

Psono Passwortmanager

Migration auf einen neuen Host — Schritt-für-Schritt-Anleitung

Stand: Juli 2026 · Psono Community Edition · Debian 12/13 · PostgreSQL 17 · Nginx · Dehydrated · nftables

Platzhalter in dieser Anleitung

Ersetze alle Platzhalter vor dem ersten Schritt durch deine eigenen Werte:

Platzhalter	Bedeutung	Beispiel
DEINE_DOMAIN	Öffentliche Domain von Psono	psono.example.com
DEIN_SERVER_HOSTNAME	Hostname des neuen Servers	server.example.com
DEINE_SERVER_IP	Öffentliche IPv4 des neuen Servers	203.0.113.42
DEIN_DB_PASSWORT	Passwort für PostgreSQL-Benutzer psono	sicheres-passwort-hier
DEINE_EMAIL	E-Mail für Psono-Benachrichtigungen	admin@example.com
BACKUP_DATEI.tar	Dateiname des Datenbank-Backups	2026-06-13_00-00.tar

Überblick & Voraussetzungen

Diese Anleitung führt durch die komplette Migration von Psono von einem alten Host auf einen neuen Debian-Server.

Was bereits vorhanden sein muss

- ✓ Debian 12 oder 13 mit root-Zugriff auf dem neuen Host
- ✓ Docker installiert (empfohlen: download.docker.com, trixie stable)
- ✓ PostgreSQL 17 installiert
- ✓ Nginx installiert
- ✓ nftables installiert (auf Debian 13 standardmäßig vorhanden)
- ✓ Dehydrated installiert mit gültigem Zertifikat für DEINE_DOMAIN
- ✓ Datenbankbackup vom alten System (.tar, PostgreSQL Custom Format)
- ✓ settings.yaml gesichert vom alten psono-combo Container
- ✓ DNS-Eintrag für DEINE_DOMAIN zeigt auf DEINE_SERVER_IP

🔑 Wichtig: Kryptographische Schlüssel

Die settings.yaml vom alten System enthält SECRET_KEY, PRIVATE_KEY und PUBLIC_KEY. Diese müssen 1:1 übernommen werden — neue Schlüssel würden bedeuten dass alle verschlüsselten Passwörter verloren sind.

Übersicht der Phasen

1

PostgreSQL vorbereiten

Datenbank, Benutzer, Erweiterungen und Netzwerkkonfiguration

2

Backup einspielen

Den Datenbank-Dump vom alten System importieren

3

Psono-Konfiguration wiederherstellen

settings.yaml und config.json einrichten

4

Psono-Container starten

Docker Compose einrichten und Container starten

5

Nginx Reverse Proxy einrichten

HTTPS-Konfiguration mit Dehydrated-Zertifikat

6

Firewall einrichten

Port 5432 mit nftables von außen absichern

7

Testen & umschalten

Funktionstest, DNS umstellen, altes System abschalten

Phase 1 — PostgreSQL vorbereiten

PostgreSQL 17 ist bereits installiert. Wir legen die Datenbank und den Psono-Benutzer an, aktivieren die benötigten Erweiterungen und konfigurieren die Netzwerkzugänge für Docker.

Schritt 1.1 — Als postgres-Benutzer anmelden

```
sudo -i -u postgres
```

Erwarteter Prompt: postgres@DEIN_SERVER_HOSTNAME:~\$

Schritt 1.2 — PostgreSQL-Konsole öffnen

```
psql
```

Erwarteter Prompt: postgres=#

Schritt 1.3 — Datenbank, Benutzer und Rechte anlegen

Ersetze DEIN_DB_PASSWORT durch ein starkes Passwort und notiere es.

```
CREATE DATABASE psono;
CREATE USER psono WITH PASSWORD 'DEIN_DB_PASSWORT';
GRANT ALL PRIVILEGES ON DATABASE psono TO psono;
\c psono
GRANT ALL ON SCHEMA public TO psono;
```



Hinweis zu \c psono

Nach \c psono wechselt der Prompt auf psono=# — das ist korrekt.

