KI Server

Installationsanleitung KI Server mit GPU Unterstützung basierend auf ollama mit LLMs wie LLama3.1:8b usw.

Einleitung

Der **KI-Server** bildet das Herzstück der lokalen KI- und Automatisierungsumgebung im Homelab. Er dient als zentrale Plattform für die Ausführung von **großen Sprachmodellen (LLMs)** sowie für **KI-gestützte Anwendungen** wie Dokumentenanalyse, Bildverarbeitung und Automatisierungs-Workflows.

Durch die Virtualisierung auf **Proxmox VE 9** und die Integration einer dedizierten GPU können leistungsfähige Modelle wie *GPT-OSS:20b, Llama 3.1:12b, Mistral:7b, Gemma3:12b, DeepSeek-R1:8b* und *Qwen3:14b* lokal betrieben werden.
Damit ist der Server unabhängig von Cloud-Diensten nutzbar und bietet volle Kontrolle über Daten, Sicherheit und Performance.

Einsatzbereiche des KI-Servers:

- **Dokumentenmanagement & Automatisierung**: Anbindung an Paperless-NGX, Immich App und Paperless-AI zur automatischen Dokumentenerfassung und -analyse.
- **Assistenzsysteme**: Nutzung von OpenWebUI und Ollama für Chatbots, persönliche Assistenten und interaktive Workflows.
- **Security & Schwachstellenmanagement**: Geplante Integration in bestehende Security-Tools (z. B. Greenbone, osTicket, n8n-Automatisierungen).
- **Experimentelle KI-Projekte**: Testumgebung für lokale Sprach- und Bildmodelle, u. a. zur Text-, Sprach- und Bildgenerierung.

Damit bildet der Server die Grundlage für ein **flexibles, erweiterbares und sicheres KI-Ökosystem** im Homelab, das sowohl für den produktiven Einsatz als auch für Test- und Forschungszwecke genutzt wird.

Hardwareübersicht

Der aktuelle KI-Server basiert auf leistungsfähiger Gaming- und Serverhardware, die speziell für Virtualisierung und GPU-Passthrough ausgelegt ist:

- **Mainboard**: ASUS TUF Gaming Z790 Pro Wifi
- **CPU**: Intel i5-13400 (10 Kerne, 16 Threads)
- **RAM**: 128 GB DDR5 (2 × 64 GB)
- **GPU**: MSI GeForce RTX 5060 Ti 16G Ventus 2X OC Plus (16 GB VRAM, passthrough-fähig)
- **Netzwerk**: Dual 10 Gbit LAN (Intel X540-T2)
- **Speicher**:
  - Samsung NVMe 990 Pro 2 TB (System & VM-Speicher)
  - Crucial NVMe 4 TB (Schneller Datenspeicher für Modelle und Container)
  - 10 TB SATA HDD (Langzeitspeicher)
- **Netzteil**: ASUS TUF Gaming 750W Gold
- **Gehäuse**: 19" Servergehäuse

Betriebssystem: **Proxmox VE 9** (Virtualisierungshost)
KI-VM: **Ubuntu 24.04.3 LTS** mit durchgereichter GPU

Zusammenbau Server

Der Zusammenbau erfolgt in einem 19"-Servergehäuse und erfordert besondere Aufmerksamkeit auf **Kühlung, Stromversorgung und die richtige Anordnung der PCIe-Geräte**.

Vorbereitung

- **Arbeitsplatz**: Antistatik-Armband oder Erdung, saubere und helle Arbeitsfläche.
- **Komponenten prüfen**: Alle Bauteile (Mainboard, CPU, RAM, GPU, NVMe, Netzwerkkarte, Netzteil, Gehäuse) vorab bereitlegen.
- **Werkzeug**: Kreuzschlitz-Schraubendreher, ggf. Drehmomentschrauber für CPU-Kühler, Kabelbinder für sauberes Kabelmanagement.

Schritt-für-Schritt Aufbau

1. Mainboard montieren

- Das **ASUS TUF Gaming Z790 Pro Wifi** ins Gehäuse einsetzen und mit Abstandshaltern korrekt verschrauben.
- I/O-Shield prüfen (bei TUF-Boards oft integriert).

