / awx ".*" ".*" ".*"
```

3. AWX-Quellcode auschecken

```
sudo mkdir -p /opt/awx
sudo chown $USER:$USER /opt/awx
cd /opt/awx
git clone -b 17.1.0 https://github.com/ansible/awx.git .
```

4. Python-Virtualenv & Abhängigkeiten

```
cd /opt/awx
python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip
pip install -r requirements/requirements.txt
pip install psycopg2-binary redis kombu django gunicorn
deactivate
```

5. AWX-Settings anlegen

Erstelle `~/awx_settings.py` mit:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'awx',
        'USER': 'awx',
        'PASSWORD': 'AWX_db_passwort',
        'HOST': 'localhost',
```

```
        'PORT': '5432',
    }
}

BROKER_URL = 'amqp://awx:MQ_passwort@localhost:5672//'

CACHES = {
    'default': {
        'BACKEND': 'django_redis.cache.RedisCache',
        'LOCATION': 'redis://127.0.0.1:6379/1',
        'OPTIONS': {'CLIENT_CLASS': 'django_redis.client.DefaultClient'},
    }
}

SECRET_KEY = '$(pwgen -N1 -s32)'
ALLOWED_HOSTS = ['*']
```

Exportiere:

```
export AWX_SETTINGS_MODULE=awx_settings
```

---

6. Datenbank-Migration & Admin anlegen

```
cd /opt/awx/awx
source /opt/awx/.venv/bin/activate
./manage.py makemigrations
./manage.py migrate
./manage.py createsuperuser --username=admin --email=admin@example.com
deactivate
```

---

7. Gunicorn als systemd-Service

Erstelle `/etc/systemd/system/awx-gunicorn.service`:

```
[Unit]
Description=AWX Gunicorn Service
After=network.target

[Service]
User=www-data
Group=www-data
WorkingDirectory=/opt/awx/awx
Environment="AWX_SETTINGS_MODULE=awx_settings"
Environment="PATH=/opt/awx/.venv/bin"
ExecStart=/opt/awx/.venv/bin/gunicorn \
    --access-logfile - --workers 3 --bind 127.0.0.1:8052 awx.wsgi:application

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

Aktivieren und starten:

```
sudo systemctl daemon-reload
sudo systemctl enable --now awx-gunicorn
```

---

8. Statische Dateien sammeln

```
cd /opt/awx/awx
source /opt/awx/.venv/bin/activate
./manage.py collectstatic --noinput
deactivate
```

Die Assets liegen nun in `/opt/awx/awx/ui/static/`.

---

9. Nginx Proxy Manager konfigurieren

Im NPM-WebUI:

1. **Proxy Host** anlegen
   - **Domain Names**: `awx.deine-domain.tld`
   - **Scheme**: `http`
   - **Forward Hostname/IP**: `127.0.0.1`
   - **Forward Port**: `8052`
2. **Advanced → Custom Nginx Configuration**

   ```
   location /static/ {
       alias /opt/awx/awx/ui/static/;
       access_log off;
       expires max;
   }
   ```

3. **SSL** (optional): Let’s Encrypt aktivieren, “Block Common Exploits” ankreuzen.

Speichern und testen.

---

10. Fertig & Zugriff

Rufe im Browser auf:

```
https://awx.deine-domain.tld/
```

Login mit `admin` und deinem Passwort.

—
Viel Erfolg!

Korrupte GPT Paratition reparieren

Hier eine kompakte Schritt-für-Schritt-Anleitung, wie du unter Windows mit GPT fdisk (gdisk64 oder gdisk32) deine Backup-GPT reparierst:

---

1. Vorbereitung

1. Lade das Windows-Installationspaket von GPT fdisk (gdisk) von SourceForge herunter:
   https://sourceforge.net/projects/gptfdisk/files/gptfdisk/
2. Installiere es, dann findest du in der Installation die Programme **gdisk64.exe** (für 64 Bit) und **gdisk32.exe** (für 32 Bit).

---

2. Die richtige Datenträgernummer ermitteln

1. Öffne die **Datenträgerverwaltung** (Win+X → „Datenträgerverwaltung“).
2. Finde deine Linux-SSD in der Liste (z. B. „Datenträger 1“).
3. Merke dir die Nummer X (z. B. ist „Datenträger 1“ dann `\\.\physicaldrive1`).

---

3. GPT-Repair mit gdisk64.exe

1. Starte eine **Eingabeaufforderung als Administrator**.
2. Navigiere in das Verzeichnis, in dem **gdisk64.exe** liegt, oder gib den vollständigen Pfad an, z. B.:

   ```
   C:\Programme\gdisk\gdisk64.exe \\.\physicaldrive1
   ```

3. Im interaktiven Menü von gdisk:

   ```
   Command (? for help): v
   ```

   – Prüft die GPT-Integrität und meldet Inkonsistenzen (z. B. „Backup GPT header not present.“)
4. Wenn ein Fehler angezeigt wird, wechsle ins Recovery-Menü:

   ```
   Command (? for help): r
   Recovery/transformation command (? for help): v
   ```

   – Hier nochmal prüfen, dass nur der Backup-Header fehlt bzw. beschädigt ist.
5. Backup-Header neu schreiben:

   ```
   Recovery/transformation command (? for help): w
   ```

   – Bestätige mit `y`, damit gdisk den neuen Backup-Header ans Plattenende schreibt.
6. Beende gdisk: das Tool schreibt die Korrekturen und kehrt in die Eingabeaufforderung zurück.

---

4. Kontrolle & Neustart

1. Öffne erneut **Datenträgerverwaltung** oder eine Admin-Eingabeaufforderung und prüfe nochmal mit gdisk (`v`), dass keine Fehler mehr gemeldet werden.
2. Starte deinen Rechner neu und boote dein Linux: GRUB sollte jetzt wieder normal laden.

Kurz zusammengefasst

- **gdisk64 (oder gdisk32)** statt „gdisk“ aufrufen.
- In gdisk: `v` → `r` → `v` → `w`.
- Damit baust du den Backup-GPT-Header neu auf und stellst die GPT-Konsistenz wieder her.

So geht’s schnell und ohne Neuinstallation! Viel Erfolg.

Logs in Docker begrenzen

Ah ja, das Problem kenne ich – Docker schreibt die Container-Logs standardmäßig unlimitiert in **JSON-Dateien** unter `/var/lib/docker/containers/<container-id>/<container-id>-json.log`.
Wenn du viele Container laufen hast, können die sehr schnell die Platte volllaufen.

□□ Lösung: Log-Rotation aktivieren

Docker bringt das schon mit, du musst nur das **Logging-Driver** konfigurieren.

1. Global (für alle Container)

Du kannst in `/etc/docker/daemon.json` (oder erstellen, falls nicht existiert) Folgendes setzen:

```
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
  }
}
```

- `max-size`: maximale Größe einer einzelnen Logdatei (z. B. 10 MB)
- `max-file`: wie viele rotierte Logdateien behalten werden (z. B. 3 → max. 30 MB pro Container)

Danach Docker neu starten:

```
sudo systemctl restart docker
```

---

2. Nur für einen bestimmten Container

Beim Start eines Containers:

```
docker run -d \
  --name mein-container \
  --log-driver=json-file \
  --log-opt max-size=10m \
  --log-opt max-file=3 \
  nginx
```

---

3. Alternative Logging-Driver

Wenn du Logs nicht auf der Platte haben willst, kannst du auch andere Treiber nehmen, z. B. `syslog`, `journald` oder `none`.
Beispiel (Logs nur ins Journal):

