ADMIN:DOCS

Portainer unter Ubuntu 24.04 LTS installieren: Docker-Container per Web-GUI verwalten

Praxisanleitung zu Portainer CE unter Ubuntu 24.04 LTS: Docker-Engine-Setup, Volume-Persistenz, SSL/TLS, Stacks, Remote-Agents, Registries und Systemwartung.

Kategorie: Linux-Neulinge **Veröffentlicht:** 13.04.2025 **Zuletzt geprüft:** 08.09.2026

Die Verwaltung von Docker-Containern über die Kommandozeile mit `docker run` und dem Compose-Plugin ist der bewährte Standard im Linux-Alltag. Auf Servern, auf denen jedoch Dutzende Microservices, isolierte Bridge-Netzwerke, benannte Volumes und periodische Image-Aktualisierungen parallel betrieben werden, stößt die reine Terminal-Administration an Übersichtsgrenzen: Crash-Loops müssen über verschachtelte Log-Befehle analysiert werden, Ressourcenengpässe fallen erst bei Ausfällen auf, und verwaiste Volumes belegen unbemerkt Gigabytes an Speicherplatz.

Portainer Community Edition (CE) schließt diese operative Lücke. Als leichtgewichtige, webbasierte Verwaltungsplattform dockt Portainer direkt an die lokale Docker Engine oder an entfernt laufende Portainer-Agents an. Über eine aufgeräumte grafische Oberfläche steuerst du Container-Lebenszyklen, analysierst Streaming-Logs und Ressourcengraphen in Echtzeit, öffnest interaktive Shells direkt im Browser, deployst Multi-Container-Stacks und bereinigst ungenutzte Artefakte.

Auf einem Server mit **Ubuntu 24.04 LTS ("Noble Numbat")** erfordert der Betrieb von Portainer CE jedoch ein klares Sicherheits- und Architekturkonzept: Der Zugriff auf den Docker-Socket muss kontrolliert, Netzwerkfreigaben müssen strikt limitiert und die Weboberfläche sollte im Produktivbetrieb hinter einem gehärteten Nginx Reverse Proxy mit offiziellem TLS-Zertifikat betrieben werden.

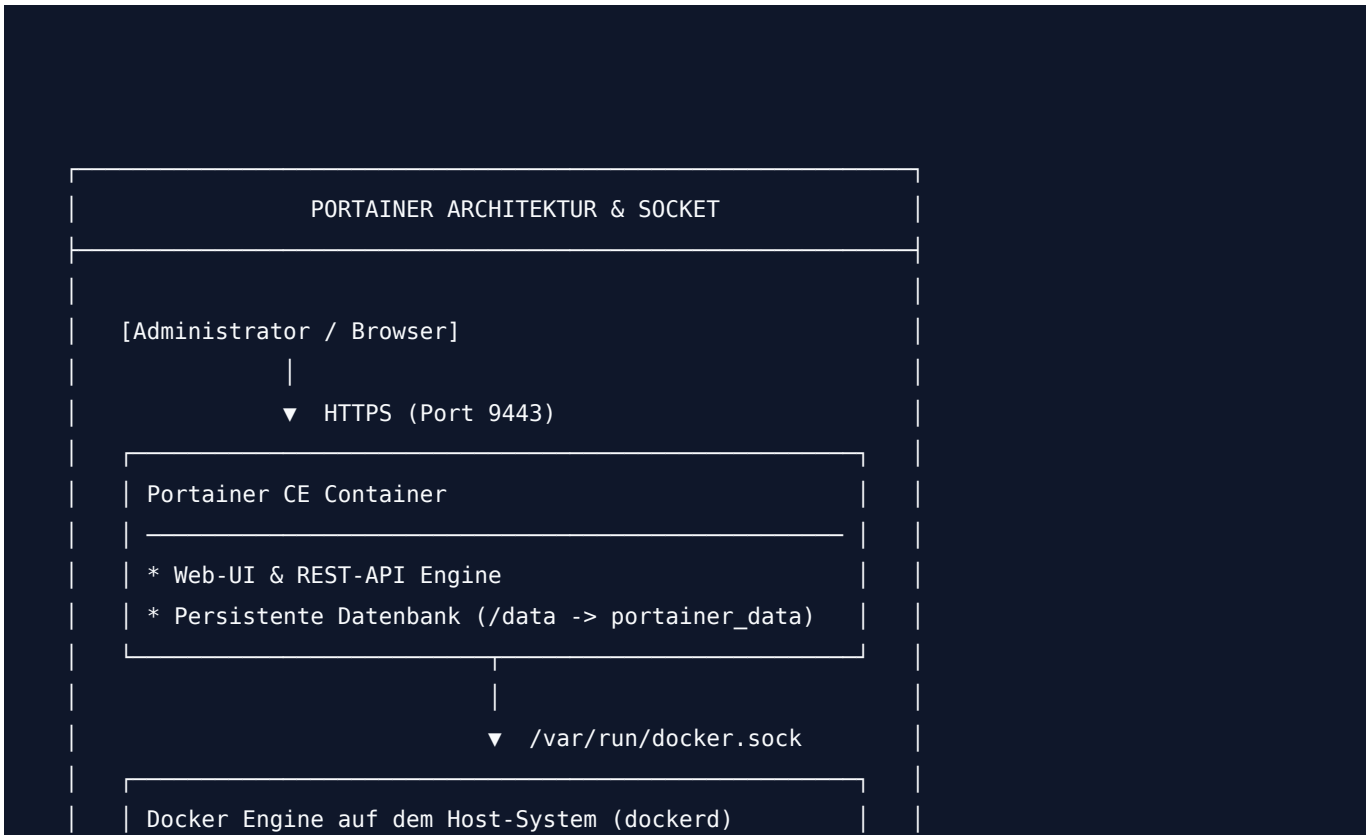
Quelle: https://admindocs.de/de/linux-neulinge/portainer-unter-ubuntu-24-04-lts-installieren-ein-umfassender-leitfaden

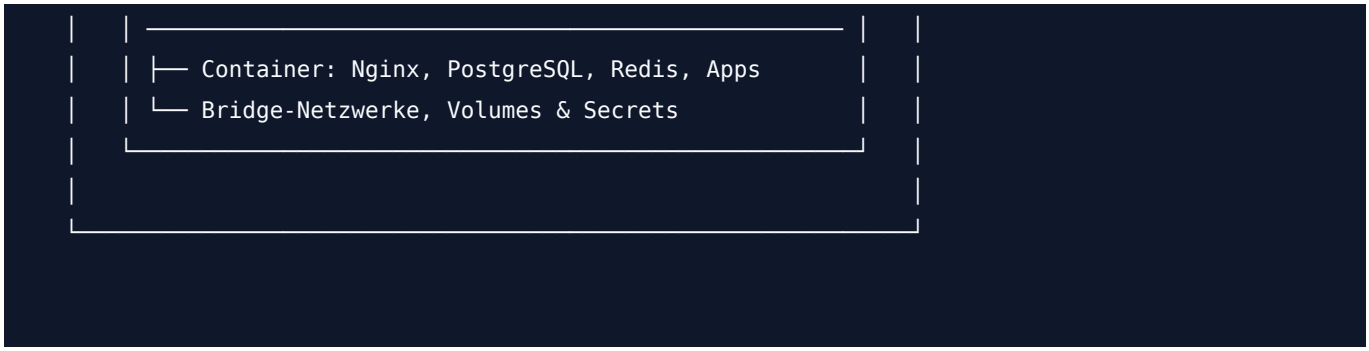
> **Voraussetzungen:** Ein Server mit Ubuntu 24.04 LTS sowie Root- bzw. `sudo`-Rechte. Falls die Docker Engine noch nicht eingerichtet ist, starten wir direkt mit der sauberen Installation des offiziellen Docker-Repositories. Weiterführende Grundlagen zur Containerisierung findest du auch in unserem Leitfaden zu Docker unter Ubuntu installieren und verwenden.

Architektur & Funktionsweise von Portainer

Wie Portainer mit der Docker Engine kommuniziert

Portainer wird nicht als klassischer Systemd-Dienst im Host-Dateisystem installiert, sondern läuft selbst als isolierter Docker-Container. Um den Host-Daemon steuern und Container starten, stoppen oder überwachen zu können, wird der UNIX-Socket des Docker-Daemons (`/var/run/docker.sock`) als Volume in den Portainer-Container durchgereicht:





Durch diesen Socket-Mount kommuniziert die Go-basierte Backend-Engine von Portainer direkt über HTTP-over-UNIX-Socket mit der Docker REST-API. Es entsteht keinerlei Virtualisierungs-Overhead, und Aktionen in der Web-Oberfläche werden ohne Verzögerung an den Host-Daemon übergeben.

Portainer Community Edition (CE) vs. Business Edition (BE)

Portainer wird in zwei Editionen gepflegt:

Funktion / Merkmal	Portainer Community Edition (CE)	Portainer Business Edition (BE)
Lizenzmodell	Open Source (kostenfrei)	Kommerziell (kostenfrei bis 3 Nodes)
Container- & Stack-Steuerung	Vollständig enthalten	Vollständig enthalten
Docker Compose Support	Integriert (Editor & Upload)	Integriert + Git-Repository-Autosync
Rechteverwaltung (RBAC)	Basis (Admin- & Standard-User)	Granulare Team-Rollen & Namensraum-Rechte
Authentifizierung	Lokal, Basis-LDAP & OAuth	Vollständiges Active Directory, SAML & SSO
Audit-Logging	Standard-Container-Logs	Revisionssicheres Aktivitäts-Audit-Log
Support	Community (Discourse / GitHub)	Kommerzieller 24/7 SLA-Herstellersupport

Für Standalone-Server, Entwicklungsumgebungen und kleine bis mittlere Produktionsumgebungen deckt die Community Edition (CE) alle administrativen Anforderungen lückenlos ab.

Netzwerkports & Sicherheitsüberlegungen

Portainer bindet standardmäßig zwei Kommunikationsports:

- **Port 9443 (HTTPS):** Der primäre, verschlüsselte Web-Port für das Dashboard und die REST-API. Portainer generiert beim Erststart automatisch ein selbstsigniertes TLS-Zertifikat.
- **Port 8000 (TCP):** Ein interner Tunnel-Port für den *Edge Agent*. Dieser Port wird nur benötigt, wenn entfernte Docker-Hosts hinter Firewalls oder NAT über das Internet angebunden werden.
- **Port 9000 (HTTP):** Der historische, unverschlüsselte Web-Port. Aus Sicherheitsgründen ist dieser Port in modernen Installationen deaktiviert und sollte nicht mehr exponiert werden.

> ⚠ **Sicherheitsrisiko des Docker-Sockets:** Der Zugriff auf `/var/run/docker.sock` verleiht faktisch **Root-Rechte auf dem Host-System**. Jeder Benutzer, der im Portainer-Dashboard neue Container anlegen darf, kann einen Container mit `-v /:/host` starten und jede Datei des Host-Betriebssystems modifizieren. Schütze den Port 9443 mit einem starken Administrator-Passwort, beschränke den Zugriff über die Firewall auf vertrauenswürdige Subnetze oder ein VPN und lies ergänzend unseren Leitfaden zur Linux-Server-Härtung mit FIDO2 und CrowdSec.

Vorbereitung & Docker-Installation auf Ubuntu 24.04 LTS

1. Systempakete aktualisieren

Vor der Installation bringen wir den Ubuntu-Server auf den aktuellen Paketstand und installieren die notwendigen Hilfswerkzeuge:

```
# Paketlisten auffrischen und installierte Software aktualisieren
sudo apt update && sudo apt full-upgrade -y

# Erforderliche Werkzeuge für sichere Repositories bereitstellen
sudo apt install -y curl ca-certificates
```

2. Offizielles Docker-Repository mit modernem Keyring einbinden

Wir binden das offizielle Upstream-Repository von Docker ein, um von kontinuierlichen Sicherheitsaktualisierungen und der aktuellen Docker Engine 27+ zu profitieren. Gemäß dem modernen APT-Standard speichern wir den Schlüssel direkt als ASCII-Armored Key (`.asc`):

```
# Verzeichnis für isolierte Schlüsselbunde anlegen
sudo install -m 0755 -d /etc/apt/keyrings

# Offiziellen GPG-Schlüssel von Docker im ASCII-Format herunterladen
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

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

3. Docker Engine und Compose-Plugin installieren

```
# Paketlisten mit der neuen Docker-Quelle synchronisieren
sudo apt update

# Docker Engine, CLI, Containerd und Compose-Plugin installieren
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-
compose-plugin

# Docker-Dienst aktivieren und Autostart sicherstellen
sudo systemctl enable --now docker
```

□□ Praktisches Beispiel:

Wir überprüfen die erfolgreiche Installation und fragen die aktive Docker-Version sowie den Socket-Status ab:

```
# Docker-Dienststatus kontrollieren
sudo systemctl is-active docker

# Installierte Versionen von Engine und Compose verifizieren
docker --version
docker compose version
```

Die Ausgabe bestätigt `active` sowie die Docker Engine Version 27 oder höher.

4. Firewall-Regeln konfigurieren (UFW)

Falls auf dem Ubuntu-Server die Standard-Firewall `ufw` aktiv ist, muss der HTTPS-Port freigegeben werden. Best Practice ist es, den Zugriff auf das eigene Administrations-Subnetz zu begrenzen:

```
# Zugriff auf Port 9443 aus dem vertrauenswürdigen Management-Netzwerk erlauben
sudo ufw allow from 192.168.1.0/24 to any port 9443 proto tcp comment "Portainer Admin
HTTPS"

# Falls kein getrenntes Management-Netz existiert (z. B. auf Cloud-Servern mit IP-
Bindung):
# sudo ufw allow 9443/tcp comment "Portainer HTTPS Web UI"

# Firewall-Status verifizieren
sudo ufw status verbose
```

Schritt-für-Schritt-Installation von Portainer CE

1. Persistentes Docker-Volume anlegen

Portainer speichert seine interne Datenbank (Benutzerkonten, API-Tokens, Endpoint-Metadaten und Stack-Konfigurationen) im Verzeichnis `/data`. Damit diese Konfiguration bei Container-Aktualisierungen oder Server-Reboots erhalten bleibt, erstellen wir ein dediziertes Docker-Volume:

```
# Benanntes Volume für Portainer-Konfiguration anlegen
sudo docker volume create portainer_data

# Volume-Eigenschaften inspizieren
sudo docker volume inspect portainer_data
```

2. Portainer CE Container starten (LTS-Release 2026)

Für Produktivsysteme empfiehlt Portainer offiziell den Einsatz des **LTS-Release-Zweigs** (`portainer/portainer-ce:lts`). Dieser garantiert langfristige Stabilität und kontinuierliche

Sicherheitspatches, ohne dass monatliche STS-Feature-Änderungen das Setup beeinträchtigen:

```
sudo docker run -d \
  -p 8000:8000 \
  -p 9443:9443 \
  --name portainer \
  --restart=always \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v portainer_data:/data \
  portainer/portainer-ce:lts
```

Bedeutung der gesetzten Parameter:

- `-d`: Startet den Container im Hintergrund (*detached mode*).
- `-p 9443:9443`: Leitet eingehenden HTTPS-Traffic von Host-Port 9443 auf den internen Port 9443 des Containers.
- `-p 8000:8000`: Hält den TCP-Port für optionale Portainer-Edge-Agents bereit.
- `--name portainer`: Weist einen eindeutigen Container-Namen zu, über den der Dienst gewartet wird.
- `--restart=always`: Stellt sicher, dass Portainer nach einem Server-Neustart oder einem Daemon-Crash automatisch hochfährt.
- `-v /var/run/docker.sock:/var/run/docker.sock`: Bindet den lokalen Docker-Socket ein, damit Portainer Container auf dem Host steuern kann.
- `-v portainer_data:/data`: Mappt das erstellte persistente Volume auf den Datenordner der Portainer-Datenbank.
- `portainer/portainer-ce:lts`: Zieht das offizielle Langzeit-Support-Image aus Docker Hub.

□□ Praktisches Beispiel:

Wir kontrollieren den Startvorgang und stellen sicher, dass der Webserver ordnungsgemäß initialisiert wurde:

```
# Laufenden Container in der Prozessliste prüfen
sudo docker ps --filter "name=portainer"

# Die letzten Logzeilen des Containers einsehen
sudo docker logs -n 15 portainer
```

In den Logs erscheint die Bestätigung: `[INFO] [main] [message: Starting Portainer...]` sowie der Hinweis auf den aktiven HTTPS-Server auf Port 9443.

3. Erstinitialisierung im Webbrowser

Öffne einen Browser und rufe die IP-Adresse deines Servers auf:

```
https://<DEINE-SERVER-IP>:9443
```

Da Portainer beim Erststart ein selbstsigniertes Zertifikat generiert, meldet der Webbrowser zunächst eine Sicherheitswarnung (*„Nicht vertrauenswürdige Verbindung“*). Bestätige die Sicherheitsausnahme, um zur Initialisierungsmaske zu gelangen.

```
┌──────────────────────────────────────────────────────────┐
│                ERSTINITIALISIERUNG DER WEB-UI            │
├──────────────────────────────────────────────────────────┤
│                                                          │
│   1. Administrator-Konto anlegen:                        │
│      * Benutzername: admin (oder individueller Login)    │
│      * Passwort: Mindestens 12 Zeichen (hohe Komplexität)  |
```

```
   |                                                                |
   |   2. 5-Minuten-Sicherheits-Timeout:                            |
   |      * Nach dem Start verbleiben exakt 5 Minuten Zeit          |
   |      * Bei Ablauf: sudo docker restart portainer               |
   |                                                                |
   |   3. Umgebungsauswahl:                                         |
   |      * Klick auf "Get Started" (Local Environment)             |
   |                                                                |
   └────────────────────────────────────────────────────────────────┘
```

1. **Administrator anlegen:** Vergib einen Benutzernamen und ein sicheres Kennwort mit mindestens 12 Zeichen.
2. **5-Minuten-Schutzfenster beachten:** Wenn nach dem ersten Starten des Containers 5 Minuten lang kein Kennwort festgelegt wird, beendet Portainer die Weboberfläche aus Sicherheitsgründen automatisch. Sollte das passieren, startest du den Container mit `sudo docker restart portainer` neu.
3. **Umgebung verbinden:** Klicke nach der Kontoerstellung auf **Get Started**, um die lokale Docker-Engine (`local`) automatisch in die Übersicht aufzunehmen.

Tour durch die Portainer-Weboberfläche

Nach der Erstanmeldung landest du im Haupt-Dashboard. Klicke auf die Kachel **local**, um die Docker-Ressourcen des Servers zu verwalten:

```
   ┌────────────────────────────────────────────────────────────────┐
   |                PORTAINER DASHBOARD ÜBERSICHT                   |
   ├────────────────────────────────────────────────────────────────┤
   |                                                                |
   |   [Lokale Docker-Umgebung: local]                              |
   |   ├── Containers : Laufende, gestoppte & pausierte Dienste     |
   |   ├── Images     : Lokale Docker-Images & Layer-Caches         |
```

```
│   ├── Volumes    : Persistente Daten-Volumes               │
│   ├── Networks   : Bridge-, Host- & Overlay-Netzwerke      │
│   ├── Stacks     : Multi-Container-Stacks via Compose      │
│   └── Templates  : Vorgefertigte 1-Click-Anwendungen       │
│                                                            │
└────────────────────────────────────────────────────────────┘
```

Die Kernbereiche im Administrationsalltag

- **Containers:**
- Verschafft einen schnellen Überblick über Status, Host-Port-Mappings, IP-Adressen und Startzeiten aller Container.
- Aktionsleiste: **Start**, **Stop**, **Kill**, **Restart**, **Pause** und **Remove**.
- Integrierte Diagnose-Tools pro Container:
- **Logs:** Live-Log-Streaming mit optionaler Zeitstempel-Anzeige und Suchfunktion.
- **Inspect:** Rohdaten und JSON-Konfiguration des Containers.
- **Stats:** Live-Diagramme zu CPU-, Arbeitsspeicher-, Netzwerk- und Festplatten-I/O-Nutzung.
- **Console / Exec:** Öffnet direkt im Browser eine interaktive Shell (`/bin/sh` oder `/bin/bash`) innerhalb des Zielcontainers, ohne dass ein SSH-Login auf dem Host erforderlich ist.
- **Images:**
- Ermöglicht das gezielte Herunterladen (*Pull*) neuer Images aus beliebigen Registries.
- Funktion **Prune**: Löscht ungenutzte Zwischenschichten (*dangling layers*) und gibt Festplattenplatz frei.
- **Networks:**
- Verwaltet Docker-Netzwerke. Hier erstellst du isolierte benutzerdefinierte Bridge-Netzwerke mit integriertem DNS zur sauberen Kapselung von Anwendungslandschaften.
- **Volumes:**
- Übersicht aller benannten Volumes. Der Filter **Unused** hebt verwaiste Volumes hervor, die nach gelöschten Containern zurückgeblieben sind.
- **App Templates:**

- Vorkonfigurierte Vorlagen für Standardanwendungen (z. B. Redis, Nginx, MariaDB, WordPress), die mit einem Klick ohne manuelle YAML-Erstellung gestartet werden können.

Container- und Stack-Management in der Praxis

1. Einzelnen Container über die Weboberfläche starten

Um beispielsweise einen Nginx-Webserver bereitzustellen:

1. Klicke im Menü auf **Containers → Add container**.
2. **Name:** `demo-webserver`.
3. **Image:** `nginx:alpine`.
4. **Port mapping:** Wähle *publish a new network port* → Host `8080` auf Container `80` (TCP).
5. **Advanced container settings (unterer Bildschirmbereich):**

- **Restart policy:** Wähle `Unless stopped`.
- **Volumes:** Binde ein Verzeichnis des Hosts oder ein benanntes Volume ein.

1. Klicke auf **Deploy the container**. Portainer zieht das Image, konfiguriert die Ports und startet den Container.

2. Multi-Container-Anwendungen mit Stacks (Docker Compose)

Im professionellen Serverbetrieb werden Dienste selten isoliert betrieben. Eine Webanwendung benötigt typischerweise eine relationale Datenbank, Caching und ein gemeinsames internes Netzwerk. In Portainer wird dies über **Stacks** abgebildet.



```
  │   Stack: "production-web" (compose.yaml / Stacks)          │
  │   ├── Service 1: Web-Frontend (Nginx / Node.js)            │
  │   ├── Service 2: Datenbank (PostgreSQL / MariaDB)          │
  │   ├── Bridge-Netzwerk: app_net (Interne DNS-Auflösung)     │
  │   └── Persistente Volumes: db_data                         │
  │                                                            │
  │   Vorteil: Zentral versionierbar, isoliert orchestrierbar  │
  │                                                            │
  └────────────────────────────────────────────────────────────┘
```

> **Compose Specification Standard:** Beachte, dass in modernen Docker-Compose-Dateien das veraltete Attribut `version: '3.8'` nicht mehr verwendet wird. Gemäß der aktuellen *Compose Specification* beginnt die Konfigurationsdatei direkt mit dem Schlüssel `services:`.

Praktisches Beispiel:

Wir erstellen einen Stack bestehend aus einer PostgreSQL-17-Datenbank und einem Webserver.

1. Navigiere im linken Menü zu **Stacks → Add stack**.
2. Vergib den Namen `production-app`.
3. Füge im integrierten Web-Editor folgende Compose-Definition ein:

```
services:
  database:
    image: postgres:17-alpine
    container_name: app_postgres
    restart: unless-stopped
    environment:
      POSTGRES_DB: production_db
      POSTGRES_USER: db_user
      POSTGRES_PASSWORD: ExampleSuperSecretPassword2026!
```

```
    volumes:
      - db_data:/var/lib/postgresql/data
    networks:
      - backend_network

  webserver:
    image: nginx:alpine
    container_name: app_web
    restart: unless-stopped
    ports:
      - "8080:80"
    depends_on:
      - database
    networks:
      - backend_network

volumes:
  db_data:
    driver: local

networks:
  backend_network:
    driver: bridge
```

1. Klicke auf **Deploy the stack**.

Portainer erstellt automatisch das Bridge-Netzwerk `production-app_backend_network`, legt das Volume an und startet beide Dienste in der definierten Abhängigkeit. Die Container können sich untereinander direkt über ihre Servicenamen (`database` und `webserver`) per internem DNS ansprechen.

Produktions-Setup: Nginx Reverse Proxy mit Let's Encrypt TLS

Der Betrieb von Portainer über ein selbstsigniertes Zertifikat auf Port 9443 ist für Testphasen akzeptabel, im Produktivbetrieb jedoch fehleranfällig und unübersichtlich. Die saubere Lösung besteht darin, Portainer an das lokale Loopback-Interface zu binden und über einen Nginx Reverse Proxy mit offiziellem Let's-Encrypt-Zertifikat bereitzustellen:



> Detaillierte Schritte zur Einrichtung eines Webservers findest du in unserem Leitfaden zu LEMP-Stack auf Ubuntu 24.04 LTS einrichten (sowie für neuere Systeme unter Ubuntu 26.04 LEMP Stack).

Nginx Virtual Host Konfiguration für Portainer (`/etc/nginx/sites-available/portainer.conf`):

```
server {
    listen 80;
    server_name portainer.example.com;
    return 301 https://$host$request_uri;
}
```

```
server {
    listen 443 ssl http2;
    server_name portainer.example.com;

    ssl_certificate /etc/letsencrypt/live/portainer.example.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/portainer.example.com/privkey.pem;

    # TLS-Sicherheitseinstellungen
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;

    location / {
        proxy_pass https://127.0.0.1:9443;
        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 https;

        # Wichtig: WebSockets für Web-Terminal und Log-Streaming aktivieren
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        # SSL-Verifizierung gegen selbstsigniertes Zertifikat des Containers deaktivieren
        proxy_ssl_verify off;
    }
}
```

Die Header `Upgrade` und `Connection` sind unverzichtbar: Fehlen diese Direktiven, verweigert die integrierte Browser-Konsole von Portainer (`docker exec`) den Verbindungsaufbau.

Externe Registries & Multi-Node-Verwaltung

Private Container-Registries anbinden

Um Images aus privaten Repositories (z. B. GitHub Container Registry `ghcr.io` , GitLab oder einer eigenen Harbor-Instanz) zu beziehen:

1. Navigiere im linken Menü zu **Settings → Registries → Add registry**.

2. Wähle **Custom registry** (oder den passenden Cloud-Provider).
3. Trage Name, Registry-URL (z. B. `ghcr.io`) sowie deine Zugangsdaten (Benutzername und *Personal Access Token*) ein.
4. Klicke auf **Add registry**. Ab sofort authentifiziert sich Portainer beim Ziehen von Images automatisch gegen diese Quelle.

Mehrere Docker-Hosts über den Portainer Agent verwalten

Müssen mehrere Docker-Server zentral administriert werden, muss auf zusätzlichen Servern kein vollständiges Portainer-Dashboard installiert werden. Stattdessen genügt der extrem ressourcensparende **Portainer Agent**.

Auf dem **entfernten Server (Remote Node)** ausführen:

```
sudo docker run -d \
  -p 9001:9001 \
  --name portainer_agent \
  --restart=always \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /var/lib/docker/volumes:/var/lib/docker/volumes \
  portainer/agent:lts
```

Anschließend im **zentralen Portainer-Dashboard**:

1. Navigiere zu **Environments → Add environment**.
2. Wähle **Docker Standalone → Portainer Agent**.
3. Hinterlege den Anzeigenamen des Servers sowie dessen Adresse: `<REMOTE-SERVER-IP>:9001`.
4. Klicke auf **Connect**.

Der entfernte Host taucht sofort in der Umgebungsliste auf und kann mit allen Funktionen gesteuert werden.

Wartung, Datensicherung & Fehlerbehebung

Portainer CE sicher aktualisieren

Da Portainer in einem Container läuft, erfolgt die Aktualisierung durch das Austauschen des Docker-Images. Sämtliche Konfigurationsdaten verbleiben im Volume `portainer_data`:

```
# 1. Bestehenden Portainer-Container stoppen
sudo docker stop portainer

# 2. Veralteten Container entfernen (das Volume bleibt vollständig erhalten!)
sudo docker rm portainer

# 3. Neuestes LTS-Image aus der Registry ziehen
sudo docker pull portainer/portainer-ce:lts

# 4. Container mit identischen Parametern neu starten
sudo docker run -d \
  -p 8000:8000 \
  -p 9443:9443 \
  --name portainer \
  --restart=always \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v portainer_data:/data \
  portainer/portainer-ce:lts
```

Vollständige Datenbanksicherung

1. **Über die grafische Oberfläche:**

Navigiere zu **Settings → Backup Portainer**, wähle *Download backup file* und schütze das Archiv optional mit einem Kennwort.

1. **Automatisiert über das Terminal (Volume-Backup):**

```
# Sichert den Inhalt des Volumes portainer_data in ein komprimiertes Tar-Archiv
sudo docker run --rm \
  -v portainer_data:/data:ro \
  -v $(pwd):/backup \
  alpine tar -czf /backup/portainer_backup_$(date +%F).tar.gz -C /data .
```

Typische Probleme lösen

□□ Praktisches Beispiel:

1. Sicherheits-Timeout beim Erststart: Erscheint die Meldung *„Your Portainer instance timed out for security purposes“*, wurde das Zeitfenster von 5 Minuten nach Containerstart überschritten:

```
# Container neu starten, um den 5-Minuten-Zähler zurückzusetzen
sudo docker restart portainer
```

2. Port-Kollision auf 9443: Läuft auf dem Server bereits ein anderer Webdienst, blockiert dieser Port 9443:

```
# Prüfen, welcher Prozess den Port belegt
sudo ss -tulpn | grep 9443
```

Lösung: Mappe den externen Host-Port auf einen freien Port (z. B. `-p 10443:9443`) oder binde Portainer hinter Nginx ein.

3. Ungenutzten Festplattenplatz bereinigen: Alte Image-Schichten und gestoppte Container füllen die Systemfestplatte:

```
# Gründliche Bereinigung aller ungenutzten Images, Build-Caches und gestoppten Container
sudo docker system prune -af --volumes
```

Befehlsreferenz (Cheatsheet)

Befehl / Aufruf	Bereich	Funktion & Zweck im Alltag
`docker volume create portainer_data`	Storage	Erstellt das persistente Speicher-Volume für die Portainer-Datenbank
`docker run -d -p 9443:9443 ... portainer-ce:lts`	Setup	Startet den Portainer-Server im abgesicherten LTS-Modus
`docker run -d -p 9001:9001 ... agent:lts`	Multi-Node	Startet den Portainer-Agenten auf entfernten Docker-Hosts
`sudo docker ps --filter "name=portainer"`	Diagnose	Kontrolliert den Betriebs- und Portstatus des Portainer-Containers
`sudo docker logs -f portainer`	Diagnose	Streamt die Live-Logs der Portainer-Engine im Terminal
`sudo docker restart portainer`	Wartung	Startet Portainer neu (setzt auch das 5-Minuten-Setup-Timeout zurück)
`sudo docker stop portainer && sudo docker rm portainer`	Update	Entfernt die alte Container-Instanz vor dem Einspielen eines Updates
`sudo docker pull portainer/portainer-ce:lts`	Update	Lädt das aktuelle LTS-Image vor dem Neustart herunter

Befehl / Aufruf	Bereich	Funktion & Zweck im Alltag
`sudo docker volume inspect portainer_data`	Storage	Zeigt den tatsächlichen Speicherpfad des Volumes auf dem Host an
`docker system prune -af --volumes`	Hygiene	Bereinigt alle ungenutzten Images, Container und verwaisten Volumes
`sudo ss -tulpn	grep -E "(9443|8000)"`	Netzwerk	Verifiziert die aktiven Netzwerk-Sockets auf dem Ubuntu-Host
`sudo ufw status verbose`	Sicherheit	Kontrolliert die aktiven UFW-Firewall-Regeln für Portainer

Weiterführende Ressourcen

Ressource / Dokumentation	Typ	Beschreibung / Einsatzzweck
Offizielle Portainer Dokumentation	Handbuch	Vollständiges Referenzhandbuch für Portainer CE und Business Edition
Portainer GitHub Repository	Quellcode	Offizielle Release Notes, Bugtracker und Community-Diskussionen
Docker Compose Spezifikation	Referenz	Offizieller Standard für moderne compose.yaml Multi-Container-Stacks
Docker unter Ubuntu installieren	Praxis-Guide	Schritt-für-Schritt-Anleitung zur sauberen Docker-Engine-Einrichtung
Linux Server Härtung	Sicherheit	Best Practices zur Absicherung von SSH, Sockets und Firewalls
Ubuntu 24.04 LEMP Stack & Nginx	Webserver	Einrichtung von Nginx als performanter Reverse Proxy mit SSL/TLS

Fazit

Portainer CE schlägt eine verlässliche Brücke zwischen der kompromisslosen Flexibilität der Linux-Kommandozeile und dem Wunsch nach einer zentralen, übersichtlichen Schaltzentrale. Auf einem Server mit Ubuntu 24.04 LTS lässt sich die Management-Plattform innerhalb weniger Minuten aufsetzen und benötigt im laufenden Betrieb kaum Systemressourcen.

Durch die native Unterstützung moderner **Docker Compose Stacks**, webbasierte Konsolenzugriffe in Container und die Möglichkeit, eine Flotte entfernter Server über den **Portainer Agent** zu steuern, wird Portainer zu einem unersetzlichen Werkzeug für Administratoren und DevOps-Teams. Wer die Sicherheitsimplikationen des Docker-Sockets verstanden hat und Portainer hinter einem gehärteten Nginx Reverse Proxy betreibt, erhält eine zukunftssichere und hochperformante Plattform.

> **Praxis-Tipp für deinen Produktivbetrieb:** Nutze für geschäftskritische Container-Umgebungen stets das `:lts`-Tag anstelle von `:latest`. Dadurch vermeidest du unvorhergesehene Breaking Changes durch monatliche STS-Feature-Rollouts und sicherst dir eine erprobte Langzeit-Stabilität.

Wenn du deine Container-Infrastruktur über die grafische Verwaltung hinaus weiter absichern möchtest, zeigt dir unser Leitfaden zur Linux-Server-Härtung mit FIDO2 und CrowdSec, wie du das Fundament deines Host-Systems gegen unberechtigte Zugriffe und automatisierte Angriffe abschirmst.

Quelle: https://admindocs.de/de/linux-neulinge/portainer-unter-ubuntu-24-04-lts-installieren-ein-umfassender-leitfaden