ADMIN:DOCS

Immich: Private Foto-Cloud unter Ubuntu 24.04 LTS Server installieren

Installiere und betreibe die selbstgehostete Foto-Cloud Immich auf Ubuntu 24.04 LTS mit Docker Compose, PostgreSQL mit pgvector, Hardware-Transcoding und Nginx.

Kategorie: Serverumgebungen **Veröffentlicht:** 12.05.2025 **Zuletzt geprüft:** 08.09.2026

Immich ist eine selbstgehostete Plattform zur Verwaltung und Sicherung von Fotos und Videos, die als eigenständige Alternative zu kommerziellen Cloud-Diensten wie Google Photos oder Apple iCloud konzipiert ist. Die Software kombiniert eine reaktionsschnelle Weboberfläche mit mobilen Begleit-Apps für Android und iOS, automatischem Hintergrund-Upload, Zeitleisten-Navigation und Albenverwaltung.

Unter der Haube unterscheidet sich Immich deutlich von traditionellen Bildgalerien.

Die Plattform setzt auf moderne Machine-Learning-Pipelines für die Gesichtserkennung und semantische Bildsuche auf Basis von CLIP-Vektormodellen, automatische Transkodierung hochauflösender Videos und eine strenge Trennung zwischen Anwendungslogik, Caching und persistenter Metadatenhaltung.

In dieser Anleitung richtest du Immich auf einem Server mit Ubuntu 24.04 LTS produktiv ein. Wir betrachten die System- und Speicherarchitektur, die Container-Bereitstellung über Docker Compose, die Integration von Hardware-Transcoding, die Absicherung über einen Nginx-Reverse-Proxy mit TLS sowie automatisierte Datenbank-Dumps und Wiederherstellungsstrategien für den Ernstfall.

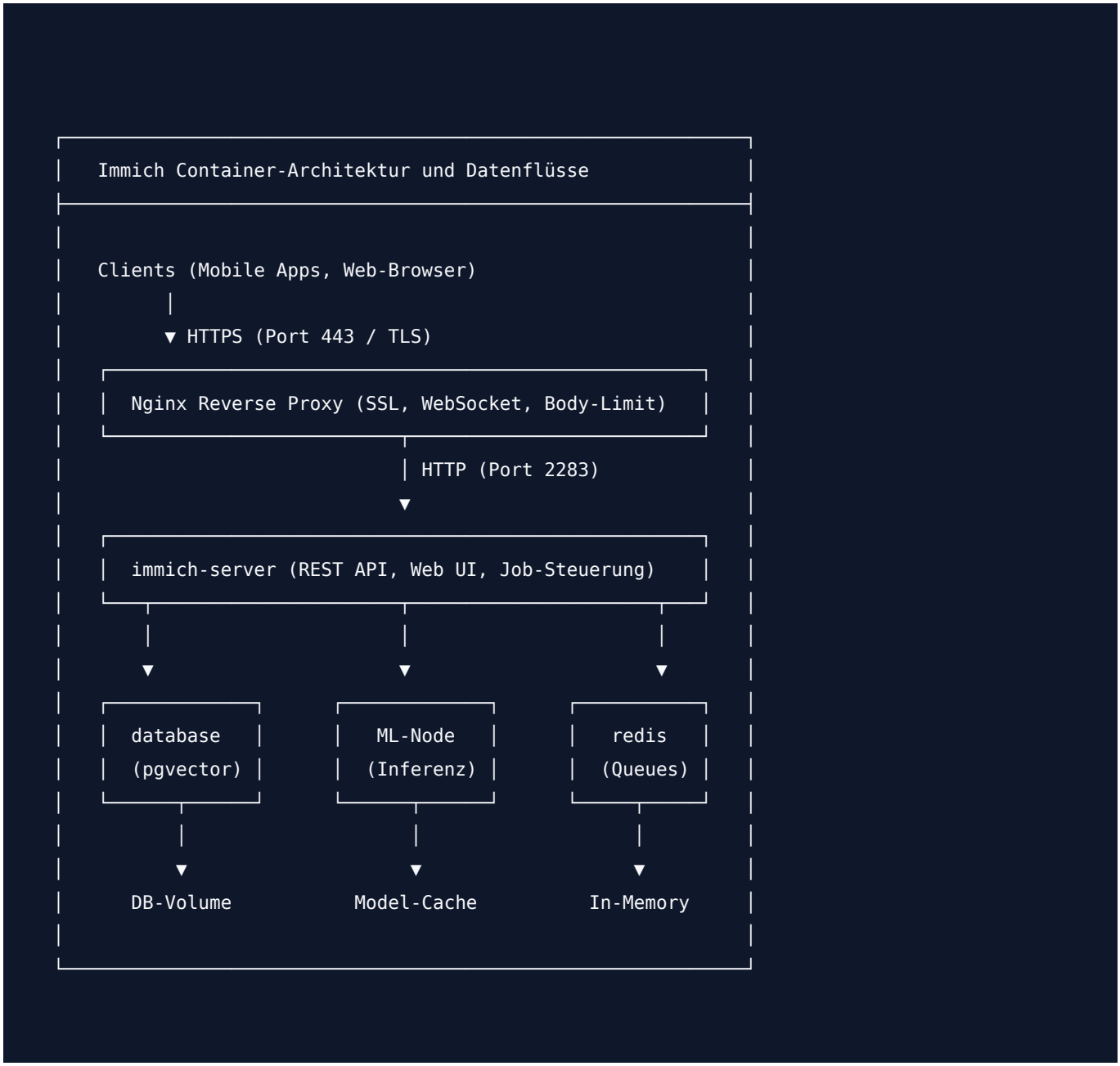
Quelle: https://admindocs.de/de/serverumgebungen/immich-private-foto-cloud-unter-ubuntu-24-04-lts-server-installieren

> **Kompatibilität mit Ubuntu 26.04 LTS:** Da Immich vollständig containerisiert über Docker Compose betrieben wird, sind sämtliche Installations- und Konfigurationsschritte dieser Anleitung 1:1 auch auf **Ubuntu 26.04 LTS** anwendbar. Falls du dein bestehendes Hostsystem aktualisieren möchtest, findest du alle Schritte im separaten Leitfaden Ubuntu-Upgrade: Von Version 24.04 LTS auf 26.04 LTS.