2. CPU installieren

- Den **Intel i5-13400** vorsichtig in den Sockel LGA1700 einsetzen (Markierung beachten).
- Verriegelung gleichmäßig schließen.
- Wärmeleitpaste mittig auftragen (Erbsengröße).
- CPU-Kühler montieren (auf gleichmäßigen Anpressdruck achten).

3. RAM installieren

- **128 GB DDR5 (2×64 GB)** einsetzen.
- Für **Dual-Channel-Performance** die Slots A2 und B2 belegen (laut Handbuch).
- Darauf achten, dass die Riegel vollständig einrasten.

4. NVMe-Speicher einsetzen

- **Samsung 990 Pro 2 TB** (System & VMs): Auf den **CPU-nahesten M.2-Slot** setzen (PCIe 4.0 ×4, direkt an die CPU angebunden).
- **Crucial 4 TB NVMe** (Daten/Modelle): In den zweiten M.2-Slot (ebenfalls PCIe 4.0 ×4, über Chipsatz).
- Wärmeleitpads/Heatsinks montieren.

> **Hinweis PCIe-Lanes**:
>
> - Die **erste NVMe** nutzt CPU-Lanes → maximaler Durchsatz (wichtig für VMs und Host-System).
> - Weitere NVMe laufen über den Z790-Chipsatz (DMI 4.0 Link) → ausreichend für Modell- und Containerdaten.

5. GPU installieren

- **MSI RTX 5060 Ti 16G** in den **obersten PCIe-x16-Slot** (direkt an CPU angebunden, volle PCIe 4.0 ×16 Bandbreite).
- Karte fixieren, zusätzliche 8-Pin-Stromversorgung anschließen.
- Darauf achten, dass keine anderen PCIe-Karten Bandbreite vom GPU-Slot abzweigen.

6. 10 Gbit LAN (Intel X540) installieren

- Karte in den **zweiten mechanischen x16-Slot** einsetzen. Dieser teilt sich bei Nutzung ggf. Lanes mit M.2 oder anderen PCIe-Slots → im BIOS prüfen.
- Für den Betrieb reichen PCIe 3.0 ×8; selbst wenn der Slot reduziert, ist die Bandbreite (ca. 64 Gbit/s) deutlich über den benötigten 20 Gbit/s.

7. SATA-Speicher anschließen

- **10 TB HDD** am SATA-Port anschließen (über Chipsatz, keine Performance-Probleme).
- Optional Vibrationsdämpfer im Gehäuse nutzen.

8. Netzteil und Verkabelung

- **ASUS TUF Gaming 750W Gold** installieren.
- 24-Pin ATX- und 8-Pin CPU-Kabel sauber verlegen.

- GPU mit **dediziertem Kabelstrang** anschließen (kein Daisy-Chaining).
- Kabelmanagement mit Kabelbindern, um Luftstrom nicht zu behindern.

BIOS Anpassungen

1. **BIOS Update** auf aktuelle Version durchführen.
2. **Resizable BAR** aktivieren (wichtig für GPU-Leistung).
3. **Virtualisierung**: Intel VT-d & SR-IOV aktivieren (für Proxmox & GPU Passthrough).
4. **PCIe-Konfiguration**: Oberster Slot auf „PCIe Gen4“ fixieren.
5. **Lüfterkurven**: Auf 24/7-Betrieb optimieren (hoher Airflow, moderater Lärm).
6. **Power Saving Features**: C-States ggf. einschränken, um Latenzen bei KI-Workloads zu reduzieren.

Installation

Installation Host

Installation von Proxmox VE 9

1. ISO herunterladen

- Offizielle Downloadseite:
  https://www.proxmox.com/en/downloads
- Gewählt wird: **Proxmox VE 9.x ISO Installer** (aktuell `proxmox-ve_9.x.iso`).

---

2. Bootfähigen USB-Stick unter Windows erstellen

Am einfachsten mit **Rufus**:

1. Download Rufus: https://rufus.ie
2. USB-Stick (≥ 4 GB, wird gelöscht) einstecken.
3. Rufus starten → folgendes einstellen:
   - **Gerät**: USB-Stick auswählen
   - **Boot-Auswahl**: heruntergeladene `proxmox-ve_9.x.iso`
   - **Partitionstyp**: GPT
   - **Zielsystem**: UEFI (nicht BIOS/Legacy)
   - Rest Standard lassen
4. Start → ISO wird auf den Stick geschrieben.

