ADMIN:DOCS

AlmaLinux: Docker und Docker Compose installieren und konfigurieren

Installiere Docker CE und Docker Compose auf AlmaLinux 9 und 10. Praxiserprobter Enterprise-Guide mit DNF, SELinux-Volume-Kontexten, firewalld und systemd.

Kategorie: Serverumgebungen **Veröffentlicht:** 21.10.2024 **Zuletzt geprüft:** 08.09.2026

AlmaLinux hat sich als binärkompatible Enterprise-Distribution im Red-Hat-Ökosystem fest etabliert. Während Red Hat im Standard auf den eigenen Container-Stack aus Podman, Buildah und Skopeo setzt, verlangen bestehende Entwicklungs-Pipelines, Third-Party-Tools und Multi-Container-Umgebungen in der Praxis häufig die originale Docker Engine (Docker CE) samt modernem Docker Compose V2.

Die Bereitstellung der Docker Engine auf Enterprise Linux unterscheidet sich jedoch spürbar von Debian- oder Ubuntu-Systemen. Das Zusammenspiel mit SELinux erfordert ein präzises Verständnis von Volume-Labeln (`:z` und `:Z`), die Paketfilterung über `firewalld` verlangt saubere Portfreigaben und vorinstallierte Podman-Pakete müssen vorab sauber aus dem System entfernt werden, um Konflikte um die Docker-Socket-Schnittstelle zu vermeiden.

In diesem Leitfaden richten wir Docker CE und das Docker Compose CLI-Plugin auf AlmaLinux 9 sowie dem modernen AlmaLinux 10 produktionsreif ein, härten den Daemon über die `daemon.json` und konfigurieren Netzwerk- sowie Speicherisolation für den fehlerfreien Betrieb.

Architektur und Enterprise-Besonderheiten

Unter AlmaLinux trifft eine traditionell auf Debian- und Ubuntu-Wurzeln fokussierte Container-Engine auf eine kompromisslose Enterprise-Architektur. Um spätere Laufzeitprobleme zu vermeiden, müssen drei Kernkomponenten verstanden werden:

Quelle: https://admindocs.de/de/serverumgebungen/alma-linux-wie-man-docker-und-docker-compose-installiert

Paketkonflikte mit Red-Hat-Tools:

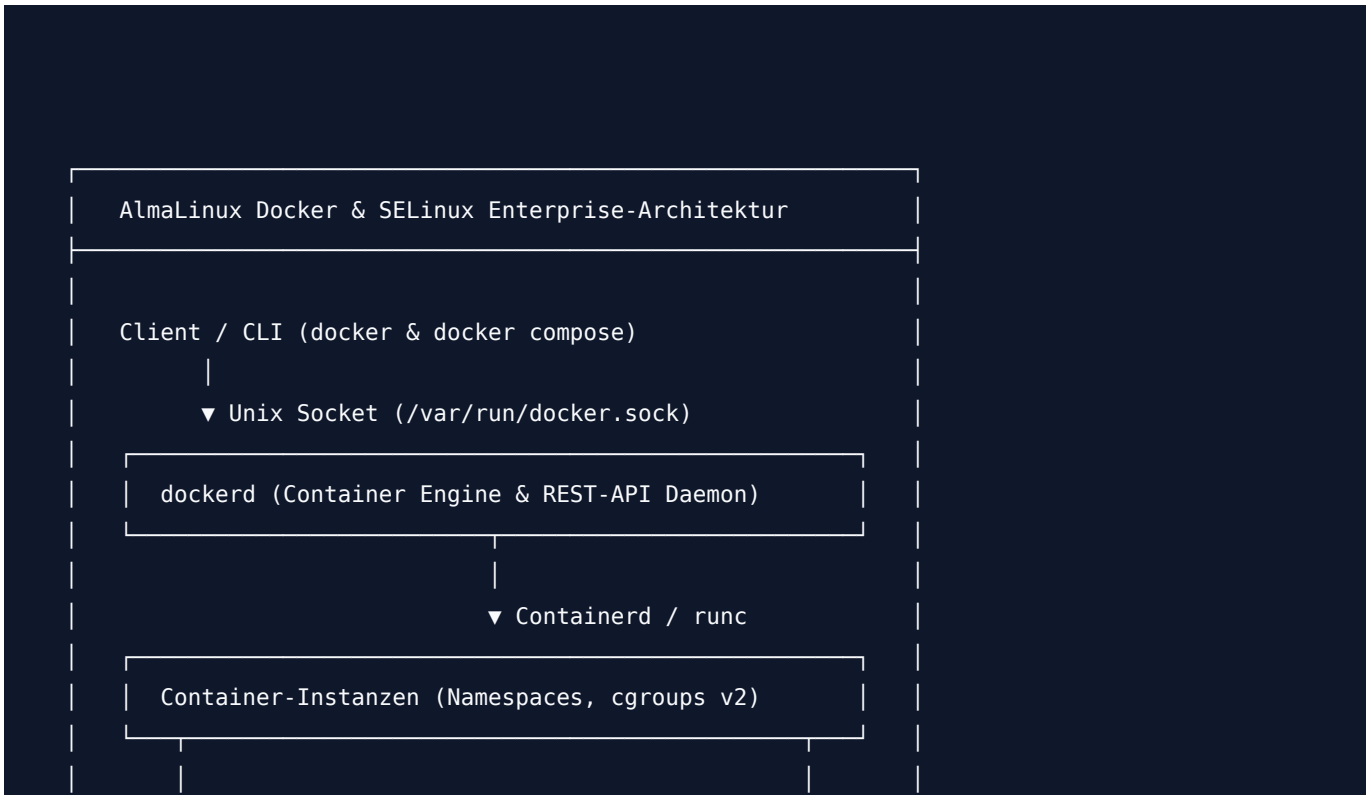
AlmaLinux liefert in seinen Standardquellen Podman und das Paket `crun` aus. Beide Tools stellen ähnliche Befehlsstrukturen und Socket-Dateien bereit. Werden diese Pakete nicht vorab entfernt, scheitert die DNF-Transaktion der offiziellen Docker-RPM-Pakete an Dateikollisionen.

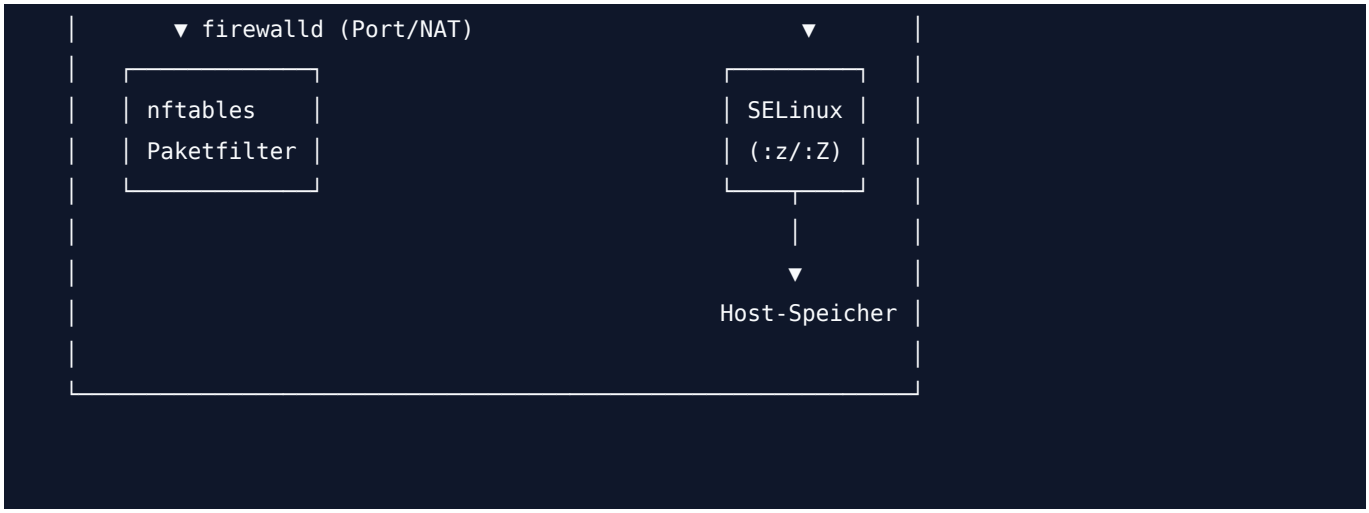
SELinux als Standard-Wächter:

SELinux läuft auf AlmaLinux standardmäßig im Modus `Enforcing`. Container laufen in einer restriktiven Sandbox (`container_t`). Greift ein Container über einen Bind-Mount auf Host-Verzeichnisse zu, blockiert der Linux-Kernel den Zugriff sofort mit `EACCES` (Permission Denied), sofern das Dateisystem-Ziel nicht den SELinux-Typ `container_file_t` besitzt.

Netzwerk-Interaktion mit firewalld:

Docker verwaltet Portweiterleitungen und Masquerading über direkte iptables- und nftables-Regeln. Ein Neustart oder Neuladen der Firewall (`firewall-cmd --reload`) flasht diese dynamischen Regeln aus dem Kernel, wodurch Container den Netzzugriff verlieren, bis der Docker-Dienst neu gestartet wird.





Die Docker Engine nutzt unter AlmaLinux standardmäßig `cgroups v2` und den Storage-Treiber `overlay2`. Beide Mechanismen sind im Enterprise-Kernel nativ integriert.

Voraussetzungen und Systemvorbereitung

Für die Installation benötigst du einen Server mit AlmaLinux 9 oder AlmaLinux 10, SSH-Zugriff sowie einen Benutzeraccount mit administrativen Rechten über `sudo`.

Kernel- und Distributionsstand prüfen

Überprüfe zunächst die installierte AlmaLinux-Version und den aktiven Linux-Kernel:

```
cat /etc/os-release
uname -r
```

AlmaLinux 9 setzt auf den Linux-Kernel 5.14, während AlmaLinux 10 auf modernen 6.x-Kernels basiert. Beide Varianten erfüllen sämtliche Anforderungen für Namespaces, Control Groups v2 und OverlayFS.

System aktualisieren

Bringe alle installierten Systempakete über den Paketmanager auf den aktuellen Stand:

```
sudo dnf upgrade -y
```

Falls während des Updates ein neuer Kernel eingespielt wurde, starte den Server einmalig neu:

```
sudo reboot
```

Vorinstallierte Container-Artefakte bereinigen

AlmaLinux bringt in vielen Installationsprofilen bereits Werkzeuge aus dem Podman-Ökosystem mit. Da diese dieselben Man-Pages, Socket-Namen und Befehls-Aliase beanspruchen können, deinstallierst du konkurrierende Pakete vollständig:

```
sudo dnf remove -y podman buildah skopeo docker docker-client docker-client-latest docker-common docker-latest docker-latest-logrotate docker-logrotate docker-engine
```

> □□ **Saubere Paketbasis:** Die Deinstallation von Podman und alten Docker-Paketen verhindert, dass DNF später beim Auflösen von Abhängigkeiten mit kryptischen Fehlermeldungen abbricht. Bestehende Konfigurationsdateien unter `/etc/` bleiben

> unberührt.

Offizielles Docker-Repository konfigurieren

Docker stellt für Enterprise-Linux-Systeme eigene RPM-Repositories bereit. Auf AlmaLinux und Rocky Linux wird hierfür das offizielle **CentOS-Repository** verwendet.

> ⚠ **CentOS- vs. RHEL-Repository:** Nutze auf AlmaLinux stets die Repository-URL `download.docker.com/linux/centos/docker-ce.repo`. Das Schwester-Repository `linux/rhel/` ist historisch für Red-Hat-Subskriptionssysteme gedacht und führt bei Minor-Point-Releases von AlmaLinux regelmäßig zu HTTP-404-Paketfehlern.