Schritt 1.4 — Erweiterungen installieren

```
CREATE EXTENSION IF NOT EXISTS ltree;
CREATE EXTENSION IF NOT EXISTS pgcrypto;
```

Schritt 1.5 — psql beenden

```
\q
exit
```

Schritt 1.6 — pg_hba.conf für lokalen psono-Benutzer anpassen

Zuerst den Pfad ermitteln:

```
sudo -u postgres psql -c "SHOW hba_file;"
```

Typischer Pfad: /etc/postgresql/17/main/pg_hba.conf. Folgende Zeile einfügen:

```
sudo sed -i '/^local    all                                all/i local    all                                psono
md5' /etc/postgresql/17/main/pg_hba.conf
```

Schritt 1.7 — pg_hba.conf für Docker-Netzwerke anpassen

```
echo "host    psono    psono    172.17.0.0/16    md5" >>
/etc/postgresql/17/main/pg_hba.conf
echo "host    psono    psono    172.18.0.0/16    md5" >>
/etc/postgresql/17/main/pg_hba.conf
```

Schritt 1.8 — listen_addresses auf * setzen

PostgreSQL muss auf allen Interfaces lauschen, damit der Docker-Container es nach dem Reboot zuverlässig erreicht. Die Docker-Bridge-IPs (172.17.0.1, 172.18.0.1) existieren beim Systemstart noch nicht — PostgreSQL würde sonst auf localhost zurückfallen und der Container könnte sich nicht verbinden.

```
sudo -u postgres psql -c "ALTER SYSTEM SET listen_addresses = '*';"
```

Sicherheit gewährleistet durch nftables

listen_addresses = "*" bedeutet nicht dass PostgreSQL von außen erreichbar ist. In Phase X wird Port 5432 über nftables auf dem externen Interface (eth0) geblockt. pg_hba.conf beschränkt den Zugriff zusätzlich auf localhost und die Docker-Netzwerke.

Schritt 1.9 — PostgreSQL neu starten

Ein Neustart (nicht nur reload) ist nötig damit listen_addresses wirksam wird:

```
systemctl restart postgresql
```

Prüfung Phase 1

```
sudo -u postgres psql -U psono -d psono -W -c '\dx'
```

Erwartung: Tabelle mit ltree und pgcrypto.

```
sudo -u postgres psql -c "SHOW listen_addresses;"
```

Erwartung: *

```
netstat -tulnp | grep 5432
```

Erwartung: PostgreSQL lauscht auf 0.0.0.0:5432 — Port wird durch nftables in Phase X abgesichert.

Phase 2 — Backup einspielen

Das Backup vom alten System ist ein PostgreSQL Custom Format Dump im tar-Format.

Schritt 2.1 — Backup auf den neuen Host übertragen

```
scp BACKUP_DATEI.tar root@DEINE_SERVER_IP:/root/
```

Schritt 2.2 — Backup entpacken

```
mkdir -p /root/psono-backup/dump
tar -xf /root/BACKUP_DATEI.tar -C /root/psono-backup/dump
ls /root/psono-backup/dump
```

Erwartung: toc.dat und viele .dat-Dateien.

Schritt 2.3 — Backup einspielen

Wichtig: Der Schalter ist -Fd (directory format), nicht -Ft:

```
pg_restore \  
-U psono \  
-d psono \  
-Fd \  
--no-owner \  
--no-privileges \  
/root/psono-backup/dump
```

⚠ Mögliche Warnungen beim Import

pg_restore kann zwei harmlose Warnungen ausgeben: "must be owner of extension ltree" und "must be owner of extension pgcrypto". Das ist normal. Echte Fehler beginnen mit ERROR:

✅ Prüfung Phase 2

Benutzer zählen (Psono speichert User in restapi_user, nicht in auth_user):

```
psql -U psono -d psono -W -c 'SELECT count(*) FROM restapi_user;'
```

Erwartung: Anzahl der migrierten Psono-Benutzer.

