Utrzymanie hosta dockerowego

Poziom: operacyjny **Dotyczy: Docker Engine 24+, RHEL/CentOS Stream**

Czas: ~35 min

Host dockerowy zostawiony bez przeglądu ma dwa typowe końce — pełny dysk pod `/var/lib/docker` albo kontener, który zjada całą dostępną pamięć hosta, bo nikt mu nie ustawił limitu.

Skróty użyte na tej stronie (5)

Skrót	Co znaczy
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
GB	Gigabyte: miliard bajtów, jednostka pojemności pamięci i plików, tu najczęściej rozmiar modelu albo pamięci karty graficznej
JSON	JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami
OOM	Out Of Memory: brak pamięci; OOM killer to mechanizm jądra zabijający proces, gdy jej zabraknie
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa

Fiszka

Komendy wykonujemy z CLI bash z uprawnieniami root (sudo).

ZMIENIA SYSTEM

Instalacja Docker Engine z repozytorium

> W naszym środowisku dane aplikacji trzymamy w `/docker/apps/<nazwa>`, a plik Compose w `/docker/compose/<nazwa>/compose.yaml`. To konwencja, nie wymóg Dockera: można ją zmienić, byle w całej firmie była jedna. Katalog `/docker` to osobny wolumen

`vgDocker/lvDocker`, żeby pełny dysk aplikacji nie zatrzymał systemu. Pełny opis konwencji, razem z instalacją tego wolumenu od zera, na stronach Jak trzymamy aplikacje w Dockerze: konwencje Zielonej i Instalacja Dockera na Rocky Linux 9 z osobnym wolumenem /docker.

Sekwencja ma sens wykonana w całości i w tej kolejności — repozytorium przed instalacją pakietu.

```
# Dodanie oficjalnego repozytorium Dockera (RHEL)
sudo dnf config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo

# Na systemach z rodziny CentOS Stream repozytorium ma inny adres:
# sudo dnf config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

# Instalacja silnika razem z wtyczkami buildx i compose
sudo dnf install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

dnf = Dandified YUM: menedżer pakietów RHEL/Rocky

TYLKO ODCZYT

Edycja pliku konfiguracyjnego demona: vim

```
vim /etc/docker/daemon.json
```

Tworzymy albo edytujemy plik konfiguracyjny demona.

vim = Vi IMproved: ulepszony edytor vi

TYLKO ODCZYT

Wklejenie konfiguracji nowej lokalizacji danych: data-root

```
{
  "storage-driver": "overlay2",
  "data-root": "/docker/docker-data"
}
```

Wklejamy konfigurację wskazującą nową lokalizację.

> Sterownik pamięci masowej deklarujemy zawsze jako `overlay2` — starszy `overlay` (bez dwójki) nie jest już wspierany przez aktualne wersje silnika.

ZMIENIA SYSTEM

Restart usługi po zmianie konfiguracji: systemctl restart

```
sudo systemctl restart docker
```

Restart usługi, żeby demon wczytał nową konfigurację.

systemctl = system control: sterowanie usługami systemd

ZMIENIA SYSTEM

Włączenie usługi na świeżej instalacji: systemctl enable --now

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

Jeśli instalacja jest świeża i usługa jeszcze nie była uruchamiana, włączamy ją zamiast restartować.

systemctl = system control: sterowanie usługami systemd

TYLKO ODCZYT

Sprawdzenie nowej lokalizacji danych: docker info

```
docker info -f '{{ .DockerRootDir }}'
```

Sprawdzamy, czy Docker faktycznie używa nowej lokalizacji.

Przykład:

```
/docker/docker-data
```

docker = narzędzie do kontenerów

□□ ZMIENIA SYSTEM

Test poprawności instalacji: docker run hello-world

```
sudo docker run hello-world
```

Uruchamiamy kontener testowy, żeby sprawdzić poprawność instalacji.

Przykład:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
Digest: sha256:926fac19d22aa2d60f1a276b66a20eb765fbeea2db5dbdaafeb45ad8ce815d8
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

□□ **docker** = narzędzie do kontenerów

□□ TYLKO ODCZYT

Sprawdzenie łącznej zajętości katalogu danych: du -sh

```
sudo du -sh /docker/docker-data
```

Przykład:

```
28G□/docker/docker-data
```

□□ **du** = disk usage: ile miejsca zajmuje katalog

□□ TYLKO ODCZYT

Rozbicie zajętości na kategorie przez Dockera: docker system df -v

```
docker system df -v
```

Flaga `-v` dokłada do standardowego podsumowania (patrz Codzienne komendy) listę pojedynczych obrazów, kontenerów i wolumenów z ich indywidualnym rozmiarem — przydatne, gdy samo zbiorcze podsumowanie nie mówi, który konkretnie element warto

usunąć.

docker = narzędzie do kontenerów

OSTROŻNIE · NIEODWRACALNE

Usunięcie nieużywanych zasobów agresywnie: docker system prune -a --volumes

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

Flaga `-a` dokłada do kasowania wszystkie obrazy niewykorzystywane przez żaden aktualnie działający kontener — łącznie z obrazami oznaczonymi tagiem, które po prostu nie mają w tej chwili uruchomionego kontenera, ale mogą być potrzebne za chwilę (np. poprzednia wersja przed wdrożeniem). Flaga `--volumes` dokłada kasowanie wolumenów nienależących do żadnego istniejącego kontenera — w tym danych, które ktoś zostawił celowo, licząc na ich trwałość między wdrożeniami.

> Komenda z obiema flagami naraz jest nieodwracalna i nie rozróżnia „nieużywane, bo śmieć" od „nieużywane teraz, ale potrzebne za godzinę" — na hoście produkcyjnym uruchamiamy ją wyłącznie po ręcznym przejrzeniu listy z `docker system df -v`, nigdy odruchowo.

docker = narzędzie do kontenerów

TYLKO ODCZYT

Ograniczenie rozmiaru logów globalnie: log-driver w daemon.json

```
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
  }
}
```

Ograniczenie rozmiaru i liczby przechowywanych plików logu globalnie, dla wszystkich nowych kontenerów, przez `/etc/docker/daemon.json`. Po restarcie demona każdy nowo

tworzony kontener rotuje log po osiągnięciu 10 MB, trzymając maksymalnie 3 pliki historii — ustawienie nie działa wstecz na kontenery już istniejące, tylko na nowo tworzone.

□□ ZMIENIA SYSTEM

Ograniczenie rozmiaru logów pojedynczego kontenera: docker run --log-opt

```
docker run -d --log-opt max-size=10m --log-opt max-file=3 NazwaObrazu
```

Ten sam limit da się ustawić pojedynczemu kontenerowi bez zmiany globalnej konfiguracji.

□□ **docker** = narzędzie do kontenerów

□□ ZMIENIA SYSTEM

Ograniczenie pamięci dostępnej dla kontenera: docker run --memory

```
docker run -d --memory=512m NazwaObrazu
```

Proces wewnątrz kontenera, który spróbuje przekroczyć ten limit, zostaje zabity przez OOM killer jądra — nie jest to łagodne ograniczenie, tylko twarda granica kończąca się zakończeniem procesu.

□□ **OOM** = Out Of Memory: brak pamięci; OOM killer to mechanizm jądra zabijający proces, gdy jej zabraknie · **docker** = narzędzie do kontenerów

□□ ZMIENIA SYSTEM

Ograniczenie liczby rdzeni CPU dostępnych dla kontenera: docker run --cpus

```
docker run -d --cpus=1.5 NazwaObrazu
```

Wartość ułamkowa oznacza limit w jednostkach rdzenia — `1.5` pozwala kontenerowi zużyć maksymalnie półtora rdzenia łącznego czasu procesora, niezależnie od tego, ile rdzeni ma cały host.

□□ **CPU** = Central Processing Unit: procesor · **docker** = narzędzie do kontenerów

□□ TYLKO ODCZYT

Sprawdzenie ustawionych limitów działającego kontenera: docker inspect

```
docker inspect NazwaKontenera --format '{{ .HostConfig.Memory }} {{ .HostConfig.NanoCpus }}'
```

□□ **docker** = narzędzie do kontenerów

□□ TYLKO ODCZYT

Sprawdzenie, które kontenery wrócą same po restarcie: docker ps -a

```
docker ps -a --format 'table {{.Names}}\t{{.Status}}' --filter 'label!=tymczasowy'
```

Przy planowaniu utrzymania hosta (np. przed zaplanowanym restartem po aktualizacji jądra) sprawdzamy z wyprzedzeniem, które kontenery faktycznie wrócą same.

□□ **docker** = narzędzie do kontenerów

□□ TYLKO ODCZYT

Sprawdzenie aktualnie zainstalowanej wersji: docker version

```
docker version --format '{{.Server.Version}}'
```

□□ **docker** = narzędzie do kontenerów

□□ ZMIENIA SYSTEM

Aktualizacja pakietu silnika: dnf update

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

Aktualizujemy pakiet z tego samego repozytorium, z którego silnik został zainstalowany.

□□ **dnf** = Dandified YUM: menedżer pakietów RHEL/Rocky

Z życia wzięte

Z ŻYCIA WZIĘTE

W piątek wieczorem monitoring budzi Tomka alarmem o pełnym dysku na `docker01`, mimo że nikt niczego tego dnia nie wdrażał. Okazuje się, że kontener Grayloga loguje bardzo intensywnie od tygodnia bez limitu rozmiaru w `daemon.json`, a plik logu urósł do kilkunastu gigabajtów. Po dodaniu `max-size` i restarcie demona miejsce wraca, ale dopiero po ręcznym usunięciu starego pliku logu.

Restart usługi po aktualizacji: systemctl restart

```
sudo systemctl restart docker
```

Restartujemy usługę po aktualizacji.

> Aktualizacja restartuje demona, co zatrzymuje wszystkie kontenery bez ustawionej polityki automatycznego wznowienia — przed aktualizacją produkcyjnego hosta sprawdzamy dokładnie tą samą komendą co wyżej, które kontenery wymagają ręcznego `docker start` po restarcie.

Zmiana domyślnej lokalizacji plików Dockera

Pojęcie: `data-root`. Katalog, w którym demon Dockera trzyma wszystkie obrazy, kontenery i wolumeny nazwane — domyślnie `/var/lib/docker`. Zmiana tej ścieżki ma sens, gdy dysk systemowy jest mały, a dane Dockera mają rosnąć na osobnym, większym wolumenie.

Ryzyko system prune -a --volumes

Pojęcie: zakres `docker system prune`. Sama komenda bez flag kasuje wyłącznie to, co bezpiecznie nieużywane — zatrzymane kontenery, nieoznaczone obrazy, nieużywane sieci i cache budowania. Każda dodatkowa flaga poszerza ten zakres o coś, co bywa wciąż potrzebne.

Rotacja logów kontenerów

Pojęcie: sterownik logowania `json-file`. Domyślny sposób, w jaki Docker przechowuje standardowe wyjście i wyjście błędów kontenera — każda linia trafia jako wpis JSON do pliku na dysku hosta. Bez limitu rozmiaru plik logu kontenera, który działa miesiącami i intensywnie loguje, potrafi urosnąć do rozmiaru liczonego w dziesiątkach gigabajtów.

Limity pamięci i CPU

Pojęcie: limit zasobów kontenera. Bez jawnie ustawionego limitu kontener może zużyć całą dostępną pamięć albo cały dostępny czas procesora hosta — mechanizm `cgroups` jądra Linuksa, z którego korzysta Docker, pozwala ograniczyć oba te zasoby per kontener.



Docker ogranicza zasoby kontenera przez cgroups jądra, a OOM killer zabija proces dopiero po przekroczeniu limitu pamięci (kliknięcie otwiera pełny rozmiar)

Polityki restartu w kontekście utrzymania

Polityka restartu (opisana jako flaga `--restart` na stronie Codzienne komendy) ma znaczenie utrzymaniowe wykraczające poza pojedynczy kontener — przy restarcie hosta demon Dockera odtwarza wszystkie kontenery z polityką `always` albo `unless-stopped` automatycznie, bez interwencji, natomiast kontenery uruchomione bez tej flagi (albo z domyślnym `no`) zostają po restarcie w stanie zatrzymanym i wymagają ręcznego `docker start`.

Częste błędy

Zmiana `data-root` bez zatrzymania i przeniesienia istniejących danych. Sama zmiana ścieżki w `daemon.json` na hoście, który już ma obrazy i wolumeny pod starą lokalizacją, sprawia, że po restarcie Docker widzi pusty katalog danych — stare dane wciąż fizycznie leżą pod starą ścieżką, ale demon o nich nie wie, dopóki nie zostaną tam skopiowane ręcznie przed restartem.

Traktowanie `docker system prune -a --volumes` jako rutynowego sprzątania. To jest operacja do świadomego, jednorazowego użycia po przejrzeniu listy z `docker system df -v`, nie polecenie do wrzucenia w cykliczne zadanie `cron` bez nadzoru.

Brak limitu pamięci na kontenerach na współdzielonym hoście. Jeden kontener bez `--memory` potrafi w razie wycieku pamięci zabrać zasoby wszystkim pozostałym kontenerom na tym samym hoście, zanim OOM killer w ogóle zainterweniuje.

□□ Ćwiczenia

> Zadania sprawdzające.

1. Instalujemy Docker Engine z oficjalnego repozytorium na świeżym hoście RHEL i zmieniamy domyślną lokalizację danych na `/docker/docker-data`, weryfikując zmianę.
2. Wyjaśniamy, dlaczego `docker system prune -a --volumes` jest ryzykowniejsze niż `docker system prune` bez flag, i opisujemy, co sprawdzamy przed jego uruchomieniem.
3. Kontener loguje bardzo intensywnie i po tygodniu jego plik logu zajmuje 4 GB. Piszemy fragment `daemon.json`, który zapobiegnie temu w przyszłości dla nowych kontenerów.
4. Host ma zaplanowany restart po aktualizacji jądra. Opisujemy, jak sprawdzimy z wyprzedzeniem, które kontenery wrócą same, a które będą wymagać ręcznego uruchomienia.

Powiązane: Obraz, kontener, warstwa, wolumen: cztery pojęcia · Codzienne komendy · Docker Compose · Sieci kontenerów i SNAT

Revision #39
Created 2025-05-05 13:00:19 UTC by Tomasz Konieczny
Updated 2026-10-26 14:09:28 UTC by Tomasz Konieczny