---

3. Installation von Proxmox VE

1. Server einschalten und vom USB-Stick booten (UEFI-Modus wählen).
2. Installationsmenü → `Install Proxmox VE` wählen.
3. Lizenzbedingungen akzeptieren.
4. **Zieldatenträger auswählen**:
   - Samsung NVMe 990 Pro 2 TB (Systemplatte).
   - Dateisystem: **ext4** (einfach und stabil, ZFS nur bei RAID oder Snapshots nötig).
5. Zeitzone: `Europe/Berlin`, Tastaturlayout `de`.
6. Root-Passwort und E-Mail-Adresse setzen (für Benachrichtigungen).
7. Netzwerkeinstellungen:
   - Hostname: `pve.localdomain` (später anpassen).
   - Management-IP manuell vergeben.
8. Installation starten, nach Abschluss neustarten.

---

4. Erste Anmeldung

- Webinterface über: `https://<IP-des-Servers>:8006`
- Login mit `root` und dem gesetzten Passwort.

---

5. BIOS-Anpassungen für GPU-Passthrough

Vor dem Start der VMs müssen im BIOS folgende Optionen aktiviert werden:

- **Intel VT-d** → `Enabled`
- **SR-IOV** → `Enabled`
- **Above 4G Decoding** → `Enabled`
- **Resizable BAR** → `Enabled` (falls vorhanden, für GPU Performance)
- **Primary Display** → auf `iGPU` oder `Onboard` stellen (damit die NVIDIA-Karte für Passthrough frei ist).

---

6. GRUB & Kernel-Anpassungen für Passthrough

Auf dem Proxmox-Host anmelden (SSH oder Shell).

a) IOMMU aktivieren

Datei `/etc/default/grub` bearbeiten:

```
nano /etc/default/grub
```

Die Zeile mit `GRUB_CMDLINE_LINUX_DEFAULT` anpassen:

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet intel_iommu=on iommu=pt"
```

Speichern und GRUB neu generieren:

```
update-grub
```

b) VFIO-Module laden

Datei `/etc/modules` bearbeiten und folgende Zeilen ergänzen:

```
vfio
vfio_iommu_type1
vfio_pci
vfio_virqfd
```

---

7. GPU identifizieren & binden

1. PCI-Geräte auflisten:

   ```
   lspci -nn
   ```

   → GPU (z. B. `10de:2805`) und Audio-Controller (`10de:228b`) notieren.
2. Datei `/etc/modprobe.d/vfio.conf` erstellen:

`options vfio-pci ids=10de:2805,10de:228b`

3. NVIDIA-Treiber auf Host blockieren:
   Datei `/etc/modprobe.d/blacklist.conf` ergänzen:

   ```
   blacklist nouveau
   blacklist nvidia
   blacklist nvidiafb
   ```

4. Initramfs neu erstellen:

   ```
   update-initramfs -u -k all
   ```

---

8. Neustart & Kontrolle

Nach Neustart prüfen, ob die GPU an VFIO gebunden ist:

```
lspci -nnk | grep -A 3 -E "10de"
```

→ Sollte `Kernel driver in use: vfio-pci` anzeigen.

Einrichten VM

Installation Ubuntu 24.04.3 LTS (Minimal Server)

1. ISO herunterladen

Offizielle Quelle:

https://ubuntu.com/download/server

- Variante: **Ubuntu Server 24.04.3 LTS**
- Image: `ubuntu-24.04.3-live-server-amd64.iso`

---

2. VM in Proxmox anlegen

1. In der Proxmox Web-GUI: **Create VM**
   - Name: `KI-VM`
   - ISO: `ubuntu-24.04.3-live-server-amd64.iso` (zuvor hochgeladen in Proxmox Storage)
   - System:
     - BIOS: **OVMF (UEFI)**
     - Machine: **q35**
     - Graphic Card: **none** (GPU wird durchgereicht)
   - Disks:
     - Speicher auf **Crucial NVMe 4 TB** oder System-SSD je nach Planung
     - Größe: mind. 250 GB (je nach Modellgrößen)
     - Cache: **Write back (unsafe)** (für Performance)
   - CPU:
     - 6 Cores (i5-13400 hat 10 → 6 reichen für KI-Workloads, Rest für Host)
     - Typ: **host**
   - RAM:

    - 32 GB (von 128 GB gesamt)
    - Ballooning: deaktivieren (konstante Zuweisung)
- Netzwerk: **VirtIO (paravirtualized)**
- Hardware: **GPU & Audio-Device** via PCI Passthrough hinzufügen (wie vorher eingerichtet).

---

3. Ubuntu Installation

1. Boot von ISO → Auswahl: **Install Ubuntu Server**
2. Sprache: `Deutsch` (oder Englisch, je nach Vorliebe)
3. Tastatur: `Deutsch`
4. Netzwerk:
    - `ens18` (VirtIO) automatisch via DHCP oder manuell konfigurieren
    - Empfehlung: **statische IP** für die VM (z. B. 192.168.33.200)
5. Storage:
    - Geführte Installation auf die virtuelle Disk
    - Partitionierung: Standard (LVM möglich, aber nicht zwingend)
6. Benutzer anlegen:
    - Username: `kiadmin`
    - Passwort: sicher setzen
    - SSH-Server installieren: **Ja**
7. Snap-Pakete: alle abwählen (nicht benötigt)
8. Installation starten → Neustart nach Abschluss

---

4. Erste Anpassungen nach der Installation

Nach Login per SSH oder Console:

a) System aktualisieren