Architektur und Komponenten

Ein stabiler Betrieb von Immich erfordert ein klares Verständnis der beteiligten Microservices. Das Gesamtsystem besteht aus vier zentralen Containern, die über ein isoliertes Docker-Netzwerk miteinander kommunizieren:

- `immich-server`: Die Kernkomponente nimmt API-Anfragen der Clients entgegen, liefert die Weboberfläche aus, steuert Benutzer- und Rechteverwaltung, verwaltet Uploads und koordiniert Hintergrund-Jobs.
- `immich-machine-learning`: Ein spezialisierter Inferenz-Dienst auf Python-Basis. Er berechnet hochdimensionale Vektoreinbettungen (CLIP) für die Volltext-Bildsuche, extrahiert Gesichtsmerkmale (Facial Recognition) und erfordert Prozessor-Instruktionen wie AVX/AVX2 oder dedizierte GPU-Hardware.
- `database` **(PostgreSQL mit Vektorerweiterung)**: Als relationale Datenbank kommt kein Standard-PostgreSQL-Image zum Einsatz, sondern eine mit `pgvector` bzw. `vectorchord` ausgestattete PostgreSQL-Instanz (`ghcr.io/immich-app/postgres`). Sie speichert Bildmetadaten, Benutzerprofile, EXIF-Attribute und Vektor-Indizes für Ähnlichkeitssuchen.
- `redis` / `valkey`: Ein speicherbasierter Schlüssel-Wert-Speicher, der als Nachrichten-Broker und Job-Queue dient. Er puffert asynchrone Aufgaben wie das Erstellen von Vorschaubildern, Video-Transkodierung und Vektorberechnungen zwischen dem Server und den Hintergrund-Workern ab.



Die Trennung dieser Aufgaben stellt sicher, dass rechenintensive Prozesse wie das Einlesen zehntausender neuer Fotos die Benutzeroberfläche nicht blockieren.

Hardware- und Systemanforderungen

Immich stellt im Vergleich zu rein statischen Galerien spürbare Anforderungen an Prozessor, Arbeitsspeicher und Speicher-I/O. Die automatische Indexierung tausender RAW-Dateien, Videos und Bilder erzeugt erhebliche Lastspitzen.

Dimensionierung der Kernkomponenten

Plane die Hardwareressourcen anhand der Bibliotheksgröße und der Anzahl paralleler Nutzer:

Einsatzbereich	CPU-Kerne	Arbeitsspeicher	Empfohlener Speicher
Einzelnutzer / Test	2 Kerne (x86_64 mit AVX)	4 GB RAM	60 GB SSD (System + DB) + Medien
Familie / Produktion	4 Kerne (AVX2-Support)	8 GB bis 16 GB RAM	120 GB NVMe (DB) + HDD/ZFS-Pool
Große Sammlungen (100k+)	6+ Kerne / iGPU (QuickSync)	16 GB bis 32 GB RAM	250 GB NVMe (DB/Cache) + Massenspeicher

> ⚠ **Wichtige CPU-Voraussetzung:** Der Machine-Learning-Container setzt für eine akzeptable Inferenzgeschwindigkeit standardmäßig moderne Vektor-Befehlssatzerweiterungen wie AVX oder AVX2 voraus. Auf sehr alten Host-CPUs oder fehlkonfigurierten VM-Hypervisoren (z. B. Proxmox-Standardtyp `kvm64` statt `host`) stürzt der ML-Container beim Laden der Modelle ab.

Überprüfe die CPU-Flags auf deinem Ubuntu-Server vorab im Terminal:

```
grep -E 'avx|avx2' /proc/cpuinfo
```

Erscheint hier keine Ausgabe, fehlen dem virtuellen oder physischen Prozessor die AVX-Erweiterungen. In virtualisierten Umgebungen stellst du das CPU-Modell in den Hypervisor-Einstellungen auf `host`.

Speicherarchitektur: Trennung von Metadaten und Medien

In Produktivumgebungen empfiehlt sich eine saubere Trennung der Speicherpfade:

- **PostgreSQL-Datenbank und Thumbnails:** Gehören zwingend auf schnellen, lokalen Flash-Speicher (NVMe- oder SATA-SSDs). Hohe I/O-Latenzen verlangsamen das Scrollen durch die Zeitleiste drastisch.
- **Medienbibliothek (Originale):** Kann auf großen Festplatten-Arrays (RAID-ZFS, mdadm) oder externen Mountpoints liegen.

> ⚠ **Datenbank niemals auf Netzwerkfreigaben:** Das Datenbank-Volume (`DB_DATA_LOCATION`) darf unter keinen Umständen auf NFS-, CIFS- oder SMB-Freigaben abgelegt werden. PostgreSQL benötigt striktes POSIX-Dateisperren und synchrone Schreibgarantien. Netzwerkdateisysteme führen bei PostgreSQL reproduzierbar zu Deadlocks und irreparablen Datenbankkorruptionen.

Bereitstellungsmethoden im Vergleich: Docker Compose vs. Snap

Für die Bereitstellung von Immich existieren grundsätzlich zwei Ansätze: das offizielle Deployment via Docker Compose und inoffizielle Community-Pakete für das Canonical Snap-Format.

