ADMIN:DOCS

Docker auf Ubuntu 26.04 LTS installieren und konfigurieren

Leitfaden zur Docker-Installation auf Ubuntu 26.04 LTS: Vom offiziellen APT-Repository über Daemon-Tuning und Log-Rotation bis zur UFW-Firewall-Integration.

Kategorie: Serverumgebungen **Veröffentlicht:** 20.08.2026

Unter Ubuntu 26.04 LTS (Resolute Raccoon) ist Docker die am weitesten verbreitete Plattform zur containerbasierten Isolation von Serverdiensten. Im Gegensatz zu vollwertigen virtuellen Maschinen teilen sich Container den Linux-Kernel des Host-Systems. Die Isolation von Prozessen, Dateisystemen und Netzwerkstapeln erfolgt nativ über Linux-Kernel-Mechanismen: `Control Groups (cgroups v2)` und `Kernel Namespaces`.

Dieser Leitfaden führt durch die Einrichtung der offiziellen Docker Community Edition (CE) auf Ubuntu 26.04 LTS. Er behandelt die sichere Verwaltung des Docker-GPG-Schlüssels über `/etc/apt/keyrings`, die Konfiguration des Daemons für automatisierte Log-Rotation, die Berechtigungssteuerung über die `docker`-Gruppe sowie das Zusammenspiel mit Paketfiltern und Firewalls wie UFW.

> **Wichtiger Hinweis:** Dieser Artikel richtet sich an **Linux-Administratoren und IT-Fachkräfte**, die bereits grundlegende Erfahrungen mit Ubuntu und der Kommandozeile haben. Du solltest dich im Terminal sicher bewegen können und verstehen, was Paketverwaltung bedeutet. Gleichzeitig ist dieser Leitfaden auch für **Einsteiger in die Container-Technologie** geeignet, die Docker von Grund auf lernen möchten.

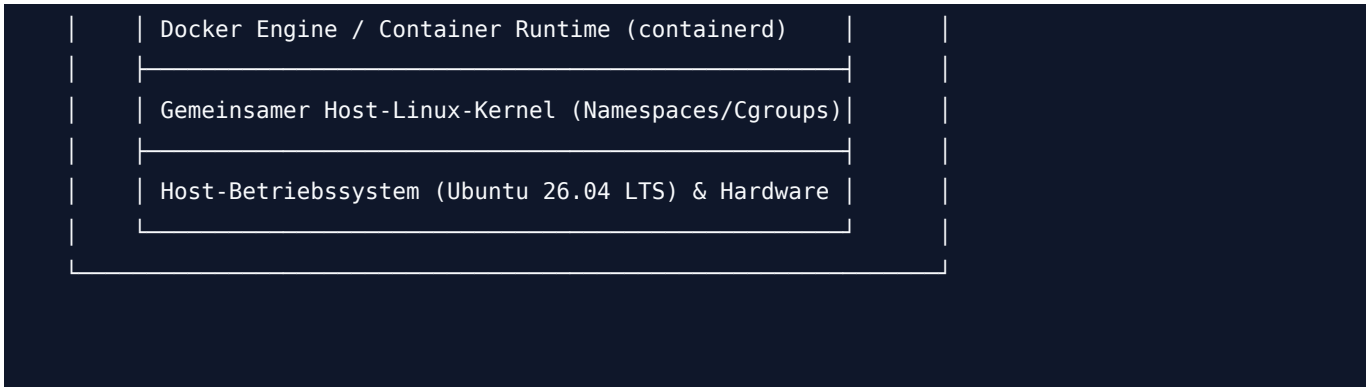
Quelle: https://admindocs.de/de/serverumgebungen/ubuntu-26-04-docker-installieren-und-verwenden

> ⚠ **Privilegienstufe des Docker-Daemons:** Der Docker-Daemon (`dockerd`) läuft standardmäßig mit vollständigen Root-Rechten. Wer Zugriff auf den Docker-Socket `/var/run/docker.sock` hat, besitzt de facto uneingeschränkten Root-Zugriff auf das Basissystem. Führe Container im Produktivbetrieb nach dem Least-Privilege-Prinzip aus und vergib Gruppenmitgliedschaften restriktiv.