Repository auf AlmaLinux 9 einbinden (DNF4)

Unter AlmaLinux 9 wird DNF4 verwendet. Hier installieren wir das Paket `dnf-plugins-core`, um den Konfigurations-Manager zu aktivieren:

```
sudo dnf install -y dnf-plugins-core
sudo dnf config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
```

Repository auf AlmaLinux 10 einbinden (DNF5)

AlmaLinux 10 nutzt den neu geschriebenen Paketmanager DNF5. Die Syntax für externe Repositories wurde hier vereinfacht und erfordert kein separates Plugin-Paket mehr:

```
sudo dnf config-manager addrepo --from-
repofile=https://download.docker.com/linux/centos/docker-ce.repo
```

GPG-Schlüssel importieren und Cache aktualisieren

Importiere den offiziellen Signaturschlüssel von Docker, damit DNF die Authentizität aller heruntergeladenen RPM-Pakete kryptografisch verifizieren kann:

```
sudo rpm --import https://download.docker.com/linux/centos/gpg
sudo dnf makecache
```

Prüfe anschließend, ob das Repository aktiv in der Liste geführt wird:

```
dnf repolist | grep docker
```

Ausgabe: `docker-ce-stable Docker CE Stable - ...`

Installation von Docker Engine und Docker Compose V2

Nachdem die Paketquelle eingerichtet ist, installieren wir die vollständige Container-Toolchain. Wir verzichten bewusst auf veraltete Wrapper-Skripte oder manuelle Binärdownloads von GitHub und setzen stattdessen auf native RPM-Pakete.

Kernpakete installieren

Installiere die Docker Engine, die CLI, die Containerd-Laufzeitumgebung sowie die offiziellen CLI-Plugins für Buildx und Compose:

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

Die Rollen der einzelnen Pakete:

- **`containerd.io`**: Die standardisierte Container-Laufzeitumgebung (Runtime), die den Lebenszyklus von Containern auf Kernel-Ebene steuert.
- **`docker-ce`**: Der eigentliche Docker-Daemon (`dockerd`), der die REST-API bereitstellt und Netzwerke, Volumes sowie Images verwaltet.
- **`docker-ce-cli`**: Das Kommandozeilenwerkzeug `docker` zur Steuerung des Daemons.
- **`docker-buildx-plugin`**: Erweitert die CLI um moderne BuildKit-Funktionen für parallele und Multi-Architektur-Builds.
- **`docker-compose-plugin`**: Integriert Docker Compose V2 als nativen Unterbefehl (`docker compose`) direkt in die Docker-CLI.

Paketversionen verifizieren

Überprüfe nach Abschluss der Transaktion die installierten Softwareversionen:

```
rpm -qa | grep -E 'docker|containerd'
```

Erwartete Ausgabe (Beispiel):

```
containerd.io-1.7.25-3.1.el9.x86_64
docker-ce-27.3.1-1.el9.x86_64
docker-ce-cli-27.3.1-1.el9.x86_64
docker-compose-plugin-2.31.0-1.el9.x86_64
docker-buildx-plugin-0.19.1-1.el9.x86_64
```

Dienststeuerung und Autostart via systemd

Nach der Paketinstallation ist der Docker-Dienst unter Enterprise Linux standardmäßig weder gestartet noch für den Boot-Prozess aktiviert.

Docker-Dienst starten und aktivieren

Starte den Daemon und sorge mit dem Schalter `--now` dafür, dass die Systemd-Unit bei jedem Serverneustart automatisch hochfährt:

```
sudo systemctl enable --now docker
```

Überprüfe den Zustand des Daemons:

```
sudo systemctl status docker
```

Die Ausgabe muss `Active: active (running)` anzeigen. Überprüfe die Kernparameter der laufenden Engine mit `docker info`:

```
sudo docker info | grep -E 'Server Version|Storage Driver|Cgroup Version'
```

Ausgabe:

```
 Server Version: 27.3.1
 Storage Driver: overlay2
 Cgroup Version: 2
```

Nicht-Root-Zugriff einrichten

Standardmäßig gehört der Unix-Socket `/var/run/docker.sock` dem Benutzer `root` und der Gruppe `docker`. Um Docker-Befehle im Alltag ohne ständiges Voranstellen von `sudo` ausführen zu können, fügst du deinen Systembenutzer der Gruppe `docker` hinzu:

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

Damit die neue Gruppenberechtigung aktiv wird, führst du entweder `newgrp docker` aus oder meldest dich einmal über SSH ab und wieder an.

> ⚠ **Sicherheitsrisiko der Docker-Gruppe:** Wer Mitglied der Gruppe `docker` ist, besitzt effektive Root-Rechte auf dem Hostsystem. Über einen einfachen Bind-Mount von `/` in einen privilegierten Container lässt sich das Host-Dateisystem beliebig manipulieren.

> Vergib diese Berechtigung nur an vertrauenswürdige Administratoren und niemals an Dienst-Accounts.

Prüfe den Zugriff ohne administrative Zusatzrechte:

```
docker run --rm hello-world
```

Erscheint `Hello from Docker!`, kommuniziert deine unprivilegierte Shell fehlerfrei mit dem Daemon.

Enterprise-Härtung: SELinux und Volume-Mounts

Das Zusammenspiel zwischen Docker und SELinux ist auf AlmaLinux eine der häufigsten Fehlerquellen für Administratoren, die von Ubuntu oder Debian kommen.

Das Problem: EACCES bei Bind-Mounts

Wenn du ein Verzeichnis deines Hostsystems (z. B. ein Web-Root oder Konfigurationsdateien) in einen Container einbindest, greift die SELinux-Typenprüfung. Ein Prozess im Container besitzt den Kontext `container_t`. Das Verzeichnis auf dem Host besitzt standardmäßig einen Kontext wie `unconfined_u:object_r:user_home_t:s0` oder `var_t`.

Da `container_t` nicht auf allgemeine Benutzerdateien zugreifen darf, blockiert der Linux-Kernel den Zugriff sofort:

```
# Dieser Mount scheitert im Container bei aktivem SELinux:
```

```
docker run -d -p 8080:80 -v /srv/web/html:/usr/share/nginx/html:ro nginx:alpine
# Folge im Container-Log: "open() /usr/share/nginx/html/index.html failed (13: Permission
denied)"
```

Die Lösung: Die Volume-Flags `:z` und `:Z`

Docker bietet integrierte Schalter, die den SELinux-Kontext des gemounteten Host-Verzeichnisses automatisch anpassen:

- `:z` **(Shared):** Weist dem Verzeichnis den Kontext `container_file_t` zu. Mehrere voneinander unabhängige Container dürfen dieses Verzeichnis gleichzeitig lesen und beschreiben.
- `:Z` **(Private / Exclusive):** Weist dem Verzeichnis ein unikat-gelabeltes MCS-Label zu (`container_file_t:s0:c123,c456`). Ausschließlich dieser eine spezifische Container hat Zugriff. Kein anderer Container auf dem System kann auf diese Daten zugreifen.



□□ Praktisches Beispiel:

Erstelle ein Verzeichnis für statische HTML-Dateien und binde es mit dem Flag `:z` ein:

```
mkdir -p /srv/web/html
echo "<h1>AlmaLinux mit Docker und SELinux</h1>" > /srv/web/html/index.html

# Starte den Nginx-Container mit dem :z-Flag:
docker run -d --name webtest -p 8080:80 -v /srv/web/html:/usr/share/nginx/html:ro,z
nginx:alpine
```

Überprüfe den SELinux-Kontext des gemounteten Ordners auf dem Host:

```
ls -Zd /srv/web/html
```

Ausgabe: `system_u:object_r:container_file_t:s0 /srv/web/html`

Dank `container_file_t` greift Nginx im Container ohne Zugriffsfehler auf die Datei zu. Deaktiviere niemals SELinux (`setenforce 0`), nur um Berechtigungsprobleme zu umgehen – nutze stattdessen konsequent die Flags `:z` oder `:Z`.

firewalld-Integration und Netzwerkabsicherung

Unter AlmaLinux verwaltet `firewalld` die Paketfilterung des Hostsystems. Die Interaktion zwischen Docker und firewalld birgt eine technische Besonderheit, die Administratoren kennen müssen:

iptables-Bypass und Portfreigaben

Docker manipuliert bei aktiver Netzwerkunterstützung die `iptables`- und `nftables`-Tabellen des Kernels eigenständig (`nat`-Tabelle in der `PREROUTING`-Kette und `filter`-Tabelle in der `DOCKER`-Kette). Veröffentlichst du einen Port via `-p 8080:80`, greift diese Regel im Linux-Kernel noch vor den regulären Eingangsregeln der firewalld-Zonen.

Dennoch ist die explizite Freigabe in firewalld auf Enterprise-Systemen unerlässlich:

```
sudo firewall-cmd --permanent --add-port=8080/tcp
sudo firewall-cmd --reload
```

Der Firewall-Reload-Konflikt

> ⚠ **Achtung nach `firewall-cmd --reload`:** Wenn du `firewalld` im laufenden Betrieb neu lädst, flasht der Firewall-Daemon alle Kernel-Filtertabellen. Dabei werden die dynamisch von Docker injizierten Routing- und Masquerading-Regeln restlos entfernt. Laufende Container können danach weder das Internet erreichen noch eingehende Pakete empfangen.

Tritt nach einer Firewall-Änderung ein Netzwerkabbruch auf, startest du den Docker-Daemon neu, damit er seine Filterketten im Kernel neu aufbaut:

```
sudo systemctl restart docker
```

Bridge-Schnittstelle dauerhaft absichern

Damit firewalld den weitergeleiteten Netzwerkverkehr (Traffic Forwarding) über die virtuelle Docker-Bridge `docker0` nicht verwirft, ordnest du die Schnittstelle der Zone `trusted` zu:

```
sudo firewall-cmd --permanent --zone=trusted --add-interface=docker0
sudo firewall-cmd --reload
sudo systemctl restart docker
```

Produktions-Konfiguration: `/etc/docker/daemon.json`

Im Enterprise-Betrieb sollten die Standardeinstellungen der Docker Engine nicht ungeprüft übernommen werden. Unbegrenzt anwachsende Logdateien und Container-Abbrüche bei Daemon-Upgrades sind typische Betriebsrisiken.

Erstelle die zentrale Konfigurationsdatei `/etc/docker/daemon.json`:

```
sudo nano /etc/docker/daemon.json
```

Hinterlege die folgende gehärtete, validierte Konfiguration:

```
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "20m",
```

```
    "max-file": "5"
  },
  "live-restore": true,
  "userland-proxy": false
}
```

Erklärung der Einstellungen:

- **`log-opts (max-size & max-file)`**: Begrenzt die JSON-Logdateien jedes Containers auf maximal 20 MB und rotiert über 5 Dateien. Dies verhindert verlässlich, dass fehlerhafte oder geschwätzige Container die Root-Partition füllen.
- **`live-restore: true`**: Ermöglicht es der Docker Engine, laufende Container am Leben zu halten, wenn der `dockerd`-Prozess neu gestartet oder per `dnf upgrade` aktualisiert wird. Dies minimiert Ausfallzeiten bei Wartungsarbeiten drastisch.
- **`userland-proxy: false`**: Schaltet den zusätzlichen Hilfsprozess `docker-proxy` ab. Der Datenverkehr wird stattdessen performant und ressourcenschonend direkt über Kernel-NAT (iptables/nftables) geroutet.

Wichtiger Härtungshinweis zu no-new-privileges:

Häufig wird empfohlen, die Option `no-new-privileges` zu aktivieren, um Privilege-Escalation über `setuid`- oder `setgid`-Binaries in Containern zu blockieren. Diese Option existiert im Docker-Ökosystem jedoch **nicht** als globaler Schlüssel in der `daemon.json` (der Daemon würde den Start mit einer Fehlermeldung verweigern). Sie wird stattdessen pro Container via `--security-opt=no-new-privileges:true` oder im Compose-File deklariert.

Lade die neue Konfiguration durch einen Neustart des Daemons:

```
sudo systemctl restart docker
```

Verifiziere die aktiven Einstellungen:

```
docker info | grep -E 'Logging Driver|Live Restore'
```

Praxiseinsatz mit Docker Compose V2

Docker Compose V2 ist als CLI-Plugin vollständig in den Docker-Befehlssatz integriert. Der Aufruf erfolgt modern über `docker compose` mit Leerzeichen (statt des veralteten Python-Tools `docker-compose` mit Bindestrich).

Multi-Container-Projekt anlegen

Erstelle ein Projektverzeichnis für einen typischen Web-Stack aus einem Nginx-Frontend und einem Redis-Cache:

```
mkdir -p ~/webstack/html
cd ~/webstack

echo "<h1>Produktions-Stack auf AlmaLinux</h1>" > html/index.html
```

Erstelle die Konfigurationsdatei `docker-compose.yml`:

```
nano docker-compose.yml
```

Füge folgenden Stack-Inhalt ein. Beachte das `:ro,z`-Flag am Host-Volume für SELinux:

```
name: enterprise-stack

services:
  web:
    image: nginx:alpine
    container_name: web_frontend
    ports:
      - "8080:80"
    volumes:
      - ./html:/usr/share/nginx/html:ro,z
    security_opt:
      - no-new-privileges:true
    restart: unless-stopped
    healthcheck:
      test: ["CMD-SHELL", "wget -q --spider http://127.0.0.1/ || exit 1"]
      interval: 15s
      timeout: 5s
      retries: 3

  cache:
    image: redis:alpine
    container_name: redis_cache
    command: redis-server --appendonly yes
    volumes:
      - redis_data:/data
    security_opt:
      - no-new-privileges:true
    restart: unless-stopped

volumes:
  redis_data:
```

Stack starten und überwachen

Starte den gesamten Stack im Hintergrund:

```
docker compose up -d
```

Docker Compose lädt die Images herunter, erstellt das isolierte Netzwerk `enterprise-stack_default`, richtet das benannte Volume `redis_data` ein und startet die Container.

Überprüfe den Zustand der Dienste:

```
docker compose ps
```

Erwartete Ausgabe:

```
NAME            IMAGE          COMMAND                  SERVICE   CREATED
STATUS                    PORTS
redis_cache     redis:alpine   "docker-entrypoint.s…"   cache     15 seconds ago   Up 14
seconds            6379/tcp
web_frontend    nginx:alpine   "/docker-entrypoint.…"   web       15 seconds ago   Up 14
seconds (healthy)   0.0.0.0:8080->80/tcp
```

Prüfe den Zugriff auf den Webserver über die Kommandozeile:

```
curl http://127.0.0.1:8080
```

Ausgabe: `<h1>Produktions-Stack auf AlmaLinux</h1>`

Beende den Stack bei Bedarf wieder geordnet:

```
docker compose down
```

Wartung, Updates und Troubleshooting

Ein stabiler Produktivbetrieb erfordert standardisierte Wartungsabläufe und schnelles Eingreifen bei Fehlern.

Updates der Docker Engine einspielen

Da wir die offizielle Paketquelle eingebunden haben, werden Updates der Docker Engine und des Compose-Plugins nahtlos über den Systempaketmanager eingespielt:

```
# Auf AlmaLinux 9:
sudo dnf check-update | grep docker
sudo dnf upgrade -y docker-ce docker-ce-cli containerd.io docker-compose-plugin

# Auf AlmaLinux 10:
sudo dnf upgrade -y 'docker-*' containerd.io
```

Dank der Konfiguration `"live-restore": true` in der `daemon.json` laufen deine bestehenden Container während des Daemon-Updates ohne Unterbrechung weiter.

SELinux-Verweigerungen analysieren

Wenn ein Container unerwartet Berechtigungsfehler wirft, prüfe das Audit-Log auf SELinux-AVCs:

```
sudo ausearch -m avc -ts recent
```

Wird dort `denied { read }` für einen Container gemeldet, fehlt am Volume-Mount das passende `:z`-Flag oder das Verzeichnis benötigt eine explizite Kontext-Zuweisung. Für eine schnelle temporäre Anpassung genügt `chcon`:

```
sudo chcon -Rt container_file_t /pfad/zum/verzeichnis
```

Soll der SELinux-Typ dauerhaft persistent hinterlegt werden (sodass auch ein System-Relabeling via `restorecon` die Berechtigung nicht überschreibt), nutzt du `semanage`:

```
sudo semanage fcontext -a -t container_file_t "/pfad/zum/verzeichnis(/.*)?"
sudo restorecon -Rv /pfad/zum/verzeichnis
```

Nicht genutzte Ressourcen bereinigen

Alte Container-Images, verwaiste Build-Caches und ungenutzte Netzwerke belegen mit der Zeit wertvollen Flash-Speicher. Bereinige diese Artefakte kontrolliert:

```
docker system prune -a --volumes -f
```

Befehlsreferenz (Cheatsheet)

Die zentralen Befehle für die Verwaltung von Docker und Compose unter AlmaLinux im Überblick:

Befehl	Zweck	Kontext
`sudo dnf install -y docker-ce docker-compose-plugin`	Docker Engine und Compose V2 installieren	Paketmanager
`sudo systemctl enable --now docker`	Docker-Daemon starten und für Autostart aktivieren	systemd
`sudo usermod -aG docker $USER`	Benutzer zur Docker-Gruppe hinzufügen	Benutzerverwaltung
`docker info`	Vollständige System- und Storage-Treiber-Details anzeigen	Diagnose
`docker run --rm -v ./data:/app:z image`	Host-Verzeichnis mit SELinux-Shared-Label einbinden	Container-Start
`sudo firewall-cmd --permanent --add-port=8080/tcp`	Container-Port in der Host-Firewall freigeben	firewalld
`sudo firewall-cmd --reload && sudo systemctl restart docker`	Firewall neu laden und Docker-Netzwerk regenerieren	Netzwerkpflege
`docker compose up -d`	Multi-Container-Stack im Hintergrund starten	Compose
`docker compose ps`	Status aller Services und Healthchecks prüfen	Compose

Befehl	Zweck	Kontext
`docker compose logs -f --tail=50`	Echtzeit-Logs des aktiven Stacks verfolgen	Fehlersuche
`docker compose down`	Stack geordnet stoppen und Netzwerke entfernen	Compose
`docker system prune -f`	Ungenutzte Images und Build-Caches bereinigen	Festplattenpflege

Weiterführende Ressourcen

Offizielle Dokumentationen und Dokumente zur Vertiefung:

Ressource	Beschreibung	Typ
Docker Engine Dokumentation	Offizielles Handbuch für Installation, Konfiguration und Storage	Dokumentation
Docker Compose Spezifikation	Referenz für Compose-YAML-Dateien und Service-Attribute	Referenz
AlmaLinux Wiki	Offizielle Administrationsleitfäden und Release-Dokumentationen	Dokumentation
Red Hat SELinux Guide	Tiefgehende Dokumentation zur Mandatory Access Control unter RHEL	Referenz
firewalld Dokumentation	Zonen-, Schnittstellen- und Policy-Management für Linux-Firewalls	Dokumentation

Fazit

Mit dieser Bereitstellung betreibst du eine stabile, wartungsarme und voll enterprise-taugliche Docker-Umgebung auf AlmaLinux. Durch die Verwendung des offiziellen CentOS-Repositories erhältst du verlässliche Updates, während das native Compose-CLI-Plugin moderne Multi-Container-Deployments ohne externe Abhängigkeiten ermöglicht.

Quelle: https://admindocs.de/de/serverumgebungen/alma-linux-wie-man-docker-und-docker-compose-installiert

Die Berücksichtigung der Enterprise-Besonderheiten – insbesondere das konsequente Setzen der SELinux-Mount-Flags `:z` und die saubere Portfreigabe in `firewalld` – verhindert die typischen Fallstricke, an denen unvorbereitete Setups in RHEL-basierten Umgebungen scheitern.

> **Praxistipp für den Produktivbetrieb:** Konfiguriere die `/etc/docker/daemon.json` vor dem ersten produktiven Container-Start. Parameter wie `log-opts` und `live-restore` schützen dein System dauerhaft vor volllaufenden Dateisystemen und ermöglichen unterbrechungsfreie Daemon-Updates während regulärer Wartungsfenster.