Phase 3 — Psono-Konfiguration wiederherstellen

Schritt 3.1 — Verzeichnisse anlegen

```
mkdir -p /opt/docker/psono  
mkdir -p /opt/docker/psono-client
```

Schritt 3.2 — settings.yaml auf den Host übertragen

```
scp settings.yaml root@DEINE_SERVER_IP:/opt/docker/psono/settings.yaml
```

Schritt 3.3 — settings.yaml anpassen

1. Datenbankhost umstellen:

```
sed -i "s/'HOST': 'psono-database'/'HOST': 'host.docker.internal'/"  
/opt/docker/psono/settings.yaml
```

2. Datenbankpasswort setzen:

```
sed -i "s/'PASSWORD': '.*'/'PASSWORD': 'DEIN_DB_PASSWORT'/"  
/opt/docker/psono/settings.yaml
```

3. Redis/Valkey-Cache deaktivieren:

```
sed -i "s/^CACHE_ENABLE: TRUE/CACHE_ENABLE: FALSE/" /opt/docker/psono/settings.yaml
sed -i "s/^CACHE_REDIS: TRUE/CACHE_REDIS: FALSE/" /opt/docker/psono/settings.yaml
sed -i "s/^CACHE_REDIS_LOCATION: /#CACHE_REDIS_LOCATION: /"
/opt/docker/psono/settings.yaml
```

Warum Cache deaktivieren?

Die alte Installation hatte möglicherweise einen Redis/Valkey-Container. Ohne Deaktivierung startet Psono nicht, weil es versucht sich mit dem nicht vorhandenen Cache-Container zu verbinden.

Kontrolle:

```
grep -E "HOST|PASSWORD|CACHE" /opt/docker/psono/settings.yaml | grep -v "^#" | grep -v EMAIL
```

Schritt 3.4 — config.json für den Web-Client anlegen

Das domain-Feld steuert die voreingestellte Domain im Login-Formular:

```
cat > /opt/docker/psono-client/config.json << 'EOF'
{
  "backend_servers": [{
    "title": "Psono Server",
    "url": "https://DEINE_DOMAIN/server",
    "domain": "DEINE_BASIS_DOMAIN"
  }],
  "base_url": "https://DEINE_DOMAIN/",
  "allow_custom_server": true,
  "allow_registration": true,
  "allow_lost_password": true,
  "disable_download_bar": false,
  "remember_me_default": false,
  "trust_device_default": false,
  "authentication_methods": ["AUTHKEY"]
}
EOF
```

DEINE_BASIS_DOMAIN

Das domain-Feld bestimmt die vorausgefüllte Domain im Login-Formular. Wenn Psono unter `psono.example.com` läuft, die Benutzer aber `@example.com` als Username haben sollen, hier `"example.com"` eintragen.

Phase 4 — Psono-Container starten

Schritt 4.1 — Docker Compose Datei erstellen

```
cat > /opt/docker/psono/docker-compose.yml << 'EOF'
services:
  psono-combo:
    image: psono/psono-combo:latest
    restart: unless-stopped
```

```
ports:
  - "127.0.0.1:10200:80"
volumes:
  - /opt/docker/psono/settings.yaml:/root/.psono_server/settings.yaml:ro
  - /opt/docker/psono-client/config.json:/usr/share/nginx/html/config.json:ro
  - /opt/docker/psono-client/config.json:/usr/share/nginx/html/portal/
config.json:ro
  extra_hosts:
    - "host.docker.internal:host-gateway"
EOF
```

Zwei Volume-Mounts für config.json

Web-Client und Admin-Portal laden die config.json aus unterschiedlichen Pfaden im Container. Beide Mounts verweisen auf dieselbe lokale Datei — eine Änderung wirkt sich damit automatisch auf beide Login-Seiten aus.

Schritt 4.2 — Container starten

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

Schritt 4.3 — Logs prüfen

```
docker compose logs psono-combo 2>&1 | tail -20
```

Erwartung: "Listening on TCP address 0.0.0.0:8000" ohne ERROR-Meldungen.

Prüfung Phase 4

```
curl -s http://localhost:10200/server/info/ | head -c 200
```

Erwartung: JSON-Antwort mit {"info": ...} und der aktuellen Psono-Version.