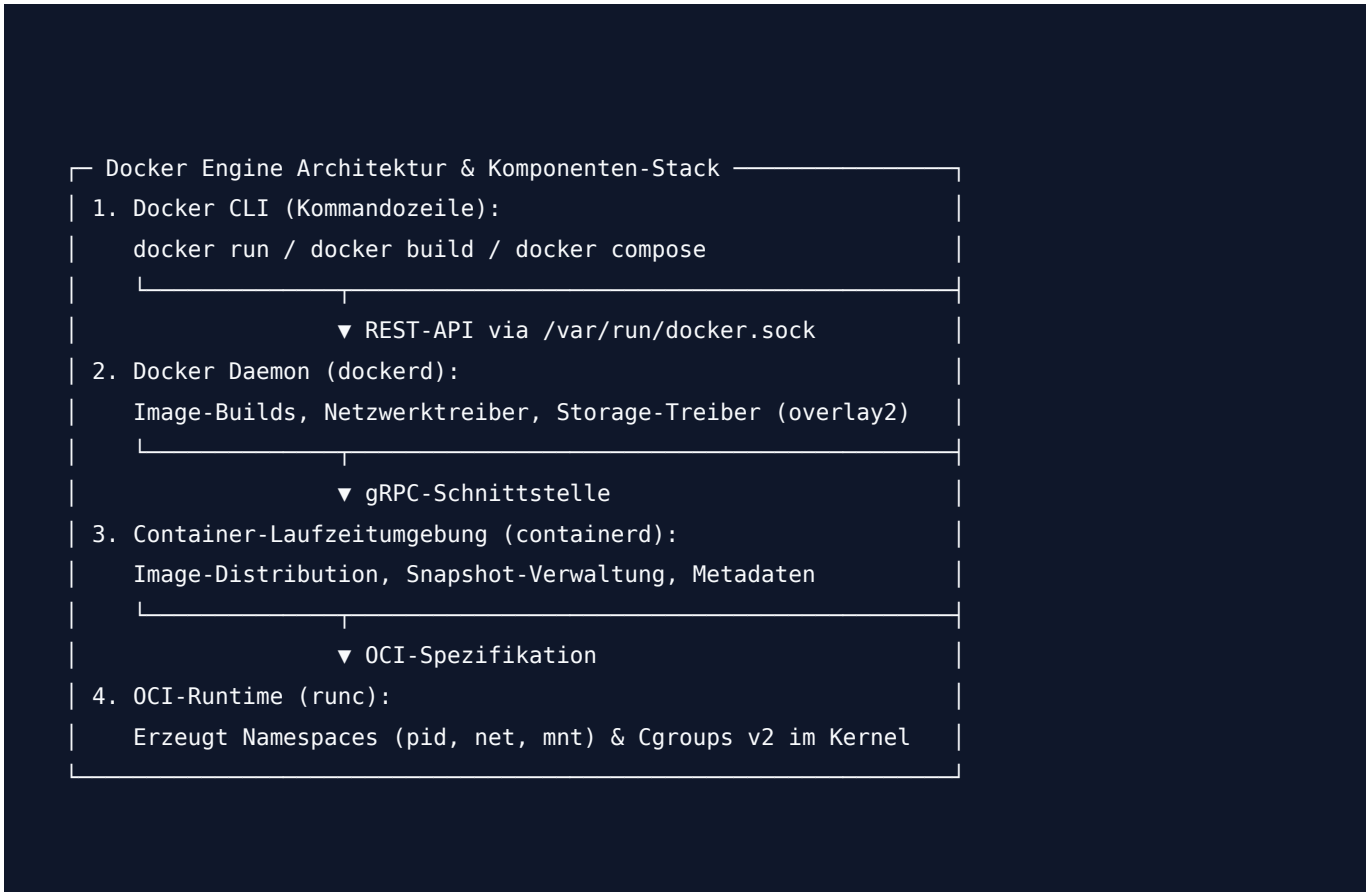
1. Architektur und Funktionsweise: Container vs. Virtuelle Maschinen