Warum Docker Compose der Standard ist

Das Kernentwicklerteam von Immich entwickelt, testet und veröffentlicht neue Versionen primär als Docker-Images. Docker Compose bietet im Betrieb handfeste Vorteile:

- **Volle Kontrolle über Hardware-Passthrough:** Intel QuickSync (`/dev/dri`) und Nvidia-Treiber für Transkodierung lassen sich transparent in den Server-Container durchreichen.
- **Freie Wahl der Speicherpfade:** Beliebige Host-Mounts und ZFS-Datasets können ohne Sandbox-Konflikte angebunden werden.
- **Gezielte Versionskontrolle:** Updates werden explizit vom Administrator ausgelöst, statt unkontrolliert im Hintergrund zu laufen.

- **Direkte Datenbankwartung:** Administrations- und Backup-Werkzeuge wie `pg_dumpall` können direkt über den Container ausgeführt werden.

Die Grenzen von Snap-Paketen

Im Snap Store existiert ein Community-Paket (`immich-distribution`). Auch wenn die Installation mit einem einzigen Befehl verlockend wirkt, birgt Snap in Produktionsumgebungen erhebliche Nachteile:

- **Strikte AppArmor-Confinement:** Snap isoliert Dienste in Sandbox-Profilen. Das Einbinden externer Festplatten oder separater Speicher-Mounts erfordert manuelle Schnittstellen-Freigaben (`removable-media`), die bei komplexen Speicher-Layouts häufig fehlschlagen.
- **Eingeschränkte GPU-Nutzung:** Der Zugriff auf Host-Grafikkarten für Hardware-Transkodierung ist über Snap fehleranfällig.
- **Gefahr durch automatische Hintergrund-Updates:** Snap aktualisiert installierte Pakete standardmäßig viermal täglich vollautomatisch. Bei komplexen Microservice-Stacks mit Schema-Migrationen in der Datenbank kann ein unvorbereitetes Update zu Ausfallzeiten führen.

Aus diesen Gründen ist Docker Compose die einzig verlässliche Wahl für den dauerhaften Betrieb.

Systemvorbereitung und Docker-Installation

Wir beginnen mit der Vorbereitung des Ubuntu 24.04 LTS Servers und der Installation der offiziellen Docker Engine inklusive Docker Compose Plugin.

System aktualisieren und Basiswerkzeuge einrichten

Aktualisiere die Paketquellen und installiere die notwendigen Hilfswerkzeuge:

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

Offizielles Docker-Repository einbinden

Ubuntu liefert im eigenen Repository oft ältere Docker-Pakete aus. Wir binden daher das offizielle Repository von Docker ein, um stets aktuelle Releases und Sicherheits-Patches zu erhalten:

```
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

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

Aktualisiere die Paketlisten erneut und installiere die Docker Engine samt Compose-Plugin:

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

Überprüfe nach Abschluss den Zustand des Docker-Dienstes:

```
sudo systemctl is-active docker
```

Ausgabe: `active`

Um Docker-Befehle als regulärer Benutzer ohne ständiges Voranstellen von `sudo` ausführen zu können, füge deinen Benutzer der Systemgruppe `docker` hinzu:

```
sudo usermod -aG docker $USER
```

> **Gruppenzugehörigkeit aktivieren:** Melde dich einmal von der SSH-Sitzung ab und wieder an (oder führe `newgrp docker` aus), damit die neue Gruppenberechtigung für deine aktuelle Shell wirksam wird.

Konfiguration des Docker-Compose-Stacks

Wir organisieren die Immich-Konfiguration in einem dedizierten Verzeichnis. Dies erleichtert spätere Backups, Updates und die Versionsverwaltung.

Projektverzeichnis und Quelldateien

Erstelle ein Arbeitsverzeichnis und lade die offiziellen Release-Dateien herunter:

```
mkdir -p ~/immich
cd ~/immich

wget -O docker-compose.yml https://github.com/immich-
app/immich/releases/latest/download/docker-compose.yml
wget -O .env https://github.com/immich-app/immich/releases/latest/download/example.env
```

Konfigurationsdatei `.env` anpassen

Die Datei `.env` definiert alle variablen Parameter des Stacks. Öffne die Datei mit einem Editor:

```
nano .env
```

Passe die folgenden Schlüsselwerte an deine Systemumgebung an:

```
# Speicherort für hochgeladene Originalfotos und generierte Thumbnails
UPLOAD_LOCATION=/srv/immich/library

# Speicherort für die PostgreSQL-Datenbank (muss zwingend lokal auf Flash-Speicher liegen)
DB_DATA_LOCATION=/srv/immich/postgres

# Zeitzone des Systems
TZ=Europe/Berlin

# Feste Versionsangabe oder Hauptversionszweig
IMMICH_VERSION=v3
```

```
# Datenbankpasswort: Nur alphanumerische Zeichen verwenden (A-Za-z0-9)
DB_PASSWORD=EinSicheresAlphanumerischesKennwort42

# Datenbank-Standardwerte (in der Regel belassen)
DB_USERNAME=postgres
DB_DATABASE_NAME=immich
```

> ⚠ **Sonderzeichen im Datenbankpasswort vermeiden:** Verwende im Parameter `DB_PASSWORD` ausschließlich Buchstaben und Zahlen (`[A-Za-z0-9]`). Sonderzeichen wie `@` , `:` , `/` oder Anführungszeichen führen bei der automatischen Generierung von Datenbank-Verbindungs-URIs im Server-Container regelmäßig zu Parsing-Fehlern.

Speicherverzeichnisse anlegen und absichern

Erstelle die in der `.env` definierten Verzeichnisse auf dem Host und sorge für die passenden Eigentumsrechte:

```
sudo mkdir -p /srv/immich/library /srv/immich/postgres
sudo chown -R $USER:$USER /srv/immich
```

Aufbau der Datei `docker-compose.yml`

Die heruntergeladene Compose-Datei bindet alle vier Microservices ein. Sie nutzt Umgebungsvariablen aus der `.env` und setzt standardmäßig die Neustart-Richtlinie `restart: always` .