```
docker run -d \
  --log-driver=journald \
  nginx
```

□□ Empfehlung: Setz am besten die **globale Einstellung** in `daemon.json`, dann musst du dich um einzelne Container nicht mehr kümmern.

Willst du, dass ich dir auch ein kleines **Einmal-Skript** bastle, das deine bestehenden übergroßen Logdateien automatisch kürzt oder leert, damit du gleich wieder Platz bekommst?

Virtuelle Workstation mit Proxmox und mehreren Screens via X11

1. Grundlegende Architektur

Komponente	Beschreibung
Proxmox-Host (Hetzner)	Ryzen 12-Core-Server mit Proxmox VE, UFW-Firewall (nur SSH & Proxmox-Ports).
Windows-VM	VM mit **2 virtuellen Monitoren**, erreichbar über **RDP/NoMachine**.
Linux-VM	VM mit **2 virtuellen Monitoren**, erreichbar über **NoMachine/x2go**.
Firewall (UFW)	Muss **RDP (3389), NoMachine (4000), x2go (22/SSH)** freigeben.
Zugang über Internet	**Reverse-Proxy (optional)** für sicheren Zugriff oder **WireGuard-VPN**.

---

2. Schritt-für-Schritt-Anleitung

□□ Schritt 1: VMs erstellen (Windows & Linux)

Windows-VM (mit 2 virtuellen Monitoren)

1. **VM in Proxmox erstellen**:
   - **OS**: Windows 10/11 (ISO von Microsoft).

- **CPU**: 4-6 Kerne (Ryzen 12-Core erlaubt das).
- **RAM**: 8-16 GB (je nach Nutzung).
- **Grafik**: **SPICE (QXL)** oder **Standard-VGA** (wir nutzen später **RDP/NoMachine**).
- **Festplatte**: 50-100 GB (SSD/NVMe empfohlen).
- **Netzwerk**: **Bridge (vmbr0)** für Internetzugriff.

2. **Windows installieren & Treiber aktualisieren**:
   - **VirtIO-Treiber** installieren (Download).
   - **SPICE-Guest-Tools** installieren (für bessere Grafikperformance):

     ```
     # In der VM (als Admin):
     msiexec /i spice-guest-tools-latest.exe
     ```

   - **2 virtuelle Monitore aktivieren** (über **RDP** oder **NoMachine**).

Linux-VM (mit 2 virtuellen Monitoren, z. B. Ubuntu/KDE)

1. **VM in Proxmox erstellen**:
   - **OS**: Ubuntu 22.04 LTS oder Debian 12 (mit **KDE/XFCE** für bessere Multi-Monitor-Unterstützung).
   - **CPU**: 4-6 Kerne.
   - **RAM**: 4-8 GB.
   - **Grafik**: **SPICE (QXL)** oder **VirtIO-GPU** (für bessere Performance).
   - **Festplatte**: 30-50 GB.
   - **Netzwerk**: **Bridge (vmbr0)**.
2. **Linux installieren & für Multi-Monitor vorbereiten**:
   - **SPICE-Treiber installieren**:

     ```
     sudo apt update && sudo apt install spice-vdagent spice-webdavd
     ```

   - **X11/KDE für 2 Monitore konfigurieren**:

     ```
     # Virtuellen Monitor hinzufügen (z. B. rechts vom Hauptbildschirm)
     xrandr --output Virtual1 --mode 1920x1080 --right-of eDP1
     ```

     (Für **permanente Einstellung** in `~/.xprofile` eintragen.)

□□ Schritt 2: Remote-Zugriff einrichten (RDP/NoMachine/x2go)

Option 1: RDP (für Windows-VM)

- **Vorteil**: Einfach, integriert, Multi-Monitor-Unterstützung.
- **Einrichtung**:
  1. **RDP in Windows aktivieren**:
     - `Win + R` → `sysdm.cpl` → **"Remoteverbindungen zulassen"**.
  2. **Port in UFW freigeben**:
     ```
     sudo ufw allow 3389/tcp comment "RDP for Windows VM"
     ```
  3. **Von außen verbinden**:
     - **Mit** `mstsc.exe /multimon` (Windows) oder **Remmina** (Linux).
     - **IP:Port** = `DEIN_SERVER_IP:3389` (oder **Port-Forwarding**, falls NAT im Spiel ist).

⚠ **Sicherheitshinweis**:

- **RDP ist unsicher, wenn direkt dem Internet ausgesetzt!**
- **Besser**: **WireGuard-VPN** oder **Reverse-Proxy mit Authentifizierung** (z. B. **Nginx + Authelia**).

Option 2: NoMachine (für Windows & Linux, besser als RDP)

- **Vorteil**: Schnell, gute Multi-Monitor-Unterstützung, verschlüsselt.
- **Einrichtung**:
  1. **NoMachine auf VMs installieren**:
     - Download (für Windows & Linux).
     - **Server installieren** (Standard-Port: **4000**).
  2. **Port in UFW freigeben**:
     ```
     sudo ufw allow 4000/tcp comment "NoMachine"
     ```
  3. **Von außen verbinden**:
     - **NoMachine-Client** installieren → `DEIN_SERVER_IP:4000` eingeben.
     - **2 Monitore aktivieren** in den NoMachine-Einstellungen.

□□ **Sicherer als RDP!** (Verschlüsselung, weniger Angriffsfläche).

Option 3: x2go (für Linux, leichtgewichtig)

- **Vorteil**: Gut für Linux, nutzt **SSH (Port 22)**.
- **Einrichtung**:
  1. **x2go-Server installieren**:
     ```
     sudo apt install x2goserver x2goserver-xsession
     ```
  2. **Port 22 ist bereits offen (SSH)**.

3. **Von außen verbinden**:
   - **x2go-Client** installieren → `DEIN_SERVER_IP` + **SSH-Benutzer** eingeben.
   - **2 Monitore** in den Sitzungseinstellungen aktivieren.

□□ Schritt 3: Firewall (UFW) & Sicherheit

Aktuell lässt deine UFW nur **SSH & Proxmox-Ports** zu. Du musst **zusätzliche Ports freigeben**:

Dienst	Port	Befehl (UFW)	Sicherheitsrisiko
RDP	3389	`sudo ufw allow 3389/tcp`	⚠ Hoch
NoMachine	4000	`sudo ufw allow 4000/tcp`	□ Mittel
x2go	22	(bereits offen)	□ Niedrig (SSH)

□□ Sicherheitsempfehlungen

1. **WireGuard-VPN einrichten** (beste Lösung für sicheren Remote-Zugriff):
   - **Tutorial**: WireGuard auf Proxmox.
   - **Vorteil**: Nur VPN-Benutzer können auf RDP/NoMachine zugreifen.
2. **Fail2Ban installieren** (gegen Brute-Force):

```
sudo apt install fail2ban
sudo systemctl enable fail2ban
```

3. **Port-Knocking** (optional, für zusätzliche Sicherheit).
4. **Reverse-Proxy mit Authentifizierung** (z. B. **Nginx + Authelia**):
   - Beispiel: `https://deine-domain.com/rdp` → leitet zu RDP weiter, aber mit **2FA**.

□□ Schritt 4: Performance-Optimierung

- **Windows-VM**:
  - **GPU-Passthrough** (falls möglich, aber bei Hetzner **nicht erlaubt**).
  - **Alternativ**: **VirtIO-GPU** (bessere Performance als Standard-VGA).

```
# In der Proxmox-VM-Konfiguration (/etc/pve/qemu-server/<VMID>.conf):
args: -device virtio-gpu-pci
```

- **Linux-VM**:
  - **SPICE + VirtIO-GPU** für bessere Grafik:

```
# In der VM-Konfig:
vga: virtio
```

---

3. Fertige Lösung: So sieht der Workflow aus

1. **Du startest die Windows-VM** (z. B. für Office/Adobe).
   - Verbindung über **NoMachine/RDP** mit **2 Monitoren**.
2. **Du startest die Linux-VM** (z. B. für Entwicklung/Docker).
   - Verbindung über **NoMachine/x2go** mit **2 Monitoren**.
3. **UFW schützt die Ports**, aber:
   - **WireGuard-VPN** (empfohlen) oder **Reverse-Proxy** macht es sicherer.
4. **Beide VMs laufen gleichzeitig**, aber nur eine wird genutzt (Ressourcen-sparend).

---

4. Alternative: Einfacherer Ansatz mit "Single-Port-Lösung"

Falls du **keine zusätzlichen Ports öffnen willst**, kannst du **alles über SSH tunneln**:

```
# Beispiel: RDP über SSH-Tunnel (lokaler Port 3390 → VM:3389)
ssh -L 3390:localhost:3389 user@DEIN_SERVER_IP
```

Dann verbindest du dich mit `localhost:3390` (nur lokal, sicher).

Vorteil: Keine zusätzlichen Firewall-Regeln nötig!

---

5. Zusammenfassung: Was du tun musst

Schritt	Aktion
VMs erstellen	Windows & Linux-VM mit SPICE/VirtIO-GPU.
Multi-Monitor einrichten	Windows: RDP/NoMachine, Linux: xrandr/NoMachine.
Remote-Zugriff konfigurieren	RDP (3389), NoMachine (4000), x2go (22).
Firewall anpassen	`sudo ufw allow 3389,4000/tcp`.
Sicherheit erhöhen	WireGuard-VPN oder Reverse-Proxy.
Performance optimieren	VirtIO-GPU & SPICE-Treiber.

---

6. Empfohlene Tools für besseres Erlebnis

Tool	Zweck	Link
NoMachine	Schnellster Remote-Desktop mit Multi-Monitor	nomachine.com
x2go	Leichtgewichtig für Linux	x2go.org
Remmina	RDP-Client für Linux	remmina.org
WireGuard	Sicheres VPN für Remote-Zugriff	wireguard.com
Authelia	2FA für Reverse-Proxy	authelia.com

Fazit: Ja, das geht!

Ja, du kannst auf deinem Hetzner-Proxmox-Server:

- **Eine Windows-VM mit 2 virtuellen Monitoren** (über RDP/NoMachine).
- **Eine Linux-VM mit 2 virtuellen Monitoren** (über NoMachine/x2go).

- **Beide über das Internet erreichbar** (mit UFW & Sicherheitseinstellungen).
- **Nur eine VM gleichzeitig nutzen** (Ressourcen-sparend).

□□ **Beste Lösung für Sicherheit: WireGuard-VPN** + **NoMachine** (statt RDP).

□□ **Beste Lösung für Einfachheit: NoMachine (Port 4000)** + **UFW-Regel**.

Fragen?

- Brauchst du Hilfe bei der **UFW-Konfiguration**?
- Soll ich eine **WireGuard-Anleitung** für Proxmox detaillierter beschreiben?
- Oder möchtest du eine **Step-by-Step für NoMachine**?

Virtuelle Workstation mit Proxmox und mehreren Screens via Wayland

Ja, **Wayland** macht die Sache etwas komplizierter, aber es gibt **Lösungen**, um **virtuelle Multi-Monitor-Setups in einer Proxmox-VM mit Wayland** (z. B. Kubuntu 24.04+) zu realisieren. Da Wayland **kein** `xrandr` unterstützt und **klassische Remote-Tools** (wie RDP oder x2go) oft noch auf X11 angewiesen sind, müssen wir **Workarounds** nutzen.

---

1. Wayland & virtuelle Monitore: Die Herausforderungen

Problem	Erklärung
Kein `xrandr`	Wayland verwaltet Monitore über **Kompositoren** (KWin, Mutter, Weston), nicht über X11-Tools.
Kein direkter Multi-Monitor-Support in SPICE/QXL	Die standardmäßige **SPICE-Grafikkarte** in Proxmox/QEMU unterstützt Wayland nur eingeschränkt.
RDP/NoMachine/x2go oft X11-basiert	Viele Remote-Tools setzen auf X11 und funktionieren nicht nativ mit Wayland.
Wayland-Sessions blockieren Remote-Zugriff	Standardmäßig erlaubt Wayland **keine Remote-Sessions** (Sicherheitsfeature).

---

2. Lösungen für Wayland in Proxmox-VMs

□□ Option 1: Wayland mit "virtuellen Monitoren" über KWin (KDE)

Funktionsweise

KDEs **KWin** (Wayland-Kompositor) erlaubt das **Hinzufügen virtueller Monitore** über **Skripte oder manuelle Konfiguration**.

Schritt-für-Schritt-Anleitung (Kubuntu 24.04+)

1. **VM erstellen (Kubuntu 24.04 mit Wayland)**
   - **Grafik**: **VirtIO-GPU** (besser als SPICE für Wayland).

     ```
     # In /etc/pve/qemu-server/<VMID>.conf:
     vga: virtio
     ```

   - **CPU/RAM**: Mind. 4 Kerne, 8 GB RAM (Wayland braucht mehr Ressourcen als X11).
2. **KWin für virtuelle Monitore konfigurieren**
   - **Manuell über `kwin_wayland`**:

     ```
     # Temporärer virtueller Monitor (rechts vom Hauptbildschirm)
     kwin_wayland --xwayland --width 3840 --height 1080
     ```

   - **Permanent über Autostart**:
     - Erstelle eine Datei `~/.config/autostart/kwin-virtual-monitor.desktop`:

       ```
       [Desktop Entry]
       Name=KWin Virtual Monitor
       Exec=kwin_wayland --xwayland --width 3840 --height 1080
       Type=Application
       ```

   - **Alternativ: `wlr-randr` (für wlroots-basierte Compositors)**

     ```
     sudo apt install wlr-randr
     wlr-randr --output Virtual1 --mode 1920x1080 --right-of DP-1
     ```

3. **Remote-Zugriff einrichten (NoMachine oder RDP über XWayland)**

- **NoMachine** funktioniert mit **XWayland** (X11-Kompatibilitätsschicht in Wayland).
  - Installiere NoMachine wie gehabt, aber starte die Session im **XWayland-Modus**.
- **RDP (xrdp) mit XWayland**:

```
sudo apt install xrdp
sudo systemctl enable --now xrdp
```

  - **Problem**: Standardmäßig startet `xrdp` eine **X11-Session**, nicht Wayland.
  - **Lösung**: **Manuell Wayland-Session erzwingen** (experimentell):

```
# /etc/xrdp/startwm.sh anpassen:
export XDG_SESSION_TYPE=wayland
export QT_QPA_PLATFORM=wayland
exec dbus-run-session -- startplasma-wayland
```

□□ Option 2: Wayland mit SPICE + `spice-vdagent` (begrenzt)

Funktionsweise

- **SPICE** (Proxmox-Standard) unterstützt **Wayland nur eingeschränkt**, aber mit **`spice-vdagent`** kann man **Bildschirmgrößen anpassen**.
- **Keine echten virtuellen Monitore**, aber **ein großer Desktop, der wie zwei Monitore aussieht**.

Einrichtung

1. **SPICE-Treiber installieren**:

```
sudo apt install spice-vdagent spice-webdavd
```

2. **VM-Konfiguration anpassen** (`/etc/pve/qemu-server/<VMID>.conf`):

```
args: -device virtio-gpu-pci -display spice,gl=on
```

3. **Manuell "Fake-Multi-Monitor" einrichten**:
   - Setze die **Auflösung auf 3840x1080** (2x 1920x1080 nebeneinander).
   - Nutze **KWin-Skripting**, um den Desktop in zwei Hälften zu teilen:

```
kwriteconfig5 --file kwinrc --group Windows --key BorderlessMaximizedWindows true
```

   - **Nachteil**: Keine echte Multi-Monitor-Unterstützung (nur ein großer Bildschirm).

□□ Option 3: Wayland mit NoMachine (beste Lösung für Remote)

Funktionsweise

- **NoMachine** unterstützt **Wayland seit Version 8** (aber noch experimentell).
- **Vorteil**: Echte **Multi-Monitor-Unterstützung** über das Netzwerk.

Einrichtung

1. **NoMachine installieren**:

```
wget https://download.nomachine.com/download/8.6/Linux/nomachine_8.6.1_1_amd64.deb
sudo dpkg -i nomachine_*.deb
```

2. **Wayland-Session erzwingen**:
   - Bearbeite `/usr/NX/etc/node.cfg`:

```
EnableWaylandSupport 1
```

3. **Neu starten & verbinden**:
   - NoMachine-Client → **Wayland-Session auswählen**.
   - **2 Monitore** in den Einstellungen aktivieren.

□□ Option 4: XWayland als Fallback (einfachste Lösung)

Funktionsweise

- **XWayland** ist eine **X11-Kompatibilitätsschicht** in Wayland.
- **Vorteil**: Du kannst **X11-Tools wie `xrandr`** nutzen, während die Session eigentlich Wayland ist.

Einrichtung

1. **XWayland aktivieren** (standardmäßig aktiv in Kubuntu).
2. **Virtuelle Monitore mit `xrandr` hinzufügen** (auch in Wayland!):

```
xrandr --output Virtual1 --mode 1920x1080 --right-of eDP1
```

3. **Remote-Tools nutzen** (RDP, NoMachine, x2go) **über XWayland**.

---

3. Remote-Zugriff für Wayland (Sicher & Performant)

Methode	Wayland-Unterstützung	Multi-Monitor	Sicherheit	Empfehlung
NoMachine (Wayland-Modus)	✅ (seit v8)	✅	🔒🔒🔒🔒	**Beste Wahl**
RDP (xrdp + XWayland)	⚠ (X11-Fallback)	✅	🔒🔒🔒	Gut für Windows-Clients
SPICE + VirtIO-GPU	⚠ (kein echter Multi-Monitor)	✅	🔒🔒🔒🔒	Nur für Single-Monitor
VNC (TigerVNC)	✅ (nur X11)	⚠	🔒🔒	Nicht ideal
WireGuard + NoMachine	✅	✅	🔒🔒🔒🔒🔒	**Sicherste Lösung**

---

4. Schritt-für-Schritt: Kubuntu 24.04 in Proxmox mit Wayland & 2 Monitoren

🖥️ Vorbereitung (Proxmox-Host)

1. **VM erstellen**:
   - **OS**: Kubuntu 24.04 ISO.
   - **Grafik**: **VirtIO-GPU** (nicht SPICE!).

```
# In /etc/pve/qemu-server/<VMID>.conf:
vga: virtio
```

- **CPU**: 4 Kerne, **RAM**: 8 GB.

2. **VirtIO-Treiber installieren** (für bessere Performance):

```
sudo apt install virtio-driver
```

□□ Einrichtung in der VM (Kubuntu 24.04)

1. **Wayland-Session starten** (standardmäßig in Kubuntu 24.04).
2. **Virtuelle Monitore hinzufügen** (über KWin):

```
# Temporärer Test:
kwin_wayland --xwayland --width 3840 --height 1080
```

   - **Permanent**:
     - Erstelle ein **KWin-Skript** (`~/.config/kwinrc`):

```
[ScreenEdges]
BorderlessMaximizedWindows=true
```

     - **Autostart-Skript** für virtuelle Monitore:

```
echo 'kwin_wayland --xwayland --width 3840 --height 1080' >> ~/.config/autostart/kwin-virtual.sh
chmod +x ~/.config/autostart/kwin-virtual.sh
```

3. **NoMachine für Remote-Zugriff installieren**:

```
wget https://download.nomachine.com/download/8.6/Linux/nomachine_8.6.1_1_amd64.deb
sudo dpkg -i nomachine_*.deb
sudo systemctl enable --now nxserver
```

   - **Konfiguration für Wayland**:

```
sudo sed -i 's/EnableWaylandSupport 0/EnableWaylandSupport 1/' /usr/NX/etc/node.cfg
sudo systemctl restart nxserver
```

4. **Firewall (UFW) anpassen**:

```
sudo ufw allow 4000/tcp comment "NoMachine"
```

□□ Verbindung von außen (Client)

1. **NoMachine-Client installieren** (Download).
2. **Verbindung herstellen**:
   - **Server**: `DEIN_SERVER_IP:4000`
   - **Session-Typ**: **"Wayland"** auswählen.
   - **2 Monitore aktivieren** in den Einstellungen.

---

5. Sicherheitsoptimierung (wichtig für Internet-Zugriff!)

□□ Option A: WireGuard-VPN (empfohlen)

- **Vorteil**: Keine offenen Ports im Internet, nur VPN-Zugriff.
- **Einrichtung**:

```
# Auf dem Proxmox-Host:
sudo apt install wireguard
wg genkey | tee privatekey | wg pubkey > publickey
```

  - **Konfiguration**: `/etc/wireguard/wg0.conf`

```
[Interface]
PrivateKey = <SERVER_PRIVATE_KEY>
Address = 10.0.0.1/24
ListenPort = 51820
PostUp = iptables -A FORWARD -i %i -j ACCEPT; iptables -t nat -A POSTROUTING -o vmbr0 -j MASQUERADE
PostDown = iptables -D FORWARD -i %i -j ACCEPT; iptables -t nat -D POSTROUTING -o vmbr0 -j MASQUERADE
```

```
[Peer]
PublicKey = <CLIENT_PUBLIC_KEY>
AllowedIPs = 10.0.0.2/32
```

- **UFW-Regel**:

```
sudo ufw allow 51820/udp comment "WireGuard"
```

- **Client-Konfiguration** (z. B. auf deinem Laptop):

```
[Interface]
PrivateKey = <CLIENT_PRIVATE_KEY>
Address = 10.0.0.2/24

[Peer]
PublicKey = <SERVER_PUBLIC_KEY>
Endpoint = DEIN_SERVER_IP:51820
AllowedIPs = 10.0.0.0/24
```

- **Verbindung herstellen**:

```
sudo wg-quick up wg0
```

- **Dann NoMachine/RDP über VPN nutzen** (`10.0.0.1:4000`).

□□ Option B: Reverse-Proxy mit Authelia (für direkte Exposition)

- **Vorteil**: Kein VPN nötig, aber **2FA & HTTPS**.
- **Einrichtung (Nginx + Authelia)**:

```
sudo apt install nginx certbot python3-certbot-nginx
sudo snap install --edge authelia
```

- **Nginx-Konfiguration** (`/etc/nginx/sites-available/nomachine`):

```
server {
    listen 443 ssl;
    server_name nomachine.deine-domain.com;
```

```
    ssl_certificate /etc/letsencrypt/live/nomachine.deine-domain.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/nomachine.deine-domain.com/privkey.pem;

    location / {
        proxy_pass http://localhost:4000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
    }
}
```

- **Authelia für 2FA**:

```
# /etc/authelia/configuration.yml
authentication_backend:
  file:
    path: /etc/authelia/users_database.yml
access_control:
  default_policy: deny
  rules:
    - domain: "nomachine.deine-domain.com"
      policy: two_factor
```

- **Zertifikat mit Let’s Encrypt**:

```
sudo certbot --nginx -d nomachine.deine-domain.com
```

---

6. Performance-Tipps für Wayland in Proxmox

Problem	Lösung
Laggy Grafik	Nutze **VirtIO-GPU** statt SPICE.
Keine Hardware-Beschleunigung	Aktiviere **OpenGL in NoMachine**: `sudo nxserver --enable-opengl`
Wayland-Session crasht	Starte mit **XWayland-Fallback**: `export XDG_SESSION_TYPE=x11`

Problem	Lösung
Keine virtuellen Monitore	Nutze **KWin-Skripting** oder `wlr-randr`.
Hohe CPU-Last	Reduziere die **Auflösung** (z. B. 1920x1080 statt 4K).

---

7. Fazit: Wayland in Proxmox mit Multi-Monitor

Methode	Funktionsweise	Vorteile	Nachteile
NoMachine (Wayland-Modus)	Echte Wayland-Session mit Multi-Monitor	✅ Beste Performance, ✅ Echte Wayland-Unterstützung	⚠ Noch experimentell
XWayland + xrandr	Wayland mit X11-Kompatibilität	✅ Einfach, ✅ Stabil	❌ Kein "echtes" Wayland
KWin-Skripting	Virtuelle Monitore über KWin	✅ Nativ in Wayland	⚠ Manuelle Konfiguration nötig
SPICE + VirtIO-GPU	Emulierte Grafik	✅ Einfach	❌ Kein echter Multi-Monitor

🚀 Empfohlene Lösung für dein Setup

1. **Kubuntu 24.04-VM mit VirtIO-GPU**.
2. **NoMachine in Wayland-Modus** (Port 4000).
3. **WireGuard-VPN** für sicheren Zugriff.
4. **KWin-Skript für virtuelle Monitore** (falls NoMachine nicht reicht).

🔄 Alternative: Bei X11 bleiben (bis Wayland stabiler ist)

Falls Wayland zu viele Probleme macht:

```
# X11 erzwingen (in Kubuntu 24.04)
echo "exec startplasma-x11" > ~/.xsession
```

- **Vorteil**: `xrandr` funktioniert, NoMachine/RDP/x2go laufen stabil.

- **Nachteil**: Kein Wayland (aber für Remote-Setups oft besser).

---

8. Zusammenfassung: Was du tun musst

Schritt	Aktion
✅ **VM erstellen**	Kubuntu 24.04 mit **VirtIO-GPU**.
✅ **Wayland aktivieren**	Standard in Kubuntu 24.04.
✅ **Virtuelle Monitore einrichten**	`kwin_wayland` oder `xrandr` (XWayland).
✅ **NoMachine installieren**	Mit `EnableWaylandSupport 1`.
✅ **Firewall anpassen**	`sudo ufw allow 4000/tcp`.
✅ **Sicherheit erhöhen**	**WireGuard-VPN** oder **Reverse-Proxy**.
✅ **Testen**	Verbindung mit NoMachine-Client (Wayland-Modus).

Fragen?

- Brauchst du eine **detaillierte WireGuard-Anleitung** für Proxmox?
- Soll ich die **NoMachine-Wayland-Konfiguration** genauer erklären?
- Oder möchtest du eine **Alternative mit X11** (stabiler, aber ohne Wayland)?

SQL Server Logs

Kurzantwort

- Vollständige Logs findest du als Dateien im MSSQL-Logverzeichnis:
  - SQL Server: ERRORLOG, ERRORLOG.1, …
  - SQL Server Agent: SQLAGENT.OUT, SQLAGENT.1, …
- Die „Job-Historie“ im Agent-Fenster wird auf ca. 4.000 Zeichen pro Schritt gekürzt. Für die komplette Ausgabe leite die Steps in eine Datei oder Tabelle um (siehe unten).

Wo liegen die Log-Dateien?

- Windows (Standardpfad, kann abweichen):
  C:\Program Files\Microsoft SQL Server\MSSQLXX.<Instanz>\MSSQL\Log\
  Dateien: ERRORLOG, ERRORLOG.1 … und SQLAGENT.OUT, SQLAGENT.1 …
- Linux:
  /var/opt/mssql/log/errorlog
  /var/opt/mssql/log/sqlagent.out

Per T-SQL den aktuellen Pfad ermitteln:

```
-- voller Pfad zur aktuellen ERRORLOG-Datei
SELECT SERVERPROPERTY('ErrorLogFileName') AS ErrorLogFile;
```

In SSMS:

- Management > SQL Server-Protokolle (Error Logs)
- SQL Server-Agent > Fehlerprotokolle (Agent Error Logs)

Per T-SQL lesen:

```
-- SQL Server Errorlog (aktuelles Log)
EXEC xp_readerrorlog 0, 1;  -- 1 = SQL Server


-- SQL Agent Log (aktuelles Log)
EXEC xp_readerrorlog 0, 2;  -- 2 = SQL Agent


-- optional Filter & Zeitraum
```

```
EXEC xp_readerrorlog
  0, 2, 'Failed', NULL, '2025-11-01', '2025-11-10', 'DESC';
```

Tipp: Mit

```
EXEC sp_cycle_errorlog;
```

kannst du das Errorlog rotieren (z. B. täglich per Job).

Warum ist die Ausgabe im Agent-Fenster abgeschnitten?

Die Job-Historie (msdb..sysjobhistory) speichert die Step-Meldungen nur bis ca. 4.000 Zeichen. SSMS zeigt daher oft nur den Anfang. Das ist der Grund, warum du im Agent-Fenster keine vollständige Ausgabe siehst.

So bekommst du die vollständige Job-Ausgabe

1. In eine Datei loggen (empfohlen)

- Job > Schritt > Erweitert (Advanced) > „Ausgabe in Datei“
- Optional: „Ausgabe an vorhandene Datei anhängen“ aktivieren
- Du kannst Tokens für dynamische Namen verwenden:

  ```
  E:\SQLAgentLogs\$(JOBNAME)_Step$(STEPID)_$(DATE)_$(TIME).txt
  ```

Per T-SQL setzen:

```
EXEC msdb.dbo.sp_update_jobstep
  @job_name = N'Mein Job',
  @step_id = 1,
  @output_file_name = N'E:\SQLAgentLogs\$(JOBNAME)_Step$(STEPID)_$(DATE)_$(TIME).txt',
  @append_output = 1;
```

2. In Tabelle loggen (für lange Ausgaben)

- Job > Schritt > Erweitert > „In Tabelle protokollieren“ aktivieren
- Optional „An vorhandene Protokolle anhängen“ aktivieren
- Danach kannst du die Logs in msdb abfragen, z. B.:

```
-- Schritt-Logs (wenn "In Tabelle protokollieren" aktiviert war)
SELECT TOP (100)
    j.name,
    l.step_id,
    l.log_date,
    l.log -- i. d. R. NVARCHAR(MAX)
FROM msdb.dbo.sysjobstepslogs AS l
JOIN msdb.dbo.sysjobs AS j ON j.job_id = l.job_id
ORDER BY l.log_date DESC;
```

3. Über sqlcmd/powershell ausführen und Output umleiten

- SQLCMD: `sqlcmd -S <Server> -E -i script.sql -o "E:\SQLAgentLogs\script_output.txt"`
- Den Agent-Step als „Betriebssystem (CmdExec)“-Schritt konfigurieren und so die komplette Ausgabe erfassen.

4. Job-Historie-Einstellungen anpassen

- SQL Server-Agent > Eigenschaften > Verlauf: Anzahl Einträge erhöhen, aber beachte: das hebt die 4.000-Zeichen-Grenze pro Step nicht auf.

5. Während der Laufzeit „live“ Ausgaben erzwingen

- In T-SQL Steps hilft z. B.:

```
RAISERROR('Zwischenstand ...', 0, 1) WITH NOWAIT;
```

So siehst du Fortschritt früher; für die Vollständigkeit nutze trotzdem eine Datei/Tabelle.

Nützliche Abfragen rund um Agent-Jobs

Letzte Jobläufe mit (ggf. abgeschnittenen) Meldungen:

```
SELECT TOP (50)
    j.name,
    h.step_id,
    h.run_date, h.run_time,
```

```
    h.run_duration,
    h.run_status,   -- 0=Fehler, 1=Erfolg, 2=Abgebrochen, 3=Abgebrochen vom Benutzer, 4=Retry
    h.message
FROM msdb.dbo.sysjobhistory AS h
JOIN msdb.dbo.sysjobs AS j ON j.job_id = h.job_id
WHERE h.step_id > 0
ORDER BY h.instance_id DESC;
```

Zusammenfassung

- Die vollständigen Server-/Agent-Logs sind als Dateien im MSSQL-Logordner (ERRORLOG, SQLAGENT.OUT).
- Die Anzeige im Agent-Fenster ist der Natur nach gekürzt.
- Für komplette Jobausgaben: „Ausgabe in Datei“ oder „In Tabelle protokollieren“ am Step aktivieren.
- Bei Bedarf per xp_readerrorlog oder direkt in SSMS die Error-/Agent-Logs ansehen.

Wenn du magst, sag mir deine SQL Server-Version und ob du Windows oder Linux nutzt, dann gebe ich dir eine exakt passende Schritt-für-Schritt-Anleitung.

BtrFS Installation Kubuntu 25.10

Dann machen wir es so automatisch wie unter Kubuntu eben möglich.
Du klickst im Prinzip *nur noch den Installer durch*, der Rest sind zwei Skripte.

Ich schlage vor:

1. **Prepare-Script** im Live-System → macht GPT-Layout + Btrfs-Partition
2. Kubuntu ganz normal installieren (1–2 Klicks bei der Partitionierung)
3. **Post-Install-Script** im Live-System → richtet Subvolumes + `fstab` vollautomatisch ein

Beide Skripte sind **`sgdisk`-frei** (nur `parted` + `btrfs`).

1️⃣ Prepare-Script (vor der Installation)

Zweck: Platte leeren, GPT anlegen, EFI + Btrfs-Partition erstellen.
Noch keine Subvolumes, die kommen nach der Installation.

Passe oben **`DISK`** an (z. B. `/dev/nvme0n1` oder `/dev/sda`).

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

### === KONFIGURATION ANPASSEN ===
DISK="/dev/sdX"             # z.B. /dev/nvme0n1 oder /dev/sda
EFI_SIZE_MIB=512            # Größe der EFI-Partition in MiB
BTRFS_LABEL="KUBUNTU_ROOT"   # Label für das Btrfs-Dateisystem

echo ">>> WARNUNG: $DISK wird KOMPLETT GELÖSCHT!"
read -p "Weiter mit ENTER, Abbrechen mit STRG+C..."

### 1. Alle alten Signaturen entfernen
sudo wipefs -a "$DISK"

### 2. GPT + Partitionen anlegen (nur parted, kein sgdisk)
```

```
sudo parted -s "$DISK" \
  mklabel gpt \
  unit MiB \
  mkpart EFI fat32 1 $((1 + EFI_SIZE_MIB)) \
  set 1 esp on \
  mkpart ROOT btrfs $((1 + EFI_SIZE_MIB)) 100%

EFI_PART="${DISK}1"
ROOT_PART="${DISK}2"

echo ">>> Neue Partitionen:"
lsblk "$DISK"

### 3. Dateisysteme anlegen
echo ">>> Formatiere EFI-Partition ($EFI_PART) als FAT32..."
sudo mkfs.vfat -F32 -n EFI "$EFI_PART"

echo ">>> Formatiere ROOT-Partition ($ROOT_PART) als Btrfs..."
sudo mkfs.btrfs -f -L "$BTRFS_LABEL" "$ROOT_PART"

echo
echo "=== SCHRITT IM INSTALLER (KUBUNTU 25.10) ==="
echo "1) 'Etwas Anderes' / manuelle Partitionierung wählen."
echo "2) $EFI_PART:"
echo "     - Typ: EFI System Partition"
echo "     - Dateisystem: FAT32"
echo "     - Einhängepunkt: /boot/efi"
echo "     - NICHT neu formatieren."
echo "3) $ROOT_PART:"
echo "     - Dateisystem: Btrfs"
echo "     - Einhängepunkt: /"
echo "     - NICHT neu formatieren."
echo
echo "Dann Installation ganz normal durchlaufen lassen."
```

Benutzung:

1. Kubuntu-Live-ISO booten
2. Terminal öffnen, Script z. B. als `prepare-btrfs.sh` speichern, ausführbar machen:

```
chmod +x prepare-btrfs.sh
./prepare-btrfs.sh
```

3. Danach Kubuntu-Installer starten und die Hinweise aus der Ausgabe befolgen.

---

2 Post-Install-Script (nach der Installation)

Zweck:

- vorhandene Kubuntu-Installation auf Btrfs-Subvolumes umziehen
- `@`, `@home`, `@log`, `@cache`, `@snapshots` anlegen
- `/etc/fstab` automatisch anpassen

Das machst du **am bequemsten wieder vom Live-System aus**, nachdem Kubuntu fertig installiert wurde.

Oben **`ROOT_PART`** auf deine Btrfs-Partition setzen (z. B. `/dev/nvme0n1p2`).

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

### === ANPASSEN: Btrfs-Root-Partition der Kubuntu-Installation ===
ROOT_PART="/dev/nvme0n1p2"

TOP_MNT="/mnt/btrfs-top"
NEWROOT_MNT="/mnt/newroot"

echo ">>> Verwende ROOT_PART=$ROOT_PART"
read -p "Weiter mit ENTER, Abbrechen mit STRG+C..."

sudo mkdir -p "$TOP_MNT" "$NEWROOT_MNT"

### 1. Top-Level (subvolid=5) mounten
echo ">>> Mount Top-Level Btrfs (subvolid=5)..."
sudo mount -o subvolid=5 "$ROOT_PART" "$TOP_MNT"

echo ">>> Inhalt von $TOP_MNT:"
ls "$TOP_MNT"
```

```
### 2. Subvolumes anlegen
echo ">>> Erstelle Subvolumes @, @home, @log, @cache, @snapshots..."
sudo btrfs subvolume create "$TOP_MNT/@"
sudo btrfs subvolume create "$TOP_MNT/@home"
sudo btrfs subvolume create "$TOP_MNT/@log"
sudo btrfs subvolume create "$TOP_MNT/@cache"
sudo btrfs subvolume create "$TOP_MNT/@snapshots"

### 3. Daten in die Subvolumes kopieren
echo ">>> Kopiere Root-Dateisystem nach @ (ohne /home, /var/log, /var/cache, /proc, /sys, /dev, /run, /tmp, /mnt, /media)..."
sudo rsync -aHAX \
  --exclude="/home/*" \
  --exclude="/var/log/*" \
  --exclude="/var/cache/*" \
  --exclude="/proc/*" \
  --exclude="/sys/*" \
  --exclude="/dev/*" \
  --exclude="/run/*" \
  --exclude="/tmp/*" \
  --exclude="/mnt/*" \
  --exclude="/media/*" \
  "$TOP_MNT"/ "$TOP_MNT/@"/

echo ">>> Kopiere /home nach @home..."
sudo rsync -aHAX "$TOP_MNT/home"/ "$TOP_MNT/@home"/ || true

echo ">>> Kopiere /var/log nach @log..."
sudo rsync -aHAX "$TOP_MNT/var/log"/ "$TOP_MNT/@log"/ || true

echo ">>> Kopiere /var/cache nach @cache..."
sudo rsync -aHAX "$TOP_MNT/var/cache"/ "$TOP_MNT/@cache"/ || true

### 4. Alte Verzeichnisse sichern (nicht löschen, nur umbenennen)
echo ">>> Sichere alte Verzeichnisse (home, var/log, var/cache)..."
[ -d "$TOP_MNT/home" ]      && sudo mv "$TOP_MNT/home"      "$TOP_MNT/home.oldroot"      || true
[ -d "$TOP_MNT/var/log" ]   && sudo mv "$TOP_MNT/var/log"   "$TOP_MNT/var.log.oldroot"   || true
[ -d "$TOP_MNT/var/cache" ] && sudo mv "$TOP_MNT/var/cache" "$TOP_MNT/var.cache.oldroot" || true
```

