Flickr zu Immich Migration

Vollständige Anleitung mit Docker/Podman

Plattformübergreifend: Linux, Mac, Windows

Plattform	Browser	Unterstützung
Linux	X11-Forwarding (automatisch)	Voll unterstützt
Linux	Domain Socket (`USE_DSOCKET`)	Voll unterstützt
Linux	D-Bus (`USE_DBUS`)	Voll unterstützt (Podman empfohlen)
Mac	URL manuell öffnen	Unterstützt
Windows (WSL2)	URL manuell öffnen	Unterstützt

Teil 1 - Grundlagen

1. Übersicht

Diese Anleitung beschreibt die Migration einer Flickr-Fotobibliothek zu Immich unter Verwendung der Flickr-API. Der Prozess läuft in einem Docker/Podman-Container und funktioniert auf Linux, Mac und Windows.

Workflow:

- **flickr_download** - Fotos und Videos via API herunterladen (Metadaten werden automatisch eingebettet)
- **Immich CLI** - Upload zu Immich (integriert im Container)

  Hinweis: Das Script erkennt automatisch das Betriebssystem und die Container-Runtime (Docker/Podman) und passt sich entsprechend an. Ein manuelles Metadaten-Embedding-Script ist **nicht mehr nötig** - die Metadaten werden beim Download über das `--metadata_store` Flag automatisch verarbeitet.

2. Voraussetzungen

Komponente	Linux	Mac	Windows
Container Runtime	Docker oder Podman	Docker Desktop	Docker Desktop / WSL2
X11	Ja (`xauth`)	Nicht nötig	Nicht nötig
Shell	Bash (nativ)	Bash (nativ)	WSL2 oder Git Bash

3. Flickr API-Key erstellen

1. Gehe zu: https://www.flickr.com/services/apps/create/
2. Klicke auf “Request an API Key” -> “Non-Commercial”
3. Fülle das Formular aus (Name: z.B. “Flickr Backup”)
4. Notiere **API Key** und **API Secret**

4. Einrichtung

Option A: Docker Hub Image verwenden (empfohlen) Das fertige Image kann direkt von Docker Hub bezogen werden:

```
docker pull xomoxcc/flickr-download:latest
```

Alternativ mit explizitem Tag:

```
docker pull xomoxcc/flickr-download:python-3.14-slim-trixie
```

Option B: Lokal bauen

```
# Repository klonen
git clone https://github.com/<user>/flickrtoimmich.git
cd flickrtoimmich

# Image lokal bauen
make build
```

API-Konfiguration erstellen

```
mkdir -p flickr-config
cat > flickr-config/.flickr_download << EOF
api_key: DEIN_API_KEY
api_secret: DEIN_API_SECRET
EOF
```

System-Info anzeigen

```
./flickr-docker.sh info
```

5. Authentifizierung

Linux (X11 - Standard): Unter Linux öffnet sich automatisch ein Browser-Fenster im Container für die OAuth-Authentifizierung.

```
# Standard (Chrome)
./flickr-docker.sh auth

# Mit Firefox
BROWSER=firefox ./flickr-docker.sh auth
```

Mac / Windows: Auf Mac und Windows wird die OAuth-URL im Terminal angezeigt. Diese URL muss manuell im Browser geöffnet werden.

```
./flickr-docker.sh auth

# Ausgabe:
# HINWEIS für Mac/Windows:
#   - Eine URL wird im Terminal angezeigt
```

```
#   - Öffne diese URL manuell in deinem Browser
#   - Nach dem Login wird der Callback automatisch verarbeitet
```

Wichtig: Nach erfolgreichem Login wird das Token in `flickr-config/.flickr_token` gespeichert.

Netzwerk: Unter Linux wird `--network=host` verwendet. Auf Mac/Windows werden die Ports 8080-8100 für den OAuth-Callback-Server automatisch veröffentlicht.