```
sudo apt update && sudo apt upgrade -y
sudo reboot
```

b) Nützliche Basis-Tools installieren

```
sudo apt install -y htop ncdu git curl wget unzip zip tar net-tools iftop lsb-release pciutils
```

c) SSH konfigurieren

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

Empfohlen:

- `PermitRootLogin no`
- `PasswordAuthentication no` (falls SSH-Key genutzt wird)

Dann:

```
sudo systemctl restart ssh
```

d) Zeitsynchronisation prüfen

```
timedatectl set-timezone Europe/Berlin
timedatectl status
```

---

5. GPU-Treiber vorbereiten

Da die GPU durchgereicht wird, braucht die VM die NVIDIA-Treiber:

a) Repository aktivieren

```
sudo apt install -y software-properties-common
sudo add-apt-repository ppa:graphics-drivers/ppa -y
sudo apt update
```

b) NVIDIA-Treiber installieren

```
sudo apt install -y nvidia-driver-550 nvidia-utils-550
```

c) Neustart & Test

```
nvidia-smi
```
→ sollte GPU-Daten (Modell RTX 5060 Ti, Treiber, CUDA-Version) anzeigen.

---

6. Optional: Basis-Optimierungen für KI-Workloads

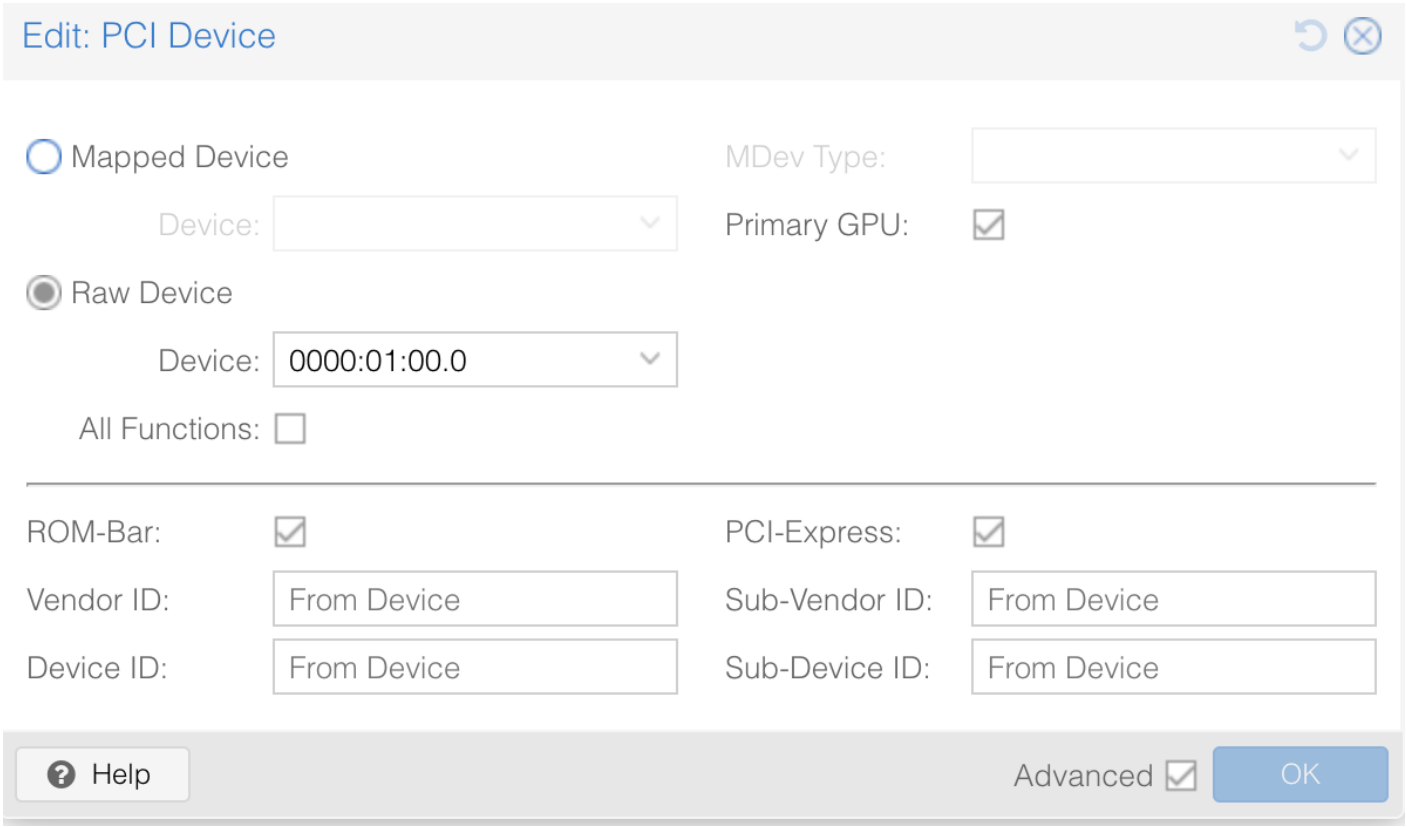
- **Swappiness reduzieren** (damit RAM bevorzugt genutzt wird):

