Codzienne komendy Dockera

Poziom: podstawy **Dotyczy: Docker Engine 24+, Docker CLI** **Czas: ~30 min**

Komendy wykonujemy z CLI bash jako user z uprawnieniami sudo albo jako user dodany do grupy `docker` — bez jednego z tych dwóch warunków każde polecenie kończy się błędem `permission denied` na gnieździe demona.

Skróty użyte na tej stronie (6)

Skrót	Co znaczy
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
CPU	Central Processing Unit: procesor
GB	Gigabyte: miliard bajtów, jednostka pojemności pamięci i plików, tu najczęściej rozmiar modelu albo pamięci karty graficznej
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
JSON	JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami
NAT	Network Address Translation: podmiana adresów IP na granicy sieci

Fiszka

TYLKO ODCZYT

Wylistowanie uruchomionych kontenerów: docker ps

```
docker ps
```

Zwracamy listę aktualnie działających kontenerów wraz z ich identyfikatorem, obrazem źródłowym, czasem działania i zmapowanymi portami.

Przykład:

```
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS
PORTS                    NAMES
a1b2c3d4e5f6   nginx:latest   "/docker-entrypoint.…"   3 hours ago     Up 3 hours
0.0.0.0:8080->80/tcp     app_web
f6e5d4c3b2a1   postgres:15    "docker-entrypoint.s…"   3 hours ago     Up 3 hours
5432/tcp                 app_db
```

□□ **docker** = narzędzie do kontenerów

□□ TYLKO ODCZYT

Wylistowanie wszystkich kontenerów, łącznie z zatrzymanymi: docker ps -a

```
docker ps -a
```

Dokładamy do wyniku kontenery w stanie `Exited` — przydatne przy szukaniu kontenera, który się zatrzymał albo nigdy poprawnie nie wystartował.

Przykład:

```
CONTAINER ID   IMAGE          COMMAND                  CREATED
STATUS                     PORTS     NAMES
a1b2c3d4e5f6   nginx:latest   "/docker-entrypoint.…"   3 hours ago     Up 3
hours                  80/tcp    app_web
9f8e7d6c5b4a   redis:7        "docker-entrypoint.s…"   1 day ago       Exited (137) 2
hours ago              app_cache_old
```

□□ **docker** = narzędzie do kontenerów

□□ TYLKO ODCZYT

Wylistowanie pobranych obrazów: docker images

```
docker images
```

Zwracamy obrazy dostępne lokalnie, z ich rozmiarem i identyfikatorem — rozmiar dotyczy unikalnych warstw obrazu, nie sumy warstw współdzielonych z innymi obrazami.

Przykład:

```
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
nginx        latest    a6bd71f48f68   2 weeks ago     192MB
postgres     15        4c0dcb1c497b   3 weeks ago     412MB
```

docker = narzędzie do kontenerów

ZMIENIA SYSTEM

Uruchomienie kontenera w tle: -d

```
docker run -d nginx:latest
```

Uruchamiamy kontener w tle (detached) i od razu odzyskujemy kontrolę nad terminalem — bez tej flagi proces kontenera zajmuje terminal, w którym wpisaliśmy komendę, dopóki go nie zatrzymamy.

Przykład:

```
a1b2c3d4e5f61234567890abcdef1234567890abcdef1234567890abcdef12
```

docker = narzędzie do kontenerów

ZMIENIA SYSTEM

Uruchomienie kontenera z interaktywnym terminalem: -it

```
docker run -it ubuntu:22.04 bash
```

Łączymy dwie flagi naraz — `-i` utrzymuje otwarte standardowe wejście, `-t` przydziela pseudoterminal — dzięki czemu dostajemy działającą interaktywną powłokę wewnątrz kontenera, tak jakbyśmy się do niego zalogowali.

Przykład:

```
root@a1b2c3d4e5f6:/#
```

docker = narzędzie do kontenerów

ZMIENIA SYSTEM

Zmapowanie portu kontenera na port hosta: -p

```
docker run -d -p 8080:80 nginx:latest
```

Publikujemy port `80` kontenera pod portem `NumerPortu` hosta (tutaj `8080`) — ruch przychodzący na host trafia do kontenera przez regułę NAT dodaną przez Dockera do `iptables`. Bez tej flagi port kontenera jest osiągalny wyłącznie z sieci dockerowej, nie z zewnątrz hosta.