Phase 5 — Nginx Reverse Proxy einrichten

Schritt 5.1 — Voraussetzungen prüfen

- Dehydrated ist installiert und konfiguriert
- DEINE_DOMAIN ist in /etc/dehydrated/domains.txt eingetragen
- Zertifikate wurden generiert:

```
ls /var/lib/dehydrated/certs/DEINE_DOMAIN/
# Erwartung: fullchain.pem und privkey.pem
```

Schritt 5.2 — Nginx-Konfiguration erstellen

Erstelle /etc/nginx/sites-available/DEINE_DOMAIN.conf (DEINE_DOMAIN ersetzen):

```
server {
    listen 80;
    listen [::]:80;
    server_name DEINE_DOMAIN;
    server_tokens off;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl default_server;
    listen [::]:443 ssl default_server;
    server_name DEINE_DOMAIN;

    include snippets/letsencrypt-acme-challenge.conf;

    ssl_certificate      /var/lib/dehydrated/certs/DEINE_DOMAIN/fullchain.pem;
    ssl_certificate_key  /var/lib/dehydrated/certs/DEINE_DOMAIN/privkey.pem;
    ssl_dhparam /etc/nginx/ssl/dhparam.pem;

    ssl_session_cache shared:MozSSL:50m;
    ssl_session_timeout 1d;
    ssl_session_tickets off;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-
ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-
POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-
SHA384;
    ssl_prefer_server_ciphers off;

    server_tokens off;

    add_header Strict-Transport-Security "max-age=15768000; includeSubDomains;"
always;
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header Referrer-Policy "same-origin" always;
    add_header Permissions-Policy "geolocation=(), camera=(), microphone=()" always;

    client_max_body_size 256m;

    location / {
        proxy_pass          http://127.0.0.1:10200;
        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;
    }
}
```

⚠ default_server und andere VHosts

Falls auf dem gleichen Nginx weitere VHosts mit Port 443 konfiguriert sind, muss sichergestellt werden dass nur ein VHost default_server für Port 443 ist. Andernfalls liefert Nginx beim SSL-Handshake möglicherweise das falsche Zertifikat.

Schritt 5.3 — Konfiguration aktivieren

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

✓ Prüfung Phase 5

```
curl -s https://DEINE_DOMAIN/server/info/ | head -c 200
```

Erwartung: JSON-Antwort über HTTPS mit der Psono-Version.

```
echo | openssl s_client -connect DEINE_DOMAIN:443 -servername DEINE_DOMAIN  
2>/dev/null | openssl x509 -noout -subject
```

Erwartung: subject=CN=DEINE_DOMAIN

Phase 6 — Firewall einrichten

PostgreSQL lauscht auf allen Interfaces (`listen_addresses = "*"`). Damit Port 5432 von außen nicht erreichbar ist, wird mit `nftables` eine Firewall-Regel eingerichtet die eingehende Verbindungen auf Port 5432 vom externen Interface (`eth0`) blockiert.



Warum `nftables` statt `listen_addresses` einschränken?

Docker-Bridge-Interfaces (`172.17.0.1`, `172.18.0.1`) existieren beim Systemstart noch nicht wenn PostgreSQL startet. Würde `listen_addresses` auf diese IPs eingeschränkt, fällt PostgreSQL beim Reboot auf `localhost` zurück und der Psono-Container kann sich nicht verbinden. Die Kombination `listen_addresses="*"` + `nftables`-Regel ist die zuverlässigere Lösung.

Schritt 6.1 — `nftables`-Regel anlegen

Eigene `nftables`-Tabelle für Psono erstellen und Port 5432 auf `eth0` blockieren:

```
nft add table inet psono-firewall  
nft add chain inet psono-firewall input '{ type filter hook input priority 0;  
policy accept; }'  
nft add rule inet psono-firewall input iifname "eth0" tcp dport 5432 drop
```



Interface-Name prüfen

Das externe Interface heißt hier `"eth0"`. Falls dein Server einen anderen Namen verwendet (z.B. `ens3`, `enp0s3`), zuerst mit `"ip a"` prüfen und den Befehl entsprechend anpassen.

Schritt 6.2 — Regel prüfen

```
nft list table inet psono-firewall
```

Erwartung:

```
table inet psono-firewall {
    chain input {
        type filter hook input priority filter; policy accept;
        iifname "eth0" tcp dport 5432 drop
    }
}
```

Schritt 6.3 — Regel persistent machen

Die Regel muss in `/etc/nftables.conf` eingetragen werden damit sie nach einem Reboot erhalten bleibt. Die Datei existiert bereits mit Basis-Konfiguration — wir ergänzen nur unsere Tabelle am Ende:

```
cat >> /etc/nftables.conf << 'EOF'

# Psono: PostgreSQL Port 5432 von außen blockieren
table inet psono-firewall {
    chain input {
        type filter hook input priority filter; policy accept;
        iifname "eth0" tcp dport 5432 drop
    }
}
EOF
```

Schritt 6.4 — nftables beim Systemstart aktivieren

```
systemctl enable nftables
systemctl is-enabled nftables
```

Erwartung: enabled

Schritt 6.5 — Konfiguration testen

```
nft -c -f /etc/nftables.conf
```

Keine Ausgabe = keine Fehler.