6. Alben auflisten

```
./flickr-docker.sh list dein_flickr_username
```

Die Ausgabe zeigt Album-ID, Titel, Foto- und Video-Anzahl pro Album.

7. Fotos herunterladen

Alle Alben eines Benutzers:

```
# Mit Benutzername
./flickr-docker.sh download dein_flickr_username

# Oder mit voller URL
./flickr-docker.sh download https://www.flickr.com/photos/username/
```

Einzelnes Album:

```
./flickr-docker.sh album 72157622764287329
```

Dry-Run: Vorschau ohne Download Mit `--dry-run` verbindet sich das Script mit der Flickr-API und listet auf, was heruntergeladen werden würde — ohne Dateien herunterzuladen oder Verzeichnisse zu erstellen.

```
# Alle Alben mit Foto/Video-Counts anzeigen
./flickr-docker.sh download dein_flickr_username --dry-run

# Zusätzlich einzelne Fotos pro Album auflisten
./flickr-docker.sh download dein_flickr_username --dry-run --verbose

# Einzelnes Album: Fotos auflisten
./flickr-docker.sh album 72157622764287329 --dry-run
```

Hinweis: Im `download --dry-run`-Modus werden standardmäßig nur Album-Counts angezeigt (wenige API-Aufrufe). Mit `--verbose` (oder `-v`) werden zusätzlich alle Fotos/Videos pro Album aufgelistet. Im `album --dry-run`-Modus werden einzelne Fotos immer aufgelistet.

Die Fotos werden in `./flickr-backup/` gespeichert, zusammen mit JSON-Metadaten.

Metadaten: Die Flickr-Metadaten (Datum, Titel, Tags) werden beim Download **automatisch** in die Bilddateien eingebettet. Ein separates Metadaten-Script ist nicht mehr erforderlich.

Resumable Downloads: API-Antworten werden in `./flickr-cache/` gecacht. Bei Unterbrechung kann der Download einfach erneut gestartet werden - bereits heruntergeladene Dateien werden übersprungen.

Datumsfix: Fotos mit ungültigem Datum ("0000-00-00 00:00:00") werden automatisch behandelt und führen nicht mehr zu Abbrüchen.

8. Upload zu Immich

Die Immich CLI (`@immich/cli`) ist bereits im Container vorinstalliert. Es gibt zwei Wege für den Upload:

Option A: Separater Upload nach dem Download

```
# Umgebungsvariablen setzen
export IMMICH_INSTANCE_URL=https://immich.example.com
export IMMICH_API_KEY=dein_api_key

# Upload starten
./upload-to-immich.sh
```

Das Script erkennt automatisch, ob es im Container oder auf dem Host läuft: - **Im Container:** Führt den Upload direkt aus - **Auf dem Host:** Startet automatisch einen passenden Container für den Upload

Jedes Unterverzeichnis in `flickr-backup/` wird als eigenes Album in Immich angelegt.

Option B: Download und Upload in einem Schritt Über den `download_then_upload` Entrypoint können Download und Upload kombiniert werden:

```
docker run --rm \
  -e DATA_DIR=/home/poduser/flickr-backup \
  -e IMMICH_API_KEY=dein_api_key \
  -e IMMICH_INSTANCE_URL=https://immich.example.com \
  -v ./flickr-config:/root \
  -v ./flickr-backup:/root/flickr-backup \
  -v ./flickr-cache:/root/flickr-cache \
  xomoxcc/flickr-download:latest \
  download_then_upload dein_flickr_username
```

Der `download_then_upload` Entrypoint unterstützt ebenfalls `--dry-run` und `--verbose`:

```
docker run --rm \
  -e DATA_DIR=/home/poduser/flickr-backup \
  -e IMMICH_API_KEY=dein_api_key \
  -e IMMICH_INSTANCE_URL=https://immich.example.com \
  -v ./flickr-config:/root \
  -v ./flickr-backup:/root/flickr-backup \
  xomoxcc/flickr-download:latest \
  --dry-run --verbose download_then_upload dein_flickr_username
```

Im Dry-Run-Modus werden sowohl der Download- als auch der Upload-Dry-Run ausgeführt (Alben/Fotos auflisten, Dateien ohne Upload listen).