  ```
  echo 'vm.swappiness=10' | sudo tee -a /etc/sysctl.conf
  sudo sysctl -p
  ```
- **Transparent Hugepages deaktivieren** (manchmal Performance-Gewinn bei LLMs):

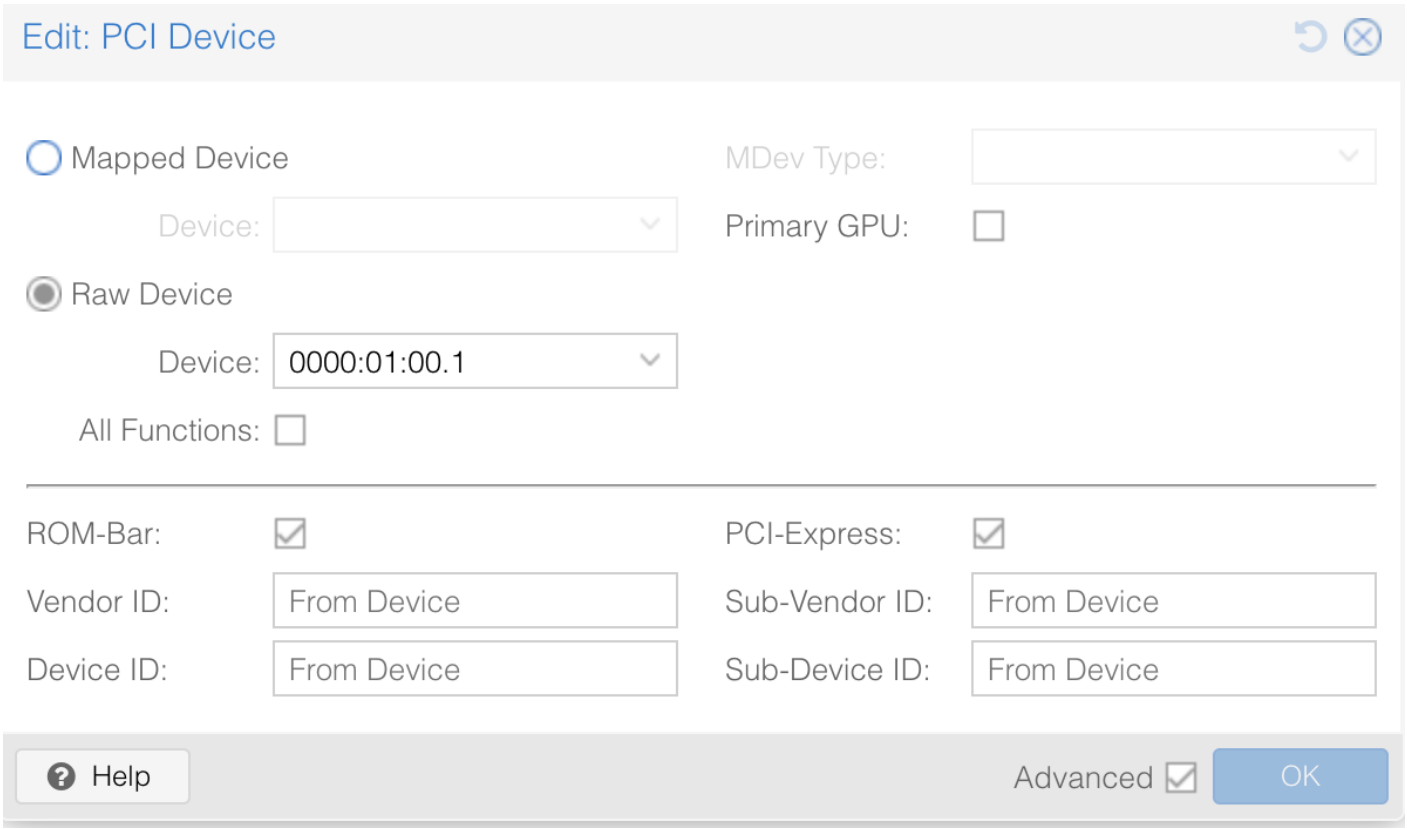
  ```
  echo never | sudo tee /sys/kernel/mm/transparent_hugepage/enabled
  ```
- **ufw aktivieren** (Firewall, falls nicht durch Docker geregelt):