Um Docker im Serverbetrieb stabil zu betreiben, ist das Verständnis der Abgrenzung zu Hypervisor-basierter Virtualisierung (wie KVM oder Proxmox VE) unerlässlich. Virtuelle Maschinen emulieren komplette Hardware-Ressourcen und booten ein eigenständiges Gast-Betriebssystem. Container hingegen kapseln Prozesse auf dem bereits laufenden Host-Kernel:





Die Docker Engine selbst setzt sich aus mehreren modular arbeitenden Schichten zusammen:



Wesentliche Kernel-Mechanismen hinter Docker

- **Namespaces:** Isolieren Sichtbarkeiten. Der `pid`-Namespace sorgt dafür, dass ein Prozess im Container eine eigene PID-Liste (beginnend bei PID 1) sieht. Der `net`-Namespace weist dem Container eigene virtuelle Netzwerkschnittstellen und Routing-Tabellen zu. `mnt` isoliert Einhängepunkte, `ipc` gemeinsame Speicherbereiche und `user` Benutzer-IDs.

- **Control Groups (cgroups v2):** Beschränken und überwachen Ressourcen. Über cgroups legst du fest, wie viel Arbeitsspeicher, CPU-Zyklen oder Disk-I/O ein Container maximal verbrauchen darf, um Denial-of-Service-Zustände durch fehlerhafte Applikationen zu unterbinden.
- **Storage-Treiber (`overlay2`):** Erstellt ein schreibbares Layer-Dateisystem auf Basis von Copy-on-Write (CoW). Das Basis-Image bleibt unveränderlich (`read-only`), während Änderungen in einer dünnen oberen Ebene (`upperdir`) gespeichert werden.

Abgrenzung: Wann Container und wann VMs?

Kriterium	Docker Container	Virtuelle Maschine (KVM / Proxmox)
Kernel	Geteilter Host-Kernel	Eigener Kernel pro Gast
Startzeit	Millisekunden bis Sekunden	10 bis 60 Sekunden (Bootvorgang)
RAM-Overhead	Gering (nur tatsächlicher Prozessbedarf)	Hoch (Host reserviert festen Gast-RAM)
Sicherheitsgrenze	Shared Kernel Isolation (Namespaces)	Strikte Hardware-Virtualisierung
Betriebssysteme	Nur Linux auf Linux-Host	Beliebig (Linux, BSD, Windows)
Kernel-Module	Nicht ladbar durch unprivilegierte Container	Vollständige Kernel-Kontrolle im Gast

2. Systemvoraussetzungen prüfen und Basissystem vorbereiten

Vor der Installation des offiziellen Docker-Repositories prüfen wir Systemarchitektur, Kernel-Version und entfernen eventuell vorhandene Altpakete aus früheren Distributionsexporten.

Hardware- und Architekturprüfung

Docker setzt zwingend eine 64-Bit-Architektur voraus (`x86_64` bzw. `amd64` oder `aarch64` / `arm64`). Führe die Abfrage im Terminal aus:

```
# Systemarchitektur und Kernelversion verifizieren
uname -m && uname -r
```

Die Ausgabe muss `x86_64` oder `aarch64` zurückgeben. Für Produktionsserver mit mehreren laufenden Containern empfiehlt sich ein Minimum von 2 vCPUs und 2–4 GB RAM, um Systempuffer und Container-Overlays sauber im Speicher vorzuhalten.