9. Befehlsreferenz

Befehl	Beschreibung
`auth`	OAuth-Authentifizierung starten
`download <user>`	Alle Alben eines Users herunterladen
`download <user> --dry-run`	Alben und Foto/Video-Counts auflisten (ohne Download)
`album <id>`	Einzelnes Album herunterladen
`album <id> --dry-run`	Fotos in einem Album auflisten (ohne Download)
`list <user>`	Alben eines Users auflisten
`shell`	Interaktive Shell im Container
`test-browser [url]`	X11/Browser-Verbindung testen
`info`	System-Informationen anzeigen
`clean`	Image und temp. Dateien löschen
`help`	Hilfe anzeigen

Optionen:

Option	Beschreibung
`--dry-run`	Vorschau: Alben/Fotos via API auflisten, ohne herunterzuladen
`--verbose`, `-v`	Im Dry-Run-Modus: einzelne Fotos pro Album auflisten

10. Verzeichnisstruktur

Verzeichnis	Inhalt
`./flickr-config/`	API-Keys (`.flickr_download`) und Token (`.flickr_token`)
`./flickr-backup/`	Heruntergeladene Fotos/Videos und JSON-Metadaten
`./flickr-cache/`	API-Cache für Resume-Funktion

Die Verzeichnisse werden als Volumes in den Container gemountet:

Host-Pfad	Docker-Container	Podman-Container
`./flickr-config/`	`/root`	`/home/poduser`
`./flickr-backup/`	`/root/flickr-backup`	`/home/poduser/flickr-backup`
`./flickr-cache/`	`/root/flickr-cache`	`/home/poduser/flickr-cache`

11. Fehlerbehebung

Browser öffnet nicht (Linux):

```
# DISPLAY prüfen
echo $DISPLAY

# xauth prüfen
xauth list

# Browser-Test im Container
./flickr-docker.sh test-browser
```

Token ungültig:

```
# Token löschen und neu authentifizieren
rm flickr-config/.flickr_token
./flickr-docker.sh auth
```

Podman-Probleme: Das Script erkennt Podman automatisch und verwendet: - `--userns=keep-id` (für X11-Zugriff und korrekte Dateirechte) - `--security-opt label=disable` (für SELinux-Kompatibilität)

Mac/Windows OAuth-Callback: Falls der OAuth-Callback nicht funktioniert, prüfe ob die Ports 8080-8100 frei sind. Das Script veröffentlicht diese Ports automatisch für den Callback-Server.

Rate Limiting (HTTP 429): Bei API-Rate-Limits pausiert das Script den Download-Prozess automatisch und wartet mit exponentiellem Backoff. Details dazu im Fortgeschrittenen-Abschnitt (Kapitel 13).

Teil 2 - Fortgeschritten

12. Alternative Authentifizierungs-Modi

Neben dem Standard-X11-Modus gibt es zwei weitere Möglichkeiten für die Browser-Authentifizierung in Container-Umgebungen.

Domain Socket Modus (`USE_DSOCKET`) Verwendet einen Unix-Socket, um URLs vom Container zum Host weiterzuleiten. Auf dem Host wird ein Python-Listener gestartet, der die URL mit `xdg-open` öffnet.

```
USE_DSOCKET=true ./flickr-docker.sh auth
```

Konfigurierbare Variablen:

Variable	Default	Beschreibung
`USE_DSOCKET`	`false`	Domain-Socket-Modus aktivieren

Variable	Default	Beschreibung
DSOCKET_PATH	/tmp/.flickr-open-url.sock	Socket-Pfad auf dem Host
DSOCKET_CONTAINER_PATH	/tmp/open-url.sock	Socket-Pfad im Container

D-Bus Modus (USE_DBUS) Verwendet das XDG Desktop Portal über D-Bus, um URLs über den System-Browser zu öffnen.

```
USE_DBUS=true ./flickr-docker.sh auth
```

> **Hinweis:** Der D-Bus-Modus funktioniert am zuverlässigsten mit **Podman**, da Docker aufgrund von UID-Unterschieden Probleme mit der D-Bus-Authentifizierung haben kann.