```
name: immich

services:
  immich-server:
    container_name: immich_server
    image: ghcr.io/immich-app/immich-server:${IMMICH_VERSION:-release}
    volumes:
      - ${UPLOAD_LOCATION}:/data
      - /etc/localtime:/etc/localtime:ro
    env_file:
      - .env
    ports:
      - '2283:2283'
    depends_on:
      - redis
      - database
    restart: always
    healthcheck:
      disable: false

  immich-machine-learning:
    container_name: immich_machine_learning
    image: ghcr.io/immich-app/immich-machine-learning:${IMMICH_VERSION:-release}
    volumes:
      - model-cache:/cache
    env_file:
      - .env
    restart: always
    healthcheck:
      disable: false

  redis:
    container_name: immich_redis
    image:
docker.io/valkey/valkey:9@sha256:8e8d64b405ce18f41b8e5ee20aa4687a8ed0022d1298f2ce31cdcf3a7
6e09411
    healthcheck:
      test: redis-cli ping || exit 1
    restart: always
```

```
  database:
    container_name: immich_postgres
    image: ghcr.io/immich-app/postgres:14-vectorchord0.4.3-
pgvectors0.2.0@sha256:bcf63357191b76a916ae5eb93464d65c07511da41e3bf7a8416db519b40b1c23
    environment:
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_USER: ${DB_USERNAME}
      POSTGRES_DB: ${DB_DATABASE_NAME}
      POSTGRES_INITDB_ARGS: '--data-checksums'
    volumes:
      - ${DB_DATA_LOCATION}:/var/lib/postgresql/data
    shm_size: 128mb
    restart: always
    healthcheck:
      disable: false

volumes:
  model-cache:
```

Hardware-beschleunigte Transkodierung einbinden (optional)

Wenn dein Server über einen modernen Intel-Prozessor mit integrierter Grafikeinheit (QuickSync) oder eine AMD-APU verfügt, kannst du die Video-Transkodierung drastisch beschleunigen und die CPU entlasten.

Überprüfe, ob das Render-Device auf dem Host vorhanden ist:

```
ls -l /dev/dri
```

Wird `/dev/dri/renderD128` gelistet, reichst du das Verzeichnis in den `immich-server`-Container weiter, indem du die `devices`-Sektion in der `docker-compose.yml` ergänzt:

```
  immich-server:
    container_name: immich_server
    image: ghcr.io/immich-app/immich-server:${IMMICH_VERSION:-release}
    devices:
      - /dev/dri:/dev/dri
    volumes:
      - ${UPLOAD_LOCATION}:/data
      - /etc/localtime:/etc/localtime:ro
    env_file:
      - .env
    ports:
      - '2283:2283'
    depends_on:
      - redis
      - database
    restart: always
```

Damit greift FFmpeg innerhalb des Containers direkt auf die VA-API- oder QuickSync-Hardwarebeschleunigung zu.

Stack starten und Betriebsverifikation

Wechsle in das Verzeichnis `~/immich` und starte den gesamten Stack im Hintergrund:

```
docker compose up -d
```

Docker lädt nun die Container-Images herunter, initialisiert das Netzwerk `immich_default` und startet die vier Dienste in korrekter Abhängigkeitsreihenfolge.

Container-Status überprüfen

Prüfe nach etwa 30 Sekunden den Zustand aller Container:

```
docker compose ps
```

Erwartete Ausgabe:

```
NAME                      IMAGE
COMMAND                   SERVICE                   CREATED
STATUS                    PORTS
immich_machine_learning   ghcr.io/immich-app/immich-machine-learning:v3
"./entrypoint.sh"         immich-machine-learning   40 seconds ago   Up 38 seconds
(healthy)
immich_postgres           ghcr.io/immich-app/postgres:...                  "docker-
entrypoint.s…"   database                  40 seconds ago   Up 38 seconds (healthy)
5432/tcp
immich_redis              docker.io/valkey/valkey:9...                     "docker-
entrypoint.s…"   redis                     40 seconds ago   Up 39 seconds (healthy)
6379/tcp
immich_server             ghcr.io/immich-app/immich-server:v3
"./start.sh"              immich-server             40 seconds ago   Up 38 seconds
(healthy)   0.0.0.0:2283->2283/tcp
```

Alle vier Container müssen den Status `Up (healthy)` oder `Up` aufweisen.

Logdateien inspizieren

Sollte ein Dienst nicht starten, geben die Container-Logs sofort Aufschluss. Überprüfe die Initialisierung des Servers und der Datenbank:

```
docker compose logs --tail=50 immich-server
```

Suche in der Ausgabe nach Zeilen wie `[ImmichServer] Immich Server is listening on http://[::]:2283`. Dies signalisiert, dass alle Datenbankmigrationen erfolgreich abgeschlossen wurden.

> **Keine manuellen systemd-Wrapper nötig:** Die Direktive `restart: always` in der `docker-compose.yml` sorgt dafür, dass die Docker Engine die Container nach einem Serverneustart automatisch wieder hochfährt. Ein zusätzlicher systemd-Service für `docker compose up` ist überflüssig und führt beim Herunterfahren des Hosts häufig zu Race-Conditions.

Erstkonfiguration über die Weboberfläche

Sobald der Stack läuft, öffnest du im lokalen Webbrowser die IP-Adresse deines Servers auf Port 2283:

`http://192.168.1.100:2283` (ersetze die IP-Adresse durch die deines Hosts).

Administrator-Konto erstellen

Beim ersten Aufruf präsentiert Immich die Registrierungsseite für das Hauptkonto:

Registrierungsmaske für das initiale Administratorkonto in Immich

Gib eine gültige E-Mail-Adresse, deinen Namen und ein langes Kennwort ein. Dieses Konto besitzt volle administrative Rechte auf dem Server.

Quelle: https://admindocs.de/de/serverumgebungen/immich-private-foto-cloud-unter-ubuntu-24-04-lts-server-installieren

> ⚠ **Dediziertes Admin-Konto nutzen:** Aus Sicherheitsgründen solltest du das initiale Administratorkonto ausschließlich für Systemeinstellungen, Backups und Benutzerverwaltung verwenden. Erstelle für deine täglichen Foto-Uploads anschließend ein reguläres Benutzerkonto ohne globale Administrationsrechte.

Nach dem Klick auf **Konto erstellen** meldest du dich an der Weboberfläche an:

Anmeldemaske der Immich Weboberfläche

Der Schnelleinrichtungs-Assistent

Immich führt dich anschließend durch die Basiskonfiguration:

- **Erscheinungsbild:** Wähle zwischen hellem und dunklem Design.
- **Privatsphäre und externe Dienste:** Hier legst du fest, ob Immich Kartenkacheln für die Geo-Standortansicht (`tiles.immich.cloud`) laden und regelmäßig auf neue Software-Versionen prüfen darf. Für einen vollständig air-gapped Betrieb im LAN kannst du diese Optionen deaktivieren.

Speichervorlagen (Storage Templates)

Standardmäßig speichert Immich hochgeladene Dateien unter einer kryptischen Asset-ID. Wenn du möchtest, dass die Dateien im Host-Dateisystem in einer sauberen, lesbaren Verzeichnisstruktur abgelegt werden, aktivierst du die **Speichervorlagen-Engine** unter **Administration > Einstellungen > Speichervorlage**.

Die Engine arbeitet mit dynamischen Platzhaltern auf Basis der EXIF-Metadaten:

Platzhalter	Bedeutung	Beispielwert
`{{y}}`	Erstellungsjahr (vierstellig)	`2026`
`{{MM}}`	Monat (zweistellig mit führender Null)	`09`
`{{dd}}`	Tag (zweistellig mit führender Null)	`08`
`{{filename}}`	Originaler Dateiname ohne Pfad	`IMG_4021`

Platzhalter	Bedeutung	Beispielwert
`{{filetype}}`	Dateiendung bzw. Medientyp	`jpg`

Ein praxiserprobtes Muster für das Eingabefeld `Vorlage`:

```
{{y}}/{{y}}-{{MM}}-{{dd}}/{{filename}}
```

Dies ordnet Fotos auf der Festplatte beispielsweise unter `/srv/immich/library/admin/2026/2026-09-08/IMG_4021.jpg` an.

> **Dateipfad-Längen beachten:** Linux-Dateisysteme begrenzen Pfadlängen in der Regel auf 4096 Bytes und einzelne Verzeichnis- oder Dateinamen auf 255 Zeichen. Halte Speichervorlagen kompakt und verzichte auf übermäßig verschachtelte Ordnerstrukturen.

Mobile App und Synchronisation

Die mobilen Apps für Android und iOS sind das Herzstück von Immich für den automatischen Foto-Upload.

Kopplung und Ersteinrichtung

1. Installiere die offizielle App über Google Play oder den Apple App Store.
2. Trage die Server-Adresse ein (z. B. deine lokale LAN-IP oder die spätere HTTPS-Domain des Reverse-Proxys).
3. Melde dich mit deinen Benutzerdaten an.
4. Wähle die Geräteordner aus, die synchronisiert werden sollen (z. B. `DCIM/Camera`).

Besonderheiten bei Hintergrund-Uploads

Moderne mobile Betriebssysteme setzen strenge Energiesparmaßnahmen durch:

- **iOS:** Das System beendet Hintergrundaktivitäten restriktiv. Aktiviere in den iOS-Einstellungen unter `Immich` die Option **Hintergrundaktualisierung** und lasse die App bei

der ersten Massensynchronisation zehntausender Fotos über Nacht bei eingestecktem Ladekabel im Vordergrund geöffnet.

- **Android:** Schließe die Immich-App in den Android-Akkueinstellungen von den automatischen Batterieoptimierungen aus (`Nicht optimieren` bzw. `Uneingeschränkt`), damit der Upload-Dienst im Hintergrund nicht nach wenigen Minuten vom Kernel beendet wird.

Externe Bibliotheken (Read-Only Integration)

Wenn du bereits über eine bestehende Fotosammlung auf einem NAS oder einer Festplatte verfügst, musst du diese nicht duplizieren. Immich unterstützt das Einbinden als **Externe Bibliothek**.

Dazu bindest du das Quellverzeichnis in der `docker-compose.yml` unter `volumes` schreibgeschützt ein:

```
volumes:
  - ${UPLOAD_LOCATION}:/data
  - /mnt/nas_photos:/data/external_photos:ro
  - /etc/localtime:/etc/localtime:ro
```

Nach einem Neustart (`docker compose up -d`) navigierst du in der Weboberfläche auf **Administration > Bibliotheken**, erstellst eine neue externe Bibliothek und gibst den Pfad `/data/external_photos` an. Immich scannt die Metadaten, belässt die Originaldateien jedoch unberührt.

Produktionsabsicherung: Nginx Reverse Proxy mit HTTPS

Es ist grob fahrlässig, den Port `2283` direkt unverschlüsselt ins Internet freizugeben. Für den sicheren Fernzugriff schalten wir einen Nginx-Webserver als Reverse-Proxy davor, der TLS-Verschlüsselung erzwingt, große Datei-Uploads puffert und WebSocket-Verbindungen für Live-

Quelle: https://admindocs.de/de/serverumgebungen/immich-private-foto-cloud-unter-ubuntu-24-04-lts-server-installieren

Updates handhabt.

Nginx und Certbot installieren

```
sudo apt install -y nginx certbot python3-certbot-nginx
```

Virtuellen Host konfigurieren

Erstelle eine neue Konfigurationsdatei für deinen Immich-VHost:

```
sudo nano /etc/nginx/sites-available/immich.conf
```

Füge die folgende Konfiguration ein. Ersetze `photos.deine-domain.de` durch deinen tatsächlichen Domainnamen:

```
server {
    listen 80;
    server_name photos.deine-domain.de;

    # Maximale Upload-Größe für große 4K-Videos (50 GB)
    client_max_body_size 50000M;

    # Ausreichende Timeouts für langsame Upload-Verbindungen
    proxy_read_timeout 600s;
    proxy_send_timeout 600s;
    send_timeout 600s;
```

```
    # Pufferung deaktivieren, um RAM-Überlauf bei Video-Uploads zu verhindern
    proxy_request_buffering off;

    location / {
        proxy_pass http://127.0.0.1:2283;

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # WebSocket-Unterstützung für Echtzeit-Statusanzeigen
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_redirect off;
    }
}
```

Erklärung der zentralen Direktiven:

- **`client_max_body_size 50000M`**: Standardmäßig verwirft Nginx Requests über 1 MB mit dem Fehler `413 Request Entity Too Large`. Der Wert 50000M erlaubt den Upload bis zu 50 GB großer Videodateien.
- **`proxy_request_buffering off`**: Verhindert, dass Nginx eingehende Uploads zuerst vollständig auf der lokalen Systemfestplatte in temporäre Dateien schreibt. Die Daten werden stattdessen direkt an den Immich-Container durchgestreamt.
- **`Upgrade` und `Connection "upgrade"`**: Ermöglicht die HTTP-Upgrade-Verbindung für WebSockets. Ohne diese Header scheitern Live-Benachrichtigungen über den Verarbeitungsstatus in der Weboberfläche.

Aktiviere die Konfiguration und prüfe die Syntax:

```
sudo ln -s /etc/nginx/sites-available/immich.conf /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl reload nginx
```

TLS-Zertifikat mit Let's Encrypt beziehen

Sichere die Domain über Certbot mit einem kostenlosen TLS-Zertifikat ab:

```
sudo certbot --nginx -d photos.deine-domain.de
```

Certbot modifiziert die Konfigurationsdatei automatisch, erzwingt HTTPS und richtet die Erneuerung über systemd-Timer ein.

Firewall-Konfiguration mit UFW

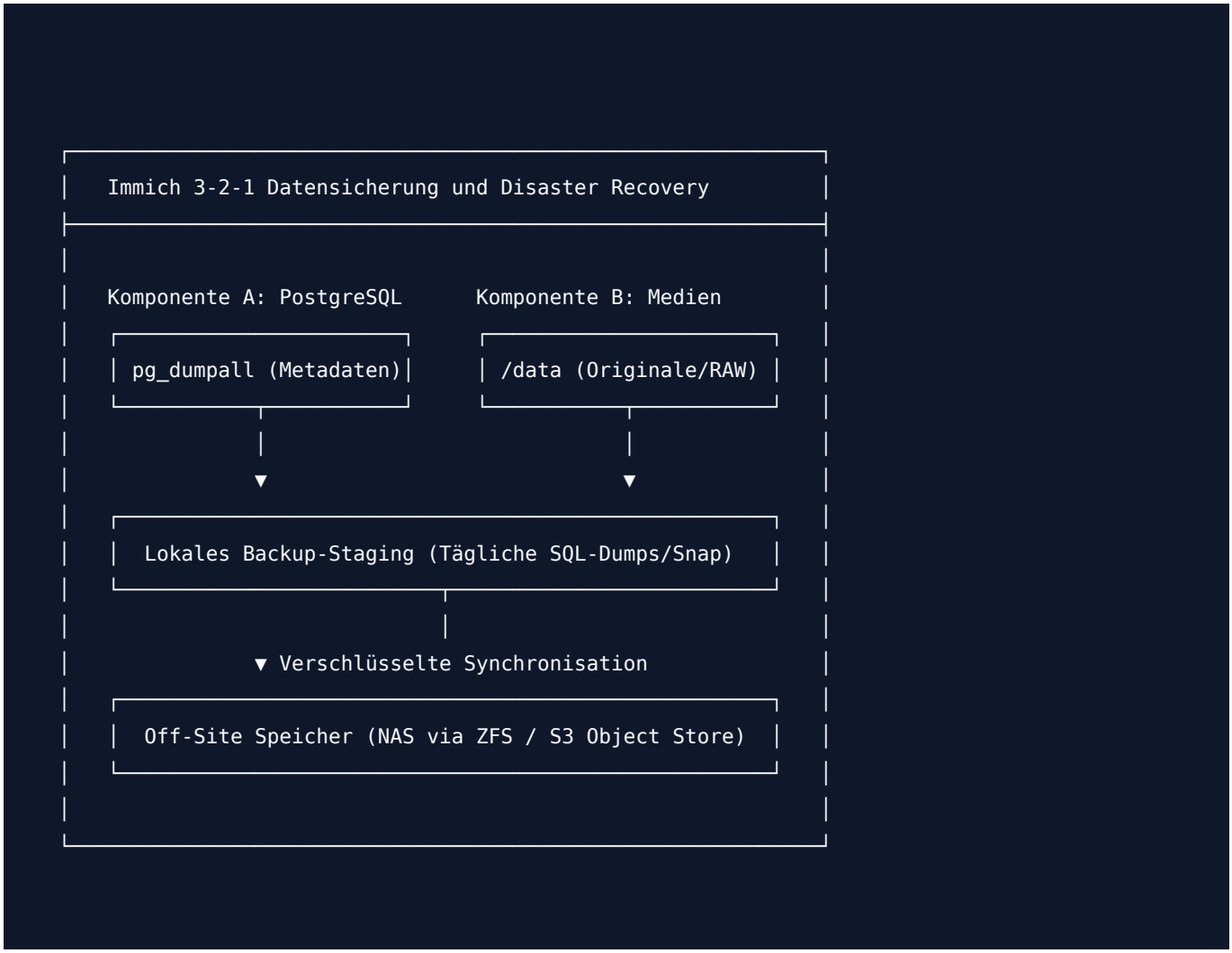
Begrenze den Zugriff auf den Server strikt. Bei Nutzung eines Reverse-Proxys darf Port 2283 von außen nicht mehr direkt erreichbar sein:

```
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow ssh
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
sudo ufw enable
sudo ufw reload
```

Backup- und Disaster-Recovery-Strategie

Eine Foto-Cloud ist nur so viel wert wie ihre Wiederherstellbarkeit nach einem Hardware-Defekt. Die Datensicherung von Immich gliedert sich zwingend in zwei separate Teile:

1. **Die relationale Datenbank:** Enthält Alben, Gesichtsdaten, Benutzerzuordnungen, Zeitachsen-Zuordnungen und Vektoren. Ohne diesen Zustand ist die Dateistruktur unvollständig.
2. **Die Medienbibliothek:** Das Verzeichnis mit den tatsächlichen Bild- und Videodateien.



Datenbank-Backup per CLI

Immich bietet zwar in der Weboberfläche unter **Administration > Einstellungen > Backup** einen internen Dump-Mechanismus, auf Betriebssystem-Ebene solltest du diesen jedoch über ein automatisiertes Skript absichern.

Der zuverlässigste Weg für einen vollständigen PostgreSQL-Dump im laufenden Betrieb:

```
docker exec -t immich_postgres pg_dumpall --clean --if-exists --username=postgres | gzip > /srv/backups/immich_db_$(date +%Y%m%d_%H%M%S).sql.gz
```

> **Konsistenter Dump-Zustand:** Für maximale Konsistenz vor großen Versionsupgrades kannst du den Server-Container kurz anhalten (`docker compose stop immich-server`), den Datenbank-Dump ausführen und den Server anschließend wieder starten (`docker compose start immich-server`).

Sicherung der Medienbibliothek

Das Medienverzeichnis `/srv/immich/library` sicherst du mit Werkzeugen wie `rsync` , Restic oder BorgBackup auf ein separates Speichersystem:

```
rsync -aAXv --delete /srv/immich/library/ /mnt/backup_storage/immich_library/
```

Wende die bewährte **3-2-1-Regel** an:

- **3** Kopien der Daten (Produktivsystem, lokales Backup, externes Archiv)
- **2** unterschiedliche Speichermedien (z. B. NVMe-SSD im Server, ZFS-Pool auf dem Backup-NAS)
- **1** Kopie an einem geographisch getrennten Standort (z. B. verschlüsselter Cloud-Storage)

Disaster Recovery: Das Wiederherstellungsszenario

Tritt ein Totalausfall ein oder ziehst du auf einen neuen Server um, verfährst du nach dieser Sequenz:

1. Richte den neuen Server mit Ubuntu 24.04 und Docker ein.
2. Synchronisiere die gesicherte Medienbibliothek an ihren ursprünglichen Pfad `/srv/immich/library`.
3. Platziere `docker-compose.yml` und `.env` im Projektordner.
4. Starte ausschließlich den Datenbank-Container:

```
docker compose up -d database
```

5. Spiele den Datenbank-Dump in die neu initialisierte PostgreSQL-Instanz ein:

```
gunzip -c /srv/backups/immich_db_backup.sql.gz | docker exec -i immich_postgres psql -U postgres -d immich
```

6. Starte den verbleibenden Stack:

```
docker compose up -d
```

7. Überprüfe die Logs: Der Immich-Server erkennt die eingespielten Metadaten und bindet die vorhandenen Medien nahtlos ein.

Wartung, Updates und Betriebsüberwachung

Immich unterliegt einer kontinuierlichen Weiterentwicklung. Bei Versionsupgrades ist strukturiertes Vorgehen Pflicht, um Inkompatibilitäten zu vermeiden.

Upgrade-Disziplin

Lies vor jedem Versionssprung zwingend die offiziellen Release Notes auf GitHub. Größere Versionsübergänge (z. B. von v1.x auf v2.x oder v2.x auf v3.x) enthalten gelegentlich vorbereitende Migrationsschritte.

Führe das Update mit folgender Befehlskette durch:

```
cd ~/immich

# 1. Sicherheits-Dump der Datenbank ziehen
docker exec -t immich_postgres pg_dumpall --clean --if-exists --username=postgres | gzip >
~/immich_pre_upgrade.sql.gz

# 2. Neue Container-Images herunterladen
docker compose pull

# 3. Stack mit den neuen Images neu starten
docker compose down
docker compose up -d

# 4. Datenbankmigration im Log beobachten
docker compose logs -f immich-server
```

Gezieltes Versions-Pinning

Statt immer die Fließmarke `:release` oder `:v3` zu verwenden, kannst du in der `.env`-Datei eine konkrete Release-Version festzurren:

```
IMMICH_VERSION=v3.0.1
```

Dies verhindert, dass ein versehentliches `docker compose pull` unerprobte Änderungen auf dein Produktivsystem bringt.

Wartungsjobs in der Weboberfläche

Unter **Administration > Jobs** stellt Immich automatisierte Wartungsworkflows bereit:

- **Gesichter erkennen / neu scannen:** Sinnvoll nach Updates des Machine-Learning-Containers, um verbesserte Erkennungsmodelle auf bestehende Gesichter anzuwenden.
- **Thumbnails generieren:** Repariert fehlende Vorschauen bei defekten Cache-Volumes.
- **Bibliothek scannen:** Findet manuell im Speicherordner abgelegte oder verschobene Bilddateien.

Fehlerbehebung (Troubleshooting)

Wenn Dienste streiken oder Uploads abbrechen, grenzt du den Fehler systematisch anhand der Systemkomponenten ein.

Weboberfläche antwortet nicht (502 Bad Gateway)

Antwortet Nginx mit `502 Bad Gateway`, läuft der Upstream-Dienst `immich-server` nicht oder horcht nicht auf Port 2283.

Diagnose:

```
docker compose ps
docker compose logs --tail=100 immich-server
```

Mögliche Ursachen:

- **Datenbankverbindung fehlgeschlagen:** Prüfe, ob der Container `immich_postgres` gesund (`healthy`) ist und ob das Passwort in der `.env` exakt mit den Initialwerten der Datenbank übereinstimmt.
- **Port-Konflikt auf dem Host:** Prüfe mit `ss -tulpn | grep 2283`, ob ein anderer Prozess Port 2283 belegt.

Uploads brechen mit HTTP 413 oder 504 ab

Tritt der Fehler `413 Request Entity Too Large` auf, blockiert der Nginx-Reverse-Proxy den Request, weil `client_max_body_size` zu niedrig eingestellt ist.

Tritt bei sehr großen Videos ein `504 Gateway Timeout` auf, haben die Direktiven `proxy_read_timeout` oder `proxy_send_timeout` im Nginx gegriffen.

Korrektur:

Stelle sicher, dass in der Nginx-VHost-Konfiguration `client_max_body_size 50000M;` und Timeouts von mindestens `600s` hinterlegt sind, und lade Nginx neu:

```
sudo nginx -t && sudo systemctl reload nginx
```

Machine-Learning-Container crasht beim Start

Stürzt der Container `immich_machine_learning` nach dem Start ab, liegt dies meist an fehlenden CPU-Befehlssätzen oder unzureichendem RAM.

Diagnose:

```
docker compose logs immich-machine-learning
```

Erscheint im Log `Illegal instruction (core dumped)`, unterstützt deine CPU keine AVX-Instruktionen.

Abhilfe:

- Stelle in virtuellen Umgebungen (KVM / Proxmox) den CPU-Typ auf `host`.
- Bei Systemen ohne physische AVX-Unterstützung muss in der Compose-Konfiguration ein alternatives Machine-Learning-Modell konfiguriert werden, das rein auf Standard-Gleitkomma-Operationen zurückgreift.

Fehlerhafte Dateiberechtigungen

Schlagen Uploads mit `EACCES: permission denied` fehl, fehlen dem Container die Schreibrechte auf dem Host-Verzeichnis `/srv/immich/library`.

Korrektur:

```
sudo chown -R 1000:1000 /srv/immich/library
sudo chmod -R 750 /srv/immich/library
```

Befehlsreferenz (Cheatsheet)

Die wichtigsten Befehle für die tägliche Administration und Wartung von Immich zusammengefasst:

Befehl	Zweck	Kontext
`docker compose up -d`	Gesamten Stack im Hintergrund starten	Projektordner `~/immich`
`docker compose down`	Alle Container kontrolliert stoppen und entfernen	Projektordner `~/immich`
`docker compose ps`	Status und Healthchecks aller Dienste anzeigen	Diagnose
`docker compose logs -f --tail=50 immich-server`	Live-Logs des Anwendungs-Servers verfolgen	Fehleranalyse
`docker compose logs -f database`	Live-Logs der PostgreSQL-Datenbank prüfen	Fehleranalyse
`docker compose restart immich-server`	Anwendungs-Server isoliert neu starten	Konfigurationsänderung
`docker exec -t immich_postgres pg_dumpall --clean --if-exists -U postgres	gzip > backup.sql.gz`	Vollständigen SQL-Datenbankdump erstellen	Datensicherung
`gunzip -c backup.sql.gz	docker exec -i immich_postgres psql -U postgres -d immich`	SQL-Dump in die Datenbank wiederherstellen	Disaster Recovery
`docker compose pull && docker compose up -d`	Neue Container-Images laden und aktualisieren	Update-Workflow
`docker system prune -f`	Nicht mehr genutzte alte Container-Images bereinigen	Festplattenpflege

Weiterführende Ressourcen

Zentrale Dokumentationen, Quellcode-Repositories und Schnittstellen für den Betrieb von Immich:

Ressource	Beschreibung	Typ
Offizielle Immich Dokumentation	Referenz für Konfigurationsparameter und Umgebungsvariablen	Dokumentation
Immich GitHub Repository	Quellcode, Bugtracker, Diskussionen und Release-Notes	Quellcode
Hardware Transcoding Guide	Offizielle Konfiguration für Intel QuickSync, VA-API und Nvidia NVENC	Konfiguration
Immich Backup & Restore Guide	Best Practices für konsistente Datenbank- und Datei-Backups	Dokumentation
Docker Engine Dokumentation	Offizielles Handbuch zur Administration von Containern unter Linux	Referenz

Fazit

Mit dieser Bereitstellung betreibst du eine vollwertige, private Foto- und Video-Cloud unter eigener Kontrolle auf deinem Ubuntu 24.04 LTS Server. Die Aufteilung in spezialisierte Microservices für Serverlogik, Vektordatenbank, In-Memory-Queues und Machine-Learning-Inferenz garantiert hohe Performance und Skalierbarkeit auch bei großen Medienbeständen.

Durch das Zusammenspiel aus Nginx als gehärtetem Reverse-Proxy, angepassten Upload- und WebSocket-Puffern, Hardware-beschleunigtem Transcoding und automatisierten PostgreSQL-Dumps steht dein Setup auf einem stabilen Produktionsfundament.

> **Praxistipp für den Produktivbetrieb:** Halte die Versions-Upgrades von Immich engmaschig nach. Da sich das Projekt zügig weiterentwickelt, sind kleine, regelmäßige Versionssprünge mit vorherigem Datenbank-Dump operativ deutlich risikoärmer als ein einzelnes Riesen-Upgrade nach zwölf Monaten.

Quelle: https://admindocs.de/de/serverumgebungen/immich-private-foto-cloud-unter-ubuntu-24-04-lts-server-installieren