Verifiziere den Distributions-Codenamen von Ubuntu 26.04 LTS:

```
# Distributionsdaten abfragen
lsb_release -a
```

Altreste und Distributions-Standardpakete entfernen

Ubuntu führt in den Standard-Paketquellen historische Pakete wie `docker.io` oder veraltete Containerd-Builds. Um Versionskonflikte mit den offiziellen Upstream-Paketen von Docker Inc. auszuschließen, entfernen wir diese rückstandsfrei:

```
# Alte Distributionspakete bereinigen
sudo apt remove -y docker docker-engine docker.io containerd runc
```

Falls die Rückmeldung meldet, dass einzelne Pakete nicht installiert waren, ist das der reguläre Soll-Zustand.

System aktualisieren und Basiswerkzeuge einrichten

Bringe den Paketindex auf den aktuellen Stand und installiere die für den Download und die Signaturprüfung erforderlichen Werkzeuge:

```
# Basissystem aktualisieren
sudo apt update && sudo apt upgrade -y

# Erforderliche Hilfswerkzeuge installieren
sudo apt install -y ca-certificates curl gnupg lsb-release
```

> **Prüfung des Kernel-Zustands:** Wurde bei `apt upgrade` ein neuer Linux-Kernel installiert, ist ein Neustart (`sudo reboot`) vor der Docker-Installation angeraten. Nur so laufen die Kernel-Namespaces und `cgroups v2` unter dem neuesten Sicherheitsstand.

3. Offizielles Docker APT-Repository einbinden

Das offizielle Repository garantiert Zugriff auf aktuelle Docker CE Versionen, Sicherheits-Patches und offizielle Plugins wie Buildx und Compose.

GPG-Schlüsselbund sicher einrichten

Nach aktuellem Debian- und Ubuntu-Standard werden Repository-Schlüssel nicht mehr über das veraltete `apt-key` oder global unter `/etc/apt/trusted.gpg` abgelegt, sondern in ein dediziertes Verzeichnis unter `/etc/apt/keyrings` mit strikten Dateirechten:

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

# Offiziellen Docker GPG-Schlüssel herunterladen und als dearmored ASCII ablegen
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc

# Leserechte für alle Systemdienste und APT sicherstellen
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

Repository zur APT-Quellenliste hinzufügen

Trage die Paketquelle mit Referenz auf den heruntergeladenen Signaturschlüssel ein:

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

Aktualisiere den APT-Paketindex, um die neuen Pakete einzulesen:

```
# Paketindex mit neuer Quelle synchronisieren
sudo apt update
```

In der Konsolenausgabe bestätigt die Zeile `Get:... https://download.docker.com/linux/ubuntu ... InRelease`, dass APT das Repository erfolgreich kontaktiert hat.

4. Docker Engine und Container-Tools installieren

Installiere das vollständige Docker-Toolset bestehend aus der Engine, dem CLI, containerd und den offiziellen Plugins:

```
# Docker CE, containerd und moderne Plugins installieren
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

Die installierten Pakete erfüllen folgende Funktionen:

- `docker-ce`: Der eigentliche Daemon-Dienst (`dockerd`).
- `docker-ce-cli`: Die Kommandozeilenschnittstelle (`docker`), die mit der Daemon-REST-API interagiert.
- `containerd.io`: Die Open-Source Container-Laufzeitumgebung nach OCI-Standard.
- `docker-buildx-plugin`: Modernes Build-Tooling für Multi-Architektur-Images (Moby BuildKit).
- `docker-compose-plugin`: Das native Compose-Kommando (`docker compose`) als direktes CLI-Plugin.

Dienststatus und Version verifizieren

Prüfe den Laufzeitstatus des Systemd-Dienstes:

```
# Docker-Dienststatus abfragen
```

```
sudo systemctl status docker --no-pager
```

Die Ausgabe muss `Active: active (running)` anzeigen. Überprüfe die installierten Versionen:

```
# Version der Engine prüfen
sudo docker version

# Version von Docker Compose prüfen
docker compose version
```

> **Hinweis zu Docker Compose v2:** Das moderne Docker Compose ist als CLI-Plugin konzipiert und wird mit Leerzeichen (`docker compose`) aufgerufen. Die historische, separate Python-Binärdatei `docker-compose` (mit Bindestrich) ist veraltet und wird nicht mehr verwendet.

Funktionstest der Installation ausführen

Starte den standardisierten Verifikationscontainer:

```
# Offiziellen Funktionstest-Container ausführen
sudo docker run --rm hello-world
```

Dieser Aufruf veranlasst die Docker Engine, das Image `hello-world` von Docker Hub herunterzuladen, einen kurzlebigen Container zu instanziieren und die Bestätigungsmeldung `Hello from Docker!` auf stdout auszugeben. Der Parameter `--rm` weist Docker an, den Container

nach Beendigung sofort aus dem Dateisystem zu entfernen.

5. Berechtigungsmodell: Docker ohne Sudo einrichten

Standardmäßig bindet der Docker-Daemon seinen Steuerungs-Socket `/var/run/docker.sock` an den Benutzer `root` und die Gruppe `docker`. Reguläre Systembenutzer können den Socket ohne Root-Rechte nicht ansprechen.

Um Docker-Befehle im Alltag ohne vorangestelltes `sudo` auszuführen, fügen wir den administrativen Benutzer zur Systemgruppe `docker` hinzu:

```
# Aktuellen Benutzer zur docker-Gruppe hinzufügen
sudo usermod -aG docker $USER
```

Damit die Gruppenänderung für die laufende Shell wirksam wird, meldest du dich entweder neu an oder aktivierst die Gruppenrechte direkt in der aktuellen Terminalsitzung:

```
# Gruppenzugehörigkeit in der aktuellen Shell sofort anwenden
newgrp docker
```

Verifiziere den unprivilegierten Zugriff ohne `sudo`:

```
# Testlauf als normaler Benutzer
docker run --rm alpine echo "Container läuft ohne sudo"
```

> ⚠ **Sicherheitsauswirkung der Docker-Gruppe:** Die Aufnahme eines Benutzers in die `docker`-Gruppe gewährt effektive Root-Berechtigung auf dem Host. Ein Benutzer mit Zugriff auf den Docker-Socket kann beliebige Host-Verzeichnisse (`/etc` , `/root`) in einen Container mounten (`docker run -v /:/host-root ...`) und manipulieren. Füge ausschließlich vertrauenswürdige Administrationskonten hinzu.

6. Daemon-Konfiguration: Logging, Storage und Performance

Ohne gezielte Konfiguration schreibt Docker Container-Logs standardmäßig unbegrenzt auf die Festplatte. Bei produktiven Diensten mit regem Traffic führt dies früher oder später zum Überlaufen der Root-Partition.

Wir hinterlegen eine zentrale Konfigurationsdatei `/etc/docker/daemon.json` , um das Logging zu limitieren und den Storage-Treiber explizit festzulegen:

```
# Konfigurationsverzeichnis anlegen
sudo mkdir -p /etc/docker

# Zentrale Daemon-Konfiguration anlegen
sudo tee /etc/docker/daemon.json > /dev/null <<'EOF'
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
  },
  "storage-driver": "overlay2",
  "live-restore": true
}
EOF
```

Die konfigurierten Direktiven bewirken:

- `log-driver: json-file`: Schreibt Standard-Logs im JSON-Format, kompatibel mit `docker logs`.
- `max-size: 10m` & `max-file: 3`: Rotiert Logs automatisch bei 10 MB und hält maximal 3 Generationen vor. Ein Container belegt somit niemals mehr als 30 MB Speicherplatz für Logs.
- `storage-driver: overlay2`: Der empfohlene, performante Speicher-Treiber für Linux-Dateisysteme (`ext4`, `xfs`).
- `live-restore: true`: Hält Container am Laufen, selbst wenn der Docker-Daemon (z. B. bei Paketupdates oder Dienstneustarts) neu gestartet wird. Dies minimiert Ausfallzeiten.

Lade den Docker-Daemon neu, um die Einstellungen zu aktivieren:

```
# Docker Daemon neu starten
sudo systemctl restart docker

# Angewendete Konfiguration verifizieren
docker info | grep -E '(Logging Driver|Storage Driver|Live Restore)'
```

7. Firewall-Integration und Kernel-Parameter (UFW vs. Docker)

Eine der häufigsten Sicherheitsfallen im Linux-Serverbetrieb ist das automatische Zusammenspiel von Docker und der Paketfilter-Firewall UFW.

Die UFW-Bypass-Falle verstehen

Docker manipuliert bei der Port-Freigabe (`-p 8080:80`) direkt die `PREROUTING`-Kette von `iptables`/`nftables`. Diese Regeln greifen **bevor** die Filterregeln von UFW ausgewertet werden. Ein Port, den du in UFW nicht freigegeben hast, ist von außen dennoch erreichbar, wenn ein Container

mit `-p 0.0.0.0:Port:Port` gebunden wird.

> **Sicherheitsregel für Port-Bindings:** Binde Container-Ports für interne Dienste (z. B. Datenbanken oder Caches) immer explizit an Localhost (`127.0.0.1`), wenn sie nicht direkt dem Internet ausgesetzt werden sollen:
>
> ```
> # Sicher: Dienst lauscht nur auf dem lokalen Loopback-Interface
> docker run -d -p 127.0.0.1:3306:3306 mariadb
> ```

Kernel-Parameter für Bridge-Netzwerkverkehr aktivieren

Damit Paketfilter wie `iptables` den Netzwerkverkehr über virtuelle Linux-Bridges (`docker0`) ordnungsgemäß filtern können, müssen die entsprechenden Kernel-Module und Sysctl-Flags gesetzt sein:

```
# Sysctl-Parameter für Bridge-Netzwerkverkehr hinterlegen
sudo tee /etc/sysctl.d/99-docker-bridge.conf > /dev/null <<'EOF'
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward = 1
EOF

# Parameter sofort ohne Neustart laden
sudo sysctl --system
```

8. Praktische Anwendung: Container-Lifecycle und Docker Compose

Nach der Grundinstallation veranschaulichen wir den typischen Betriebsablauf anhand zweier praktischer Szenarien: der manuellen Container-Steuerung über die CLI und dem deklarativen Multi-Container-Betrieb via Docker Compose.

Praktischer Container-Lifecycle an einem Webdienst

□□ Praktisches Beispiel:

Wir starten einen Nginx-Webserver als Hintergrunddienst (Detached-Modus), binden ihn an Port 8080, inspizieren die Logs und führen einen interaktiven Befehl im Container aus:

```
# Nginx-Container im Hintergrund starten
docker run -d --name webserver -p 8080:80 nginx:alpine

# Laufenden Status und Port-Mappings prüfen
docker ps

# Echtzeit-Logs des Containers einsehen
docker logs webserver

# Einen Befehl direkt im Container-Namespace ausführen
docker exec -it webserver nginx -v

# HTTP-Erreichbarkeit vom Host aus testen
curl -I http://127.0.0.1:8080

# Container nach erfolgreichem Test sauber beenden und entfernen
docker stop webserver && docker rm webserver
```

Deklarativer Betrieb mit Docker Compose

Im professionellen Serverbetrieb werden Containerkonfigurationen selten über lange Ad-hoc-Befehle gestartet, sondern versioniert in einer `compose.yaml`-Datei verwaltet.

Erstelle ein Projektverzeichnis und die Konfigurationsdatei:

```
# Projektverzeichnis anlegen
mkdir -p ~/compose-demo && cd ~/compose-demo

# Deklarative Compose-Konfiguration anlegen
tee compose.yaml <<'EOF'
services:
  web:
    image: nginx:alpine
    container_name: demo-web
    restart: unless-stopped
    ports:
      - "127.0.0.1:8080:80"
    volumes:
      - ./html:/usr/share/nginx/html:ro

  redis:
    image: redis:alpine
    container_name: demo-redis
    restart: unless-stopped
    command: redis-server --appendonly yes
    volumes:
      - redis-data:/data

volumes:
  redis-data:
EOF

# Lokales HTML-Verzeichnis mit Demodatei vorbereiten
mkdir -p html
echo "<h1>Docker Compose auf Ubuntu 26.04 laeuft</h1>" > html/index.html

# Multi-Container-Stack im Hintergrund starten
docker compose up -d

# Status aller Services des Stacks prüfen
docker compose ps
```

```
# Stack nach Abschluss herunterfahren
docker compose down
```

9. Speicherverwaltung und Bereinigungs-Workflows

Im laufenden Betrieb sammeln sich durch Image-Aktualisierungen, beendete Test-Container und Build-Caches verwaiste Datenmengen unter `/var/lib/docker` an.

Speicherbelegung analysieren

Prüfe mit den integrierten Docker-Befehlen, wie viel Speicherplatz durch Images, Container, Volumes und den Build-Cache belegt wird:

```
# Detaillierte Speicherbelegung anzeigen
docker system df
```

Die Ausgabe schlüsselt exakt auf, welcher Anteil des Speichers aktiv genutzt wird und wie viel Prozent als `Reclaimable` (freigebbar) markiert sind.

Gezielte Bereinigung verwaister Ressourcen

□□ Praktisches Beispiel:

Ein Server zeigt nach mehrfachen Deployments eine hohe Speicherbelegung im Build-Cache und ungenutzten Images. Wir bereinigen das System kontrolliert:

```
# Unbenutzte Container, Netzwerke und ungetaggte Dangling-Images entfernen
```

```
docker system prune -f

# Gründliche Bereinigung: Alle ungenutzten Images entfernen (nicht nur Dangling-Images)
docker image prune -a --filter "until=168h" -f
```

> ⚠ **Achtung bei Volumes:** Der Befehl `docker system prune` löscht standardmäßig **keine** Docker-Volumes, um versehentlichen Datenverlust (z. B. von Datenbankdaten) zu verhindern. Erst der explizite Zusatz `--volumes` entfernt verwaiste Volumes dauerhaft.

10. Autostart und Systemd-Absicherung

Damit Docker und alle Container mit Neustart-Policies (`restart: unless-stopped` oder `restart: always`) nach einem Host-Reboot zuverlässig hochfahren, aktivieren wir die Systemd-Dienste:

```
# Autostart für Docker Engine und containerd sicherstellen
sudo systemctl enable docker.service containerd.service
```

End-to-End Verifikation des Gesamtsystems

Führe abschließend einen vollständigen Testlauf aus, der Netzwerkauflösung, Dateisysteme und Prozessisolation kombiniert:

```
# Vollständiger End-to-End Funktionstest mit Alpine Linux
docker run --rm alpine sh -c "uname -a && ping -c 1 -W 2 dns.google"
```

Erscheint in der Konsole die Kernel-Ausgabe sowie der erfolgreiche Ping-Transfer (`1 packets transmitted, 1 packets received`), arbeitet die Docker Engine auf Ubuntu 26.04 LTS fehlerfrei mit funktionierender DNS- und Bridge-Netzwerkweiterleitung.

Befehlsreferenz (Cheatsheet)

Die folgende Tabelle fasst die wichtigsten Administrationsbefehle für die tägliche Arbeit mit Docker zusammen:

Befehl	Kategorie	Zweck und betriebliche Wirkung
`docker run -d --name web -p 8080:80 nginx`	Lifecycle	Startet Container im Hintergrund mit Port-Weiterleitung von Host 8080 auf 80
`docker ps`	Monitoring	Listet alle aktuell laufenden Container mit ID, Image, Status und Ports auf
`docker ps -a`	Monitoring	Listet alle Container auf (inklusive beendeter Instanzen mit Exit-Codes)
`docker logs -f --tail 100 <container>`	Logging	Verfolgt stdout/stderr des Containers im Stream mit Begrenzung auf 100 Zeilen
`docker exec -it <container> sh`	Debugging	Öffnet eine interaktive Shell innerhalb des laufenden Container-Namespaces
`docker stop <container>`	Lifecycle	Sendet SIGTERM (nach Timeout SIGKILL) zum sauberen Herunterfahren
`docker rm <container>`	Lifecycle	Löscht gestoppte Container und gibt deren beschreibbaren Layer frei
`docker images`	Storage	Zeigt alle lokal gecachten Container-Images mit Tags und Größenangaben
`docker rmi <image>`	Storage	Entfernt ein Container-Image aus dem lokalen Cache (`/var/lib/docker`)
`docker system df`	Wartung	Zeigt den Speicherverbrauch von Images, Containern, Volumes und BuildKit

Befehl	Kategorie	Zweck und betriebliche Wirkung
`docker system prune -f`	Wartung	Bereinigt gestoppte Container, ungenutzte Netzwerke und ungetaggte Images
`docker compose up -d`	Orchestrierung	Baut und startet alle in `compose.yaml` definierten Services im Hintergrund
`docker compose down`	Orchestrierung	Stoppt und entfernt alle Container, Netzwerke und interne Links des Stacks

Weiterführende Ressourcen

Ressource	Link	Zweck und Inhalt
Docker Dokumentation	Docker Engine Docs	Offizielle Installationsanleitungen, Release Notes und API-Referenz
Docker Compose	Docker Compose Docs	Spezifikation und Dokumentation für Multi-Container-Orchestrierung
Ubuntu Server Guide	Ubuntu Server Guide	Offizielle Systemdokumentation für Ubuntu 26.04 LTS (Resolute Raccoon)
Server-Härtung 2026	Linux Server Härtung: SSH & CrowdSec	Leitfaden für Basishärtung, SSH mit FIDO2 und nftables-Firewalls
LEMP-Stack Ubuntu 26.04	Ubuntu 26.04 LEMP Leitfaden	Ergänzender Leitfaden für Nginx mit HTTP/3, MariaDB und PHP 8.5
Portainer CE Web-GUI	Portainer unter Ubuntu installieren	Grafische Verwaltung von Docker-Containern, Stacks und Volumes per Web-Interface

Fazit

Unter Ubuntu 26.04 LTS stellt Docker durch die native Unterstützung von `cgroups v2` und modernen Kernel-Overlays eine hochgradig ressourcenschonende Plattform für Serverdienste bereit. Im Gegensatz zu Standard-Installationen sorgt das Härten des Systems – insbesondere

Quelle: https://admindocs.de/de/serverumgebungen/ubuntu-26-04-docker-installieren-und-verwenden

die Begrenzung von Container-Logs über die `daemon.json`, die Aktivierung von `live-restore` und das Bewusstsein für die UFW-Bypass-Eigenschaft von iptables – für einen stabilen Produktionsbetrieb.

Im laufenden Betrieb sollten Administratoren vor allem zwei Parameter im Blick behalten: den Speicherzuwachs unter `/var/lib/docker` durch automatisierte Bereinigungs-Routinen (`docker system prune`) sowie die Vergabe von Bindings an Localhost (`127.0.0.1`), um interne Services vor unabsichtlicher Exposition im Internet zu schützen. Wer Docker-Container, Netzwerke und Multi-Container-Stacks statt über die Kommandozeile lieber über ein zentrales Web-Dashboard administrieren möchte, findet in unserem Leitfaden zu Portainer unter Ubuntu installieren die schlüsselfertige Anleitung.

Quelle: https://admindocs.de/de/serverumgebungen/ubuntu-26-04-docker-installieren-und-verwenden