Variable	Default	Beschreibung
USE_DBUS	false	D-Bus-Modus aktivieren
DBUS_SESSION_BUS_ADDRESS	(vom System)	D-Bus Session-Adresse

Modus-Übersicht

Modus	Voraussetzung	Empfohlen für
X11 (Standard)	DISPLAY gesetzt, xauth	Desktop-Linux mit X11
Domain Socket	Python 3 auf dem Host	Headless-Server, SSH
D-Bus	D-Bus Session, Podman empfohlen	Wayland, moderne Desktops

> **Wichtig:** Die drei Modi sind **gegenseitig ausschließend**. Es darf nur einer gleichzeitig aktiviert sein.

13. Rate Limiting und Backoff-Konfiguration

Die Flickr-API hat Rate Limits. Bei HTTP 429-Antworten wird der Download-Prozess automatisch mit SIGSTOP/SIGCONT-Signalen pausiert und mit exponentiellem Backoff fortgesetzt.

Variable	Default	Beschreibung
BACKOFF_BASE	60	Basis-Wartezeit in Sekunden
BACKOFF_MAX	600	Maximale Wartezeit in Sekunden
BACKOFF_EXIT_ON_429	false	Bei Rate Limit sofort beenden (Exit Code 42) statt warten

Berechnung: Wartezeit = `BACKOFF_BASE * Anzahl_aufeinanderfolgender_429` (begrenzt durch `BACKOFF_MAX`)

```
# Beispiel: Kürzere Wartezeiten
BACKOFF_BASE=30 BACKOFF_MAX=300 ./flickr-docker.sh download username

# Für CI/Kubernetes: Sofort beenden statt warten
BACKOFF_EXIT_ON_429=true ./flickr-docker.sh download username
```

14. Kubernetes Deployment

Für den automatisierten Betrieb steht ein Ansible-basiertes Kubernetes-Deployment zur Verfügung.

Architektur

- **Namespace:** `flickr-downloader`
- **Jobs:** Pro Flickr-User ein separater Kubernetes-Job
- **Operator:** Automatischer Restart-Controller (`flickr-immich-k8s-sync-operator`)
- **Entrypoint:** `download_then_upload` (Download + Immich Upload in einem Schritt)

Job-Konfiguration Jeder Job wird mit folgenden Einstellungen erstellt:

```
command: ["/entrypoint.sh", "download_then_upload", "<flickr-user>"]
env:
  - DATA_DIR: /home/poduser/flickr-backup
  - IMMICH_API_KEY: <api-key>
  - IMMICH_INSTANCE_URL: <immich-url>
  - BACKOFF_EXIT_ON_429: "true"    # Sofort beenden bei Rate Limit
  - HOME: /home/poduser
  - TZ: Europe/Berlin
resources:
  requests: { cpu: 250m, memory: 1Gi }
  limits:   { cpu: 2000m, memory: 2Gi }
```

Operator-Konfiguration Der Operator überwacht Jobs und startet sie bei Bedarf neu:

Variable	Default	Beschreibung
`NAMESPACE`	`flickr-downloader`	Kubernetes-Namespace
`JOB_NAMES`	-	Komma-getrennte Liste der Job-Namen
`CHECK_INTERVAL`	60	Prüfintervall in Sekunden
`RESTART_DELAY`	3600	Wartezeit vor Neustart in Sekunden
`SKIP_DELAY_ON_OOM`	`true`	Bei OOM sofort neu starten