✅ Prüfung Phase 6

Von einem externen Rechner testen:

```
nc -zv DEINE_SERVER_IP 5432
```

Erwartung: Connection timed out — kein "Connected", kein "Connection refused".

Psono muss weiterhin funktionieren:

```
curl -s https://DEINE_DOMAIN/server/info/ | head -c 150
```

Erwartung: JSON-Antwort mit Psono-Version.

Phase 7 — Testen & Umschalten

Schritt 7.1 — DNS prüfen

```
host DEINE_DOMAIN
```

Erwartung: DEINE_SERVER_IP als Antwort.

Schritt 7.2 — Login-Test

Öffne https://DEINE_DOMAIN im Browser und melde dich mit einem bekannten Account an:

- Login funktioniert → Daten wurden korrekt migriert
- Passwörter in den Tresoren sichtbar → Schlüssel korrekt übernommen
- Login scheitert mit "Invalid password" → originale settings.yaml verwenden

Schritt 7.3 — Admin-Panel prüfen

Öffne https://DEINE_DOMAIN/portal/login und prüfe ob alle Benutzer und Gruppen vorhanden sind.

Schritt 7.4 — Altes System abschalten

- Alte Container psono-combo, psono-database und psono-valkey anhalten
- Container erst löschen wenn der neue Betrieb stabil läuft (1–2 Wochen)

Schritt 7.5 — Automatische Backups einrichten

```
mkdir -p /opt/backups/psono
# crontab -e → folgende Zeilen hinzufügen:
0 2 * * * pg_dump -U psono -Ft psono > /opt/backups/psono/psono-$(date +%Y-%m-%d).tar
0 3 * * * find /opt/backups/psono -name "*.tar" -mtime +30 -delete
```

Troubleshooting — Häufige Probleme

"Connection refused" beim curl-Test auf Port 10200

```
docker compose ps
docker compose logs psono-combo | tail -30
```

"no pg_hba.conf entry" in den Docker-Logs

Das Docker-Netzwerk hat eine unerwartete IP-Range. Prüfen welche IP der Container hat:

```
docker inspect psono-psono-combo-1 | grep IPAddress
```

Dann das passende Subnetz in pg_hba.conf eintragen und PostgreSQL neu laden:

```
systemctl reload postgresql
```

502 Bad Gateway nach Reboot

PostgreSQL war noch nicht bereit als der Docker-Container startete. Prüfen:

```
docker ps | grep psono  
docker logs psono-psono-combo-1 2>&1 | tail -10
```

Falls "Connection refused" auf 172.17.0.1 — PostgreSQL neu starten, dann Container neu starten:

```
systemctl restart postgresql  
docker compose -f /opt/docker/psono/docker-compose.yml restart
```

"Redis ConnectionError" in den Docker-Logs

Cache ist noch aktiviert. CACHE_ENABLE in settings.yaml auf FALSE setzen und Container neu starten.

Nginx liefert falsches Zertifikat

```
grep -r "default_server" /etc/nginx/sites-enabled/
```

Nur ein VHost darf default_server für Port 443 sein. Alle anderen default_server-Direktiven für Port 443 entfernen.

"Invalid password" beim Psono-Login

Die kryptographischen Schlüssel in settings.yaml stimmen nicht mit der Datenbank überein. Die originale settings.yaml vom alten psono-combo Container verwenden — niemals neu generieren.

Port 5432 nach Reboot wieder offen

nftables-Dienst ist nicht aktiviert oder /etc/nftables.conf wurde nicht korrekt gespeichert. Prüfen:

```
systemctl is-enabled nftables  
nft -c -f /etc/nftables.conf  
systemctl restart nftables  
nft list table inet psono-firewall
```