  ```
  sudo apt install ufw -y
  sudo ufw default deny incoming
  sudo ufw default allow outgoing
  sudo ufw allow ssh
  sudo ufw enable
  ```

Konfiguration GPU:



Konfiguration GPU-Audio:



Installation Docker

Installation von Docker & Docker Compose

1. Alte Versionen entfernen (falls vorhanden)

Vorherige Docker-Installationen oder Reste löschen:

```
sudo apt remove -y docker docker-engine docker.io containerd runc
```

---

2. Abhängigkeiten installieren

```
sudo apt update
sudo apt install -y ca-certificates curl gnupg lsb-release
```

---

3. Docker GPG-Key hinzufügen

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

---

4. Docker-Repository hinzufügen

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

---

5. Docker Engine installieren

```
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

---

6. Installation prüfen

```
docker --version
docker compose version
```

---

7. Benutzerrechte anpassen

Standardmäßig braucht man `sudo` für Docker. Damit normale Nutzer (z. B. `kiadmin`) Docker nutzen können:

```
sudo usermod -aG docker kiadmin
```

→ Danach **ab- und wieder anmelden**.

---

8. Docker als Dienst aktivieren

Damit Docker beim Booten automatisch startet:

```
sudo systemctl enable docker
sudo systemctl start docker
```

---

9. Test mit Hello-World Container

```
docker run hello-world
```

→ Sollte eine Bestätigungsmeldung ausgeben.

---

10. (Optional) Standard-Speicherpfad anpassen

Falls Container und Images nicht auf der Systemplatte, sondern auf einer dedizierten NVMe liegen sollen:

1. Docker-Dienst stoppen:

   ```
   sudo systemctl stop docker
   ```
2. Konfigurationsdatei erstellen:

   ```
   sudo mkdir -p /etc/docker
   echo '{
     "data-root": "/opt/docker"
   }' | sudo tee /etc/docker/daemon.json
   ```
3. Verzeichnis anlegen & Rechte setzen:

   ```
   sudo mkdir -p /opt/docker
   sudo chown -R root:docker /opt/docker
   ```
4. Docker neu starten:

   ```
   sudo systemctl start docker
   ```

Docker Container

Docker Container

Installation Ollama und OpenWebUI

1. Verzeichnisstruktur vorbereiten

Alle Containerdaten liegen unter `/opt/docker`, damit System und Daten sauber getrennt bleiben:

```
sudo mkdir -p /opt/docker/ollama
sudo mkdir -p /opt/docker/open-webui
```

Optional Zugriffsrechte setzen:

```
sudo chown -R kiadmin:docker /opt/docker
```

---

2. Docker-Compose Datei

Pfad: `/opt/docker/docker-compose.yml`

```
version: '3.8'

services:
  ollama:
    image: ollama/ollama
    container_name: ollama
    ports:
      - "11434:11434"
    restart: always
    runtime: nvidia
    volumes:
      - /opt/docker/ollama:/root/.ollama

  openwebui:
    image: ghcr.io/open-webui/open-webui:ollama
    container_name: open-webui
    restart: always
```

```
    runtime: nvidia
    ports:
      - "3000:8080"
    environment:
      - OLLAMA_API_BASE_URL=http://ollama:11434
    volumes:
      - /opt/docker/open-webui:/app/backend/data
      # Eigene CSS-Anpassungen (optional)
      # - /home/pleibling/docker/ai/custom.css:/app/build/static/custom.css:ro
    depends_on:
      - ollama
```

---

3. Container starten

```
cd /opt/docker
docker compose up -d
```

Status prüfen:

```
docker ps
```

- Ollama-API → `http://<Server-IP>:11434`
- OpenWebUI → `http://<Server-IP>:3000`

---

4. Modelle installieren

Ollama lädt Modelle nach Bedarf.
Installation erfolgt z. B. über:

```
docker exec -it ollama ollama pull gpt-oss:20b
```

Verwendete LLMs in dieser Umgebung:

- **GPT-OSS:20b** → Hauptmodell (groß, sehr leistungsfähig)
- **Llama 3.1:12b**
- **Mistral:7b**
- **Gemma3:12b**
- **DeepSeek-R1:8b**
- **Qwen3:14b**

Optional kannst du auch in OpenWebUI für jedes Modell eigene Presets/Workspaces definieren.

5. Update der Container

```
cd /opt/docker
docker compose pull
docker compose up -d
```

6. Backup-Hinweis

Da alle persistenten Daten in `/opt/docker/ollama` und `/opt/docker/open-webui` liegen, reicht ein Backup dieser Ordner.

Docker Container

Installation Paperless und Paperless-AI

Verzeichnis anlegen (einmalig)

```
sudo mkdir -p /home/pleibling/docker/paperless/{data,media,export,consume,db,redis,ai}
sudo chown -R 1000:1000 /home/pleibling/docker/paperless
```

`docker-compose.yml` (unter `/home/pleibling/docker/paperless/docker-compose.yml`)

```
version: "3.8"

services:
  paperless-ngx:
    image: ghcr.io/paperless-ngx/paperless-ngx:latest
    container_name: paperless-ngx
    restart: always
    environment:
      - PAPERLESS_REDIS=redis://paperless-redis:6379
      - PAPERLESS_DBHOST=paperless-db
      - PAPERLESS_DBUSER=paperless
      - PAPERLESS_DBPASS=PapPW050725
      - PAPERLESS_SECRET_KEY=G2v3eKLZRIkpMeUcGkLor0Lt6vtzHodKLCRVvYHHjtE=
      - PAPERLESS_AI_ENABLED=1
      - PAPERLESS_AI_PROVIDER=ollama
      - PAPERLESS_AI_MODEL=llama3.1:8b
```

```
      - PAPERLESS_AI_HOST=http://192.168.33.200:11434
      - PAPERLESS_ALLOWED_HOSTS=dms.leibling.de,192.168.33.200,localhost
      -
PAPERLESS_CSRF_TRUSTED_ORIGINS=https://dms.leibling.de,http://192.168.33.200:8080,http://local
host:8080
      - USERMAP_UID=1000
      - USERMAP_GID=1000
      - PAPERLESS_TIKA_ENABLED=true
      - PAPERLESS_TIKA_ENDPOINT=http://tika:9998
      - PAPERLESS_CONVERT_OFFICE=true
      - PAPERLESS_TIKA_CONVERT_OFFICE=true
      - PAPERLESS_GOTENBERG_ENABLED=true
      - PAPERLESS_GOTENBERG_ENDPOINT=http://gotenberg:3000
      - RAG_SERVICE_ENABLED=true
      - RAG_SERVICE_URL=http://paperless-ai:8000
    ports:
      - "8080:8000"
    volumes:
      - /opt/docker/paperless/data:/usr/src/paperless/data
      - /opt/docker/paperless/media:/usr/src/paperless/media
      - /opt/docker/paperless/export:/usr/src/paperless/export
      - /opt/docker/paperless/consume:/usr/src/paperless/consume
    depends_on:
      - paperless-db
      - paperless-redis
  gotenberg:
    image: gotenberg/gotenberg:7
    container_name: paperless-gotenberg
    restart: always
    ports:
      - "3002:3000"
    security_opt:
      - no-new-privileges:true
    cap_drop:
      - ALL

  tika:
    image: apache/tika
    container_name: paperless-tika
    restart: always
```

```
    ports:
      - "3003:9998"
    security_opt:
      - no-new-privileges:true
    cap_drop:
      - ALL

  paperless-db:
    image: postgres:15
    container_name: paperless-db
    restart: always
    environment:
      - POSTGRES_DB=paperless
      - POSTGRES_USER=paperless
      - POSTGRES_PASSWORD=PapPW050725
    volumes:
      - /opt/docker/paperless/db:/var/lib/postgresql/data

  paperless-redis:
    image: redis:7
    container_name: paperless-redis
    restart: always
    volumes:
      - /opt/docker/paperless/redis:/data

  paperless-ai:
    image: clusterzx/paperless-ai
    container_name: paperless-ai
    restart: always
    environment:
      - PUID=1000
      - PGID=1000
      - PAPERLESS_AI_PORT=3000
      - RAG_SERVICE_ENABLED=true
      - RAG_SERVICE_URL=http://paperless-ai:3000
    ports:
      - "3010:3000"
    volumes:
      - /opt/docker/paperless/ai:/app/data
    deploy:
```

```
      resources:
        reservations:
          devices:
            - capabilities: ["gpu"]
```

Wichtig:

- Deine **Ollama-Instanz** läuft in einem anderen Stack (Ollama/OpenWebUI). Zugriff erfolgt hier **per IP/Port** (`http://192.168.33.200:11434`) – das ist ideal, da unterschiedliche Compose-Netzwerke sich sonst nicht automatisch sehen.
- Wenn du später **TLS/Reverse Proxy** (z. B. Traefik/Caddy/Nginx) nutzt, passe `PAPERLESS_ALLOWED_HOSTS` und `PAPERLESS_CSRF_TRUSTED_ORIGINS` an.

Start/Update

```
cd /home/pleibling/docker/paperless
docker compose up -d
# Update:
docker compose pull && docker compose up -d
```

Kurzer Funktionstest

- Paperless-NGX: `http://<VM-IP>:8080`
- RAG-API (intern gemappt): `http://<VM-IP>:3010/health` → sollte `ok` liefern
- Tika: `http://<VM-IP>:3003/version`
- Gotenberg: `http://<VM-IP>:3002/health`

Docker Container

Installation Immich-App

Immich – Installation (Minimal Setup)

```
cd /opt/docker
git clone https://github.com/immich-app/immich.git immich-app
cd immich-app
```

`.env` erstellen

Datei: `/opt/docker/immich-app/.env`

```
# You can find documentation for all the supported env variables at https://immich.app/docs/install/environment-variables

# The location where your uploaded files are stored
UPLOAD_LOCATION=./library

# The location where your database files are stored. Network shares are not supported for the database
DB_DATA_LOCATION=./postgres

# To set a timezone, uncomment the next line and change Etc/UTC to a TZ identifier from this list: https://en.wikipedia.org/wiki/List_of_tz_database_time_zones#List
TZ=Europe/Berlin

# The Immich version to use. You can pin this to a specific version like "v1.71.0"
IMMICH_VERSION=release
```

```
# Connection secret for postgres. You should change it to a random password
# Please use only the characters `A-Za-z0-9`, without special characters or spaces
DB_PASSWORD=kdlIHdjdfifhj7f7ehekdhuvzdjenfifhHJZHGjdu65w7sh

# The values below this line do not need to be changed
################################################################################
DB_USERNAME=postgres
DB_DATABASE_NAME=immich
```

Starten

```
docker compose up -d
```

Aufruf im Browser

`http://<VM-IP>:2283`

Erster registrierter Benutzer = Administrator.