Volume-Mounts (hostPath)

```
/home/poduser              <- <prefix>/<flickr-user>/flickr-config
/home/poduser/flickr-backup <- <prefix>/<flickr-user>/flickr-backup
/home/poduser/flickr-cache  <- <prefix>/<flickr-user>/flickr-cache
```

15. Podman-Besonderheiten

Das Script erkennt Podman automatisch und wendet folgende Anpassungen an:

Einstellung	Wert	Grund
`--userns=keep-id`	(Linux)	Bewahrt User-Namespace für X11 und D-Bus

Einstellung	Wert	Grund
`--security-opt label=disable`	(Linux)	SELinux-Kompatibilität
Home-Verzeichnis	`/home/poduser`	Statt `/root` (rootless Container)

Multi-Arch Build mit Podman

```
# Lokal bauen (ohne Push)
./repo_scripts/build-container-multiarch.sh onlylocal

# Bauen und zu Docker Hub pushen
./repo_scripts/build-container-multiarch.sh
```

Podman erstellt ein Manifest mit separaten Builds pro Plattform (amd64 + arm64) und nutzt VM-Emulation für Cross-Compilation.

16. Alle Umgebungsvariablen

Container und Pfade

Variable	Default	Beschreibung
`IMAGE_NAME`	`flickr-download`	Name des Docker-Images
`IMAGE_TAG`	`latest`	Tag des Docker-Images
`WORK_DIR`	`$(pwd)/flickr-backup`	Arbeitsverzeichnis für Downloads
`CONFIG_DIR`	`$(pwd)/flickr-config`	Verzeichnis für API-Keys und Token
`CACHE_DIR`	`$(pwd)/flickr-cache`	Verzeichnis für API-Cache
`FLICKR_HOME`	-	Override für Home-Verzeichnis (Container-Erkennung)
`DATA_DIR`	`$(pwd)/flickr-backup`	Datenverzeichnis für Immich-Upload

Browser und Authentifizierung

Variable	Default	Beschreibung
`BROWSER`	`chrome` (Linux) / leer (Mac/Win)	Browser für OAuth
`DISPLAY`	(vom System)	X11 Display
`XAUTHORITY`	-	X11 Auth-Datei
`XAUTH_FILE`	`/tmp/.flickr-docker.xauth`	Temporäre xauth-Datei
`USE_DSOCKET`	`false`	Domain-Socket-Modus
`DSOCKET_PATH`	`/tmp/.flickr-open-url.sock`	Socket-Pfad (Host)
`DSOCKET_CONTAINER_PATH`	`/tmp/open-url.sock`	Socket-Pfad (Container)
`USE_DBUS`	`false`	D-Bus-Modus

Variable	Default	Beschreibung
DBUS_SESSION_BUS_ADDRESS	(vom System)	D-Bus Session-Adresse

Rate Limiting

Variable	Default	Beschreibung
BACKOFF_BASE	60	Basis-Wartezeit (Sekunden)
BACKOFF_MAX	600	Maximale Wartezeit (Sekunden)
BACKOFF_EXIT_ON_429	false	Bei Rate Limit sofort beenden (Exit Code 42)

Immich Upload

Variable	Default	Beschreibung
IMMICH_INSTANCE_URL	-	URL der Immich-Instanz (erforderlich)
IMMICH_API_KEY	-	API-Key für Immich (erforderlich)

Kubernetes / CI

Variable	Default	Beschreibung
KUBERNETES_SERVICE_HOST	(vom System)	Kubernetes-Pod-Erkennung
TZ	-	Zeitzone (z.B. Europe/Berlin)

Build

Variable	Default	Beschreibung
BUILDTIME	-	Build-Zeitstempel
PYTHONUNBUFFERED	1	Python-Output nicht puffern