```
### 5. Neue Root-Sicht (subvol=@) mounten
echo ">>> Mount subvol=@ nach $NEWROOT_MNT..."
sudo mount -o subvol=@ "$ROOT_PART" "$NEWROOT_MNT"

### 6. Mountpoints im neuen Root anlegen
echo ">>> Erstelle Mountpoints im neuen Root..."
sudo mkdir -p "$NEWROOT_MNT/home"
sudo mkdir -p "$NEWROOT_MNT/var/log"
sudo mkdir -p "$NEWROOT_MNT/var/cache"
sudo mkdir -p "$NEWROOT_MNT/.snapshots"

### 7. /etc/fstab anpassen
UUID=$(blkid -s UUID -o value "$ROOT_PART")
echo ">>> Gefundene UUID für $ROOT_PART: $UUID"

FSTAB="$NEWROOT_MNT/etc/fstab"
echo ">>> Sicherung von fstab nach fstab.bak-pre-btrfs..."
sudo cp "$FSTAB" "$FSTAB.bak-pre-btrfs"

echo ">>> Passe Root-Eintrag in fstab an (subvol=@)..."
sudo sed -i "s#^\(UUID=$UUID[[:space:]]\+\)/[[:space:]]\+btrfs.*#UUID=$UUID / btrfs
rw,relatime,ssd,space_cache=v2,subvol=@ 0 0#" "$FSTAB"

echo ">>> Füge Subvolume-Mounts für /home, /var/log, /var/cache, /.snapshots hinzu..."
cat <<EOF | sudo tee -a "$FSTAB" >/dev/null

# Btrfs Subvolumes
UUID=$UUID /home       btrfs rw,relatime,ssd,space_cache=v2,subvol=@home      0 0
UUID=$UUID /var/log    btrfs rw,relatime,ssd,space_cache=v2,subvol=@log       0 0
UUID=$UUID /var/cache  btrfs rw,relatime,ssd,space_cache=v2,subvol=@cache     0 0
UUID=$UUID /.snapshots btrfs rw,relatime,ssd,space_cache=v2,subvol=@snapshots 0 0
EOF

echo ">>> fstab nach Anpassung:"
cat "$FSTAB"

### 8. Aufräumen
echo ">>> Unmount..."
sudo umount "$NEWROOT_MNT"
sudo umount "$TOP_MNT"
```

```
echo
echo ">>> FERTIG. Starte jetzt dein installiertes Kubuntu neu."
echo "    Falls etwas schief geht, kannst du im Notfall mit dem Live-System booten"
echo "    und /etc/fstab aus fstab.bak-pre-btrfs wiederherstellen."
```

Benutzung:

1. Nach der Installation wieder mit dem Kubuntu-Live-Stick booten
2. Terminal öffnen, Script z. B. als `kubuntu-btrfs-post.sh` speichern
3. `ROOT_PART` oben anpassen (z. B. `/dev/nvme0n1p2`)
4. Ausführbar machen und laufen lassen:

   ```
   chmod +x kubuntu-btrfs-post.sh
   ./kubuntu-btrfs-post.sh
   ```

5. Reboot von der Platte → Kubuntu sollte jetzt sauber mit:
   - `/` → `@`
   - `/home` → `@home`
   - `/var/log` → `@log`
   - `/var/cache` → `@cache`
   - `/.snapshots` → `@snapshots`

   laufen.

Wenn du willst, können wir das jetzt noch **konkret auf deine Hardware anpassen**:

- Wie heißt deine Zielplatte? (`lsblk`-Ausgabe)
- Planst du Swap (Partition oder Swapfile auf Btrfs)?

Dann schreibe ich dir die beiden Skripte exakt passend (inkl. Swap-Handling) und du musst wirklich nur noch 2× Script starten + 1× Installer klicken.

Linux und Microsoft AD

Anleitungen rund um Active Directory

AD Beitritt Ubuntu LTS

Hier ist eine **komplette, kopierfertige Anleitung für Ubuntu (22.04 LTS / 24.04 LTS)** mit **SSSD + Realmd** für **AD-Integration inkl. Offline-Login** – **ohne Winbind**, da SSSD zuverlässiger für Offline-Szenarien ist.

□□ Ubuntu AD-Integration mit Offline-Login (SSSD + Realmd)

Voraussetzungen:

- Ubuntu **22.04 LTS oder 24.04 LTS** (frisch installiert)
- Domänenname: `BEISPIEL.DOMAENE.LAN` (ersetzen!)
- AD-Admin-Benutzer: `Administrator` (oder anderer berechtigter Benutzer)
- Netzwerkverbindung zum **Domänencontroller (DC)**

1□ Pakete installieren & System vorbereiten

```
# System aktualisieren
sudo apt update && sudo apt upgrade -y

# Benötigte Pakete installieren
sudo apt install -y realmd sssd sssd-tools adcli krb5-user packagekit oddjob oddjob-mkhomedir pam_krb5 libnss-sss libpam-sss ntp

# Hostname setzen (muss im AD eindeutig sein)
sudo hostnamectl set-hostname ubuntu-client1.beispiel.domäne.lan
```

2□ Kerberos konfigurieren

```
sudo vi /etc/krb5.conf
```

Inhalt ersetzen durch:

```
[libdefaults]
    default_realm = BEISPIEL.DOMAENE.LAN
    dns_lookup_realm = true
    dns_lookup_kdc = true
    ticket_lifetime = 24h
    renew_lifetime = 7d
    forwardable = true
    rdns = false

[realms]
    BEISPIEL.DOMAENE.LAN = {
        kdc = dc1.beispiel.domäne.lan
        admin_server = dc1.beispiel.domäne.lan
    }

[domain_realm]
    .beispiel.domäne.lan = BEISPIEL.DOMAENE.LAN
    beispiel.domäne.lan = BEISPIEL.DOMAENE.LAN
```

---

3 DNS prüfen & anpassen

```
# Testen, ob der DC erreichbar ist
ping dc1.beispiel.domäne.lan
nslookup dc1.beispiel.domäne.lan

# Falls DNS nicht funktioniert, manuell anpassen:
sudo vi /etc/resolv.conf
```

Inhalt ersetzen durch:

```
nameserver 192.168.1.10  # IP des DC
search beispiel.domäne.lan
```

→ **Wichtig:** Falls `systemd-resolved` aktiv ist:

```
sudo systemctl disable --now systemd-resolved
sudo systemctl enable --now NetworkManager
```

---

4 Domäne mit `realmd` beitreten

```
# Domäne entdecken (Test)
sudo realm discover BEISPIEL.DOMAENE.LAN

# Domäne beitreten (AD-Admin-Passwort eingeben)
sudo realm join -U Administrator BEISPIEL.DOMAENE.LAN --verbose
```

→ **Erwartete Ausgabe:** `Successfully enrolled machine in realm`

---

5 SSSD für Offline-Login konfigurieren

```
sudo vi /etc/sssd/sssd.conf
```

Folgende Konfiguration einfügen:

```
[sssd]
domains = BEISPIEL.DOMAENE.LAN
config_file_version = 2
services = nss, pam, sudo, ssh

[domain/BEISPIEL.DOMAENE.LAN]
ad_domain = BEISPIEL.DOMAENE.LAN
krb5_realm = BEISPIEL.DOMAENE.LAN
realmd_tags = manages-system joined-with-adcli
cache_credentials = true          # Offline-Login aktivieren!
krb5_store_password_if_offline = true
default_shell = /bin/bash
ldap_id_mapping = true
use_fully_qualified_names = false  # Kurze Benutzernamen (z. B. "user" statt "user@domäne.lan")
fallback_homedir = /home/%u
access_provider = ad
```

```
entry_cache_timeout = 1209600     # Cache-Gültigkeit: 14 Tage
account_cache_expiration = 14     # Account-Cache: 14 Tage
```

Berechtigungen setzen:

```
sudo chmod 600 /etc/sssd/sssd.conf
```

---

6 PAM für Home-Verzeichnisse & Offline-Login anpassen

```
# Automatische Home-Verzeichnisse aktivieren
sudo pam-auth-update --enable mkhomedir

# Manuell prüfen (falls nötig)
sudo vi /etc/pam.d/common-session
```

Folgende Zeile hinzufügen (falls nicht vorhanden):

```
session required pam_mkhomedir.so skel=/etc/skel umask=0022
```

---

7 Dienste neu starten

```
sudo systemctl restart sssd
sudo systemctl enable --now oddjobd
```

---

8 Test: Online-Anmeldung (Cache füllen)

```
# Testbenutzer anmelden (ersetze "testuser" mit einem AD-Benutzer)
su - testuser
exit

# Kerberos-Ticket prüfen
```

```
klist
```

→ **Erwartet:** Ein gültiges Ticket für `testuser@BEISPIEL.DOMAENE.LAN`.

---

9 Test: Offline-Login

```
# Netzwerk trennen (z. B. Kabel ziehen oder WLAN deaktivieren)
sudo systemctl stop NetworkManager

# Offline-Login testen
su - testuser  # Sollte funktionieren!
id testuser    # Sollte Benutzerdaten anzeigen
klist -l       # Sollte gecachte Tickets zeigen
```

Fehlersuche (falls nötig)

Problem	Befehl zur Diagnose
Anmeldung fehlgeschlagen	`journalctl -u sssd -f`
Keine Benutzer auflösbar	`getent passwd testuser`
Kerberos-Fehler	`kinit testuser` (manueller Test)
DNS-Probleme	`dig dc1.beispiel.domäne.lan`
SSSD-Cache löschen	`sudo systemctl stop sssd; sudo rm -f /var/lib/sss/db/cache_*.ldb; sudo systemctl start sssd`

Fertig!

- Ubuntu ist jetzt in der **AD-Domäne** und unterstützt **Offline-Login**.
- **Wichtig:** Jeder Benutzer muss sich **mindestens einmal online anmelden**, bevor er offline arbeiten kann.

Optional: Automatische Ticket-Erneuerung

Falls Kerberos-Tickets zu schnell ablaufen:

```
sudo vi /etc/krb5.conf
```

Folgende Zeilen anpassen:

```
[libdefaults]
    ticket_lifetime = 24h
    renew_lifetime = 7d
    forwardable = true
```

Dann **SSSD neu starten**:

```
sudo systemctl restart sssd
```

□□ Warum SSSD statt Winbind?

Kriterium	SSSD	Winbind
Offline-Login	□ Zuverlässig (cached credentials)	□ Oft problematisch
Kerberos-Integration	□ Voll unterstützt	⚠ Teilweise manuelle Konfiguration nötig
Performance	□ Schneller (lokaler Cache)	⚠ Langsamer (mehr DC-Abfragen)
Gruppenrichtlinien (GPO)	□ Nicht unterstützt	□ Teilweise möglich
Einfachheit	□ `realmd` macht viel automatisch	⚠ Manuelle Konfiguration nötig

---

□□ Fazit

- **SSSD + Realmd** ist die **beste Wahl für Ubuntu mit Offline-Login**.
- **Winbind** ist nur nötig, wenn **GPOs** oder **Samba-Freigaben** im Vordergrund stehen.
- **Smartcards?** → Dann zusätzlich `pam_pkcs11` installieren und konfigurieren.

Falls du **Samba-Freigaben** oder **Gruppenrichtlinien** brauchst, lass es mich wissen – dann ergänze ich die Anleitung! □□

Zertifikat mit AD erzeugen

Wenn du eine **klassische AD-Zertifizierungsstelle** hast und **nur mit Windows-Bordmitteln** ein Zertifikat anfordern willst, geht das sehr stabil mit **certreq + certutil + openssl**. Damit bekommst du auch die drei Dateien (`.crt`, `.key`, `.chain.pem`), die Nginx Proxy Manager erwartet.

□□ Schritt-für-Schritt mit Windows Bordmitteln

1. Request-Datei erstellen

Erstelle eine INF-Datei, z. B. `request.inf`:

```
[Version]
Signature="$Windows NT$"

[NewRequest]
Subject = "CN=app.contoso.com"
KeySpec = 1
KeyLength = 2048
Exportable = TRUE
MachineKeySet = TRUE
SMIME = FALSE
PrivateKeyArchive = FALSE
UserProtected = FALSE
UseExistingKeySet = FALSE
ProviderName = "Microsoft RSA SChannel Cryptographic Provider"
ProviderType = 12
RequestType = PKCS10
KeyUsage = 0xa0
```

```
[Extensions]
2.5.29.17 = "{text}"
_continue_ = "dns=app.contoso.com&"
_continue_ = "dns=www.app.contoso.com"

[EnhancedKeyUsageExtension]
OID=1.3.6.1.5.5.7.3.1 ; Server Authentication
```

2. CSR erzeugen

```
certreq -new request.inf request.csr
```

3. Zertifikat bei der AD-CA beantragen

Falls dein Client Mitglied der Domäne ist und du Berechtigungen hast:

```
certreq -submit -config "CA-SERVER\CA-NAME" request.csr cert.cer
```

- `CA-SERVER\CA-NAME` ist deine AD-CA (findest du in der CA-Konsole).
- Ergebnis: `cert.cer` (dein Serverzertifikat).
- Private Keys liegen im Windows-Keystore.

4. In PFX exportieren

Zuerst Zertifikat in den Store importieren:

```
certreq -accept cert.cer
```

Dann per `certutil` mit privatem Schlüssel exportieren:

```
certutil -exportPFX -p SuperGeheimesPasswort My "CN=app.contoso.com" app.pfx
```

Jetzt hast du alles in `app.pfx`.

5. Aufsplitten in 3 Dateien

Windows kann kein `.pem` direkt erzeugen → hier kommt **OpenSSL** ins Spiel (kannst du von slproweb.com installieren).

```
# privater Schlüssel extrahieren
openssl pkcs12 -in app.pfx -nocerts -out privkey.pem -nodes -password pass:SuperGeheimesPasswort

# Zertifikat extrahieren
openssl pkcs12 -in app.pfx -clcerts -nokeys -out cert.pem -password pass:SuperGeheimesPasswort

# Kette (Root + Intermediate) extrahieren
openssl pkcs12 -in app.pfx -cacerts -nokeys -out chain.pem -password pass:SuperGeheimesPasswort
```

Damit hast du:

- `privkey.pem` → der private Schlüssel
- `cert.pem` → dein Serverzertifikat
- `chain.pem` → die Zwischen-/Root-Zertifikate

Diese drei Dateien kannst du dann im **Nginx Proxy Manager** unter „Custom SSL Certificate“ hochladen.

Zusammenfassung

- Mit **`certreq`** → CSR erzeugen und Zertifikat von AD CS holen
- Mit **`certutil`** → in PFX exportieren
- Mit **`openssl`** → in die drei Dateien (`privkey.pem`, `cert.pem`, `chain.pem`) zerlegen

Das ist der „saubere“ Microsoft-Weg, kein Gefrickel.

Soll ich dir ein komplettes **PowerShell-Skript** bauen, das all diese Schritte (CSR → AD-CA → PFX → PEMs) automatisch abwickelt? Dann müsstest du nur einmal CN/Domains eintragen und hättest am Ende direkt die 3 Dateien im gewünschten Ordner.

Linux System zu AD hinzufügen

Allgemeines

Es gibt zwar eine relativ einfache Methode einer AD Domäne beizutreten. Dies ist hier sehr gut beschrieben. Wer jedoch die volle Funktionalität von AD nutzen möchte, kommt bei der Methode sehr schnell an seine Grenzen. Unter Open Suse 15.x / Tubleweed ist das kein Problem. Yast auf und Domäne beitreten wählen. Etwas warten, die Admin Benutzerdaten eingeben und schon ist die Sache erledigt. Bei Ubuntu ist die Sache nicht ganz so simpel. Dieser Artikel bezieht sich auf Ubuntu 20.04 und 21.04. Sollte so aber auch in späteren Versionen funktionieren.

Pakete installieren

```
sudo apt-get install -y krb5-user libpam-krb5 winbind  samba  smbclient libnss-winbind libpam-winbind
```

Während der Installation wird nach dem Realm für Kerberos 5 gefragt. Dieser ist der Domänen Name von Active Directory. Wenn der Rechner als **rechner01.ad.einedomain.org** lautet ist der Realm **einedomain.org**.

Kerberos einrichten

In der Datei **/etc/krb5.conf** mit folgendem Inhalt versehen, wobei **ad.eigenedomain.org** wieder der bei der Installation eingegebene Name ist. Bitte auch auf die Großschreibung achten, da MIT Kerberos da ein kleinwenig heikel ist:

```
[logging]
    default = FILE:/var/log/krb5.log
```

```
[libdefaults]
    ticket_lifetime = 24000
    clock_skew = 300
    default_realm = AD.EIGENEDOMAIN.ORG

[realms]
    EXAMPLE.COM = {
        kdc = pdc.eigenedomain.org:88
        admin_server = pdc.eigenedomain.org:464
        default_domain = AD.EIGENEDOMAIN.ORG
    }

[domain_realm]
    .ad.eigenedomain.org = AD.EIGENEDOMAIN.ORG
    ad.eigenedomain.org = AD.EIGENEDOMAIN.ORG
```

Sobald die Datei gespeichert ist kann dies getestet werden. Auch hier wieder auf die groß geschriebene Domain achten:

```
kinit administrator@AD.EIGENEDOMAIN.ORG
```

Sollte jetzt keine Ausgabe kommen, war die Operation erfolgreich. Wenn jedoch eine Meldung wie die kommt:

```
kinit: KDC-Antwort entsprach nicht den Erwartungen bei Anfängliche Anmeldedaten werden geholt.
```

ist etwas schief gelaufen. In den meisten Fällen ist der Fehler dann entweder ein Fehler in den Rechnernamen oder dass der Realm entweder bei kinit oder in der Konfiguration nicht groß geschrieben waren.

Samba konfigurieren

Als nächstes bringen wird die bestehende Beispielkonfiguration in Sicherheit (diese behalte ich ganz gerne, weil in den Dateien sehr viel Dokumentiert ist):

```
sudo mv /etc/samba/smb.conf /etc/samba/smb.conf.sample
```

Nun legen wird die Datei **/etc/samba/smb.conf** neu an:

```
[global]
  security = ads
  realm = AD.EIGENEDOMAIN.ORG
  password server = IPADRESSE     #IP des Domain Controllers
  workgroup = AD
  idmap uid = 10000-20000
  idmap gid = 10000-20000
  winbind enum users = yes
  winbind enum groups = yes
  winbind cache time = 10
  winbind use default domain = yes
  template homedir = /home/%U
  template shell = /bin/bash
  client use spnego = yes
  client ntlmv2 auth = yes
  encrypt passwords = yes
  restrict anonymous = 2
  domain master = no
  local master = no
  preferred master = no
  os level = 0
```

Jetzt kann der Rechner der Domäne hinzugefügt werden:

```
sudo net ads join -U administrator
```

Sollte hier die Fehlermeldung kommen, dass der DNS Eintrag nicht aktualisiert werden konnte, ist das nicht schlimm. Hierzu die Datei **/etc/hosts** aktualiesieren, dass die den vollen Domain Namen enthält:

```
192.168.1.2          rechner01.eigenedomain.org        rechner01
```

Nun müssen die RPC Dienste noch verbunden werden:

```
sudo net rpc join -U administrator
```

Um die Änderungen zu übernehmen muss nun Winbind neu gestartet werden:

```
sudo systemctl restart winbind
```

Um zu prüfen ob alles wie gewünscht funktioniert können nun anfragen an den AD Server gestellt werden. Mit volgendem Befehl werden alle Benutzer aufgelistet:

```
wbinfo -u
```

System Auth konfigurieren

Kerberos und Winbind laufen jetzt. Jetzt muss das System noch dazu überredet werden, die Auth über Winbind laufen zu lassen. Dazu muss die Datei **/etc/nsswitch.conf** bearbeitet werden.

```
passwd:         compat winbind
group:          compat winbind
shadow:         compat
```

Nun muss Winbind wieder neu gestartet werden:

```
sudo systemctl restart winbind
```

PAM Konfiguration

Zuletzt muss ein Domänen Benutzer noch in die Lage versetzt werden mit dem System zu interagieren. Hierzu folgende Zeile in die Datei **/etc/security/group.conf** einfügen:

```
* ; * ; * ; Al0000-2400 ; floppy, audio, cdrom, video, usb, plugdev, users
```

Nun noch dafür sorgen, dass die Home Verzeichnisse angelegt werden. dazu folgenden Befehl ausführen und den Eintrag "Create home directory on login" aktivieren:

```
sudo pam-auth-update
```

Die folgenden abschließenden Befehle sorgen für die Übernahme der letzten Änderungen:

```
sudo systemctl stop winbind
sudo systemctl restart smbd
sudo systemctl start winbind
```

Einhängen von Netzlaufwerken mit Benutzerrechten

Dazu müssen noch weitere Pakete installiert werden:

```
sudo apt-get install libpam-mount cifs-utils
```

Abschliesende Arbeiten

Damit Domänenbenutzer **sudo** verwenden können, müssen diese entweder in die entsprechende Gruppe, oder es muss eine Datei im Ordner **/etc/sudoers.d/** angelegt werden. Ein Beispiel:

```
%wksadmins      ALL=(ALL:ALL)     ALL
"%domain admins"    ALL= (ALL:ALL) ALL
```

Die WKSAdmins Gruppe verwende ich um den Benutzern auf den lokalen Maschinen Admin Rechte zu geben ohne, dass sie diese in der Domain hätten ;).

Auch wenn das nicht unbedingt notwendig ist, sollte nun der Rechner neu gestartet werden. Dann sollten die Domänen Benutzer in der Lage sein, sich anzumelden. Ein guter Artikel zu dem Thema kann hier gefunden werden.

AD Beitritt Open Suse 16

Hier ist eine **kompakte, kopierfertige Anleitung** für die **Domänenintegration mit Offline-Login** unter **openSUSE Leap 16 / SLE 16**. Einfach **Schritt für Schritt abarbeiten** – von der Installation bis zum Test der Offline-Anmeldung.

□□ Anleitung: openSUSE Leap 16 in Active Directory einbinden (mit Offline-Login)

Voraussetzungen:

- Domänenname: `BEISPIEL.DOMAENE.LAN` (ersetzen!)
- Domänen-Admin: `Administrator` (oder anderer Berechtigter)
- Netzwerkverbindung zum Domänencontroller (DC)

1□ Pakete installieren

```
sudo zypper refresh
sudo zypper install --no-confirm realmd sssd sssd-ad sssd-tools adcli krb5-client oddjob oddjob-mkhomedir
pam_krb5 pam_winbind samba-client
```

2□ Hostname und DNS prüfen

```
# Hostname setzen (muss im AD eindeutig sein)
sudo hostnamectl set-hostname client1.beispiel.domäne.lan


# DNS prüfen (muss den DC auflösen können)
ping dc1.beispiel.domäne.lan
nslookup dc1.beispiel.domäne.lan
```

→ Falls DNS nicht funktioniert:

```
sudo vi /etc/resolv.conf
```

Eintrag hinzufügen:

```
nameserver 192.168.1.10  # IP des DC
search beispiel.domäne.lan
```

3️⃣ Domäne mit `realmd` beitreten

```
# Domäne entdecken (Test)
sudo realm discover BEISPIEL.DOMAENE.LAN

# Domäne beitreten (Passwort des AD-Admins eingeben)
sudo realm join -U Administrator BEISPIEL.DOMAENE.LAN --verbose
```

→ **Erwartete Ausgabe:** `Successfully enrolled machine in realm`

4️⃣ SSSD für Offline-Login konfigurieren

```
sudo vi /etc/sssd/sssd.conf
```

Folgende Zeilen anpassen/ergänzen:

```
[sssd]
domains = BEISPIEL.DOMAENE.LAN
config_file_version = 2
services = nss, pam, sudo, ssh

[domain/BEISPIEL.DOMAENE.LAN]
ad_domain = BEISPIEL.DOMAENE.LAN
krb5_realm = BEISPIEL.DOMAENE.LAN
realmd_tags = manages-system joined-with-adcli
cache_credentials = true          # Offline-Login aktivieren
```

```
krb5_store_password_if_offline = true
default_shell = /bin/bash
ldap_id_mapping = true
use_fully_qualified_names = false  # Anmeldung mit "benutzername" statt "benutzername@domäne.lan"
fallback_homedir = /home/%u
access_provider = ad
entry_cache_timeout = 1209600     # Cache-Gültigkeit: 14 Tage (in Sekunden)
account_cache_expiration = 14     # Account-Cache: 14 Tage
```

Berechtigungen setzen:

```
sudo chmod 600 /etc/sssd/sssd.conf
```

5️⃣ PAM für Home-Verzeichnis und Offline-Login konfigurieren

```
# Automatische Erstellung von Home-Verzeichnissen aktivieren
sudo pam-config --add --mkhomedir

# PAM-Konfiguration prüfen
sudo vi /etc/pam.d/common-session
```

Folgende Zeile hinzufügen (falls nicht vorhanden):

```
session required pam_mkhomedir.so skel=/etc/skel umask=0022
```

6️⃣ Dienste neu starten

```
sudo systemctl restart sssd
sudo systemctl enable --now oddjobd
```

7️⃣ Test: Online-Anmeldung (Cache füllen)

```
# Testbenutzer anmelden (ersetze "testuser" mit einem AD-Benutzer)
su - testuser
exit

# Kerberos-Ticket prüfen
klist
```

→ **Erwartet:** Ein gültiges Ticket für `testuser@BEISPIEL.DOMAENE.LAN`.

---

8 Test: Offline-Login

```
# Netzwerk trennen (z. B. Kabel ziehen oder WLAN deaktivieren)
sudo systemctl stop NetworkManager

# Offline-Login testen
su - testuser  # Sollte funktionieren!
id testuser    # Sollte Benutzerdaten anzeigen
klist -l       # Sollte gecachte Tickets zeigen
```

---

9 Fehlersuche (falls nötig)

Problem	Befehl zur Diagnose
Anmeldung fehlgeschlagen	`journalctl -u sssd -f`
Keine Benutzer auflösbar	`getent passwd testuser`
Kerberos-Fehler	`kinit testuser` (manueller Test)
DNS-Probleme	`dig dc1.beispiel.domäne.lan`

Wichtige Dateien & Befehle im Überblick

Zweck	Datei/Befehl
SSSD-Konfiguration	`/etc/sssd/sssd.conf`
Kerberos-Einstellungen	`/etc/krb5.conf`

Zweck	Datei/Befehl
PAM-Konfiguration	`/etc/pam.d/common-session`
Cache löschen	`sudo systemctl stop sssd; sudo rm -f /var/lib/sss/db/cache_*.ldb; sudo systemctl start sssd`
Ticket-Cache prüfen	`klist -l`
Benutzerdaten prüfen	`id testuser`

Fertig!

- Der Rechner ist jetzt in der Domäne und unterstützt **Offline-Login**.
- **Wichtig:** Jeder Benutzer muss sich **mindestens einmal online anmelden**, bevor er offline arbeiten kann.

Optional: Automatische Ticket-Erneuerung konfigurieren

Falls Kerberos-Tickets zu schnell ablaufen:

```
sudo vi /etc/krb5.conf
```

Folgende Zeilen anpassen:

```
[libdefaults]
    ticket_lifetime = 24h
    renew_lifetime = 7d
    forwardable = true
```

Dann **SSSD neu starten**:

```
sudo systemctl restart sssd
```

Tipp: Falls **Smartcards** oder **Zertifikatsauthentifizierung** benötigt werden, muss zusätzlich `pam_pkcs11` oder `sssd` mit `certificate`-Backend konfiguriert werden. Lass es mich wissen, falls du das brauchst!

Viel Erfolg!