Przykład:

```
CONTAINER ID   IMAGE          PORTS                    NAMES
a1b2c3d4e5f6   nginx:latest   0.0.0.0:8080->80/tcp   app_web
```

NAT = Network Address Translation: podmiana adresów IP na granicy sieci · **docker** = narzędzie do kontenerów

ZMIENIA SYSTEM

Podpięcie wolumenu do kontenera: -v

> 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 na stronie Jak trzymamy aplikacje w Dockerze: konwencje Zielonej.

```
docker run -d -v NazwaWolumenu:/var/lib/postgresql/data postgres:15
```

Podpinamy wolumen nazwany pod ścieżkę wewnątrz kontenera — dane zapisywane pod tą ścieżką przeżywają usunięcie kontenera. Ta sama flaga obsługuje też bind mounty, gdy zamiast nazwy wolumenu podamy ścieżkę bezwzględną z hosta (`/SciezkaDoPliku:/var/lib/postgresql/data`).

Przykład:

```
CONTAINER ID   IMAGE          MOUNTS
a1b2c3d4e5f6   postgres:15    NazwaWolumenu
```

docker = narzędzie do kontenerów

◫◫ ZMIENIA SYSTEM

Nadanie kontenerowi nazwy: --name

```
docker run -d --name NazwaKontenera nginx:latest
```

Nadajemy kontenerowi czytelną nazwę zamiast losowo wygenerowanej — dzięki temu w kolejnych komendach (`docker logs`, `docker exec`, `docker stop`) odwołujemy się do niego po nazwie zamiast po identyfikatorze, który trzeba by najpierw sprawdzić.

Przykład:

```
CONTAINER ID   IMAGE          NAMES
a1b2c3d4e5f6   nginx:latest   NazwaKontenera
```

◫◫ **docker** = narzędzie do kontenerów

◫◫ ZMIENIA SYSTEM

Ustawienie polityki automatycznego restartu: --restart

```
docker run -d --restart unless-stopped NazwaObrazu
```

Deklarujemy, że demon Dockera ma sam ponownie uruchomić kontener po jego awarii albo po restarcie hosta — wariant `unless-stopped` robi to zawsze, chyba że kontener został zatrzymany ręcznie przez `docker stop`; wariant `always` ignoruje nawet ręczne zatrzymanie i uruchomi kontener ponownie przy starcie demona; wariant `on-failure` restartuje tylko po zakończeniu z niezerowym kodem wyjścia.

Przykład:

```
CONTAINER ID   IMAGE         STATUS
a1b2c3d4e5f6   NazwaObrazu   Up 2 seconds (health: starting)
```

◫◫ **docker** = narzędzie do kontenerów

◫◫ ZMIENIA SYSTEM

Podłączenie kontenera do konkretnej sieci: --network

```
docker run -d --network NazwaSieci NazwaObrazu
```

Podpinamy kontener do wskazanej sieci użytkownika zamiast domyślnej sieci bridge — kontenery w tej samej sieci użytkownika widzą się nawzajem po nazwie, bez znajomości adresów IP. Więcej o rodzajach sieci na stronie Sieci kontenerów i SNAT.

Przykład:

```
CONTAINER ID   IMAGE          NETWORKS
a1b2c3d4e5f6   NazwaObrazu    NazwaSieci
```

IP = Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection) · **SNAT** = Source NAT: podmiana adresu nadawcy w ruchu wychodzącym · **docker** = narzędzie do kontenerów

ZMIENIA SYSTEM

Ustawienie zmiennej środowiskowej w kontenerze: -e

```
docker run -d -e POSTGRES_PASSWORD=HasloUzytkownika postgres:15
```

Przekazujemy zmienną środowiskową do procesu wewnątrz kontenera — standardowy sposób konfiguracji obrazów gotowych (baz danych, kolejek), które odczytują parametry startowe właśnie ze zmiennych środowiskowych, nie z pliku konfiguracyjnego.

Przykład:

```
docker exec NazwaKontenera env | grep POSTGRES_PASSWORD
```

```
POSTGRES_PASSWORD=HasloUzytkownika
```

docker = narzędzie do kontenerów

OSTROŻNIE · NIEODWRACALNE

Automatyczne usunięcie kontenera po zakończeniu: --rm

```
docker run --rm -it ubuntu:22.04 bash
```

Kontener znika automatycznie zaraz po zakończeniu procesu — wygodne przy jednorazowych, testowych uruchomieniach, żeby nie zaśmiecać `docker ps -a` dziesiątkami zakończonych kontenerów, o których i tak zapomnimy.

docker = narzędzie do kontenerów

TYLKO ODCZYT

Wykonanie komendy wewnątrz działającego kontenera: docker exec

```
docker exec -it NazwaKontenera bash
```

Uruchamiamy dodatkowy proces wewnątrz już działającego kontenera, bez wpływu na jego główny proces — najczęstsze zastosowanie to wejście do powłoki działającego kontenera w celu diagnostyki, bez jego restartowania.

Przykład:

```
root@a1b2c3d4e5f6:/#
```

docker = narzędzie do kontenerów

TYLKO ODCZYT

Podgląd logów kontenera: docker logs

```
docker logs -f NazwaKontenera
```

Wypisujemy standardowe wyjście i wyjście błędów procesu kontenera, z flagą `-f` śledząc je na bieżąco — to samo wyjście, które widzielibyśmy w terminalu, gdyby kontener nie był uruchomiony w tle.

Przykład:

```
2026-09-10T08:12:03.441Z [notice] nginx worker process started
2026-09-10T08:12:04.102Z [notice] listening on 0.0.0.0:80
```

docker = narzędzie do kontenerów

TYLKO ODCZYT

Sprawdzenie pełnej konfiguracji kontenera: docker inspect

```
docker inspect NazwaKontenera
```

Zwracamy pełną strukturę JSON opisującą kontener — zmienne środowiskowe, zamontowane wolumeny, adres IP, politykę restartu, wszystko, co zostało ustawione przy tworzeniu. Przydatne, gdy trzeba sprawdzić dokładną konfigurację kontenera uruchomionego dawno

temu przez kogoś innego.

Przykład:

```
[
    {
        "Id": "a1b2c3d4e5f6...",
        "State": { "Status": "running", "Running": true },
        "NetworkSettings": { "IPAddress": "172.17.0.2" }
    }
]
```

JSON = JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami · **IP** = Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection) · **docker** = narzędzie do kontenerów

TYLKO ODCZYT

Podgląd zużycia zasobów na żywo: docker stats

```
docker stats
```

Pokazujemy zużycie CPU, pamięci i ruchu sieciowego wszystkich działających kontenerów w czasie rzeczywistym, odświeżane co sekundę — pierwsze narzędzie do sprawdzenia, który kontener zjada zasoby hosta.

Przykład:

```
CONTAINER ID   NAME       CPU %     MEM USAGE / LIMIT     MEM %     NET I/O
a1b2c3d4e5f6   app_web    0.12%     24.3MiB / 512MiB      4.75%     1.2kB / 850B
f6e5d4c3b2a1   app_db     3.41%     186MiB / 1GiB         18.16%    412kB / 1.1MB
```

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

ZMIENIA SYSTEM

Skopiowanie pliku między kontenerem a hostem: docker cp

```
docker cp NazwaKontenera:/SciezkaDoPliku /SciezkaDoPliku
```

Kopiujemy plik lub katalog z kontenera na hosta (albo odwrotnie, po zamianie kolejności argumentów) bez potrzeby montowania wolumenu — wygodne przy jednorazowym wyciągnięciu pliku diagnostycznego z już działającego kontenera.

docker = narzędzie do kontenerów

ZMIENIA SYSTEM

Zatrzymanie kontenera: docker stop

```
docker stop NazwaKontenera
```

Wysyłamy do głównego procesu kontenera sygnał `SIGTERM` i czekamy (domyślnie 10 sekund) na jego samodzielne zakończenie, zanim Docker doda `SIGKILL` — dajemy procesowi szansę na uporządkowane zamknięcie (np. dokończenie zapisu do bazy).

SIGTERM = sygnał numer 15: prośba do procesu, żeby zakończył się grzecznie, proces może ją zignorować albo posprzątać przed wyjściem · **SIGKILL** = sygnał numer 9: każe jądru natychmiast zabić proces, bez szansy na sprzątnięcie po sobie · **docker** = narzędzie do kontenerów

ZMIENIA SYSTEM

Wymuszone zabicie kontenera: docker kill

```
docker kill NazwaKontenera
```

Wysyłamy `SIGKILL` natychmiast, bez czekania na uporządkowane zamknięcie procesu — stosujemy, gdy `docker stop` nie skutkuje albo gdy zależy nam na natychmiastowym efekcie kosztem ewentualnej utraty niezapisanych danych.

SIGKILL = sygnał numer 9: każe jądru natychmiast zabić proces, bez szansy na sprzątnięcie po sobie · **docker** = narzędzie do kontenerów

OSTROŻNIE · NIEODWRACALNE

Usunięcie kontenera: docker rm

```
docker rm NazwaKontenera
```

Usuwamy zatrzymany kontener razem z jego zapisywalną warstwą — wolumeny podpięte do niego przeżywają, chyba że dodamy flagę `-v`, która usunie też anonimowe wolumeny utworzone razem z tym kontenerem.

docker = narzędzie do kontenerów

TYLKO ODCZYT

Sprawdzenie zajętości miejsca przez Dockera: docker system df

```
docker system df
```

Zwracamy podsumowanie miejsca zajętego przez obrazy, kontenery, lokalne wolumeny i cache budowania — pierwszy krok przed decyzją o sprzątaniu, żeby wiedzieć, co faktycznie zajmuje najwięcej.

Przykład:

```
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          14        6         3.8GB     2.1GB (55%)
Containers      9         3         142MB     98MB (69%)
Local Volumes   5         3         1.2GB     310MB (25%)
Build Cache     42        0         890MB     890MB (100%)
```

docker = narzędzie do kontenerów

OSTROŻNIE · NIEODWRACALNE

Usunięcie nieużywanych zasobów: docker system prune

```
docker system prune
```

Kasujemy zatrzymane kontenery, nieużywane sieci, nieoznaczone obrazy (`<none>`) i cache budowania — po potwierdzeniu pytaniem, bo operacja jest nieodwracalna. Warianty tej komendy i ryzyko flagi `-a --volumes` rozwija strona Utrzymanie hosta dockerowego.

docker = narzędzie do kontenerów

Operacje na rejestrze obrazów

Zalogowanie do prywatnego rejestru: docker login

```
docker login git.zielona.example
```

Logujemy demona Dockera do wskazanego rejestru obrazów — bez tego `docker pull` i `docker push` na obrazy z rejestrów prywatnych kończą się błędem `unauthorized`. Komenda pyta interaktywnie o login i hasło, jeśli nie zostały podane inaczej. W Zielonej rejestrem obrazów jest rejestr pakietów Gitei pod `git.zielona.example`, a obrazy nazywamy `git.zielona.example/zielona/NazwaObrazu:wersja`; w polu hasła wpisujemy token z Gitei z zakresem `read:package` albo `write:package`.

Przykład:

```
Username: nazwa_usera
Password:
Login Succeeded
```

Poświadczenia zapisują się w pliku `~/.docker/config.json` danego użytkownika systemowego — przy przenoszeniu tej samej tożsamości logowania na inny serwer wystarczy skopiować ten plik do analogicznej lokalizacji zamiast logować się tam ręcznie po raz drugi, o ile hasło dla tego konta nie wymaga rotacji między serwerami.

Z życia wzięte

> **□□ Z ŻYCIA WZIĘTE**
>
> Ola dostaje `permission denied` przy pierwszym `docker pull` z rejestru firmowego na nowym laptopie, bo nikt jej nie powiedział, że trzeba się najpierw zalogować przez `docker login git.zielona.example`. Po zalogowaniu obraz pobiera się bez problemu, a poświadczenia zostają zapisane w `~/.docker/config.json` do następnego razu. Tomek kopiuje później ten sam plik na inny serwer, myśląc że to wystarczy, i zapomina, że tamto konto wymaga osobnej rotacji hasła.

Częste błędy

Wpisywanie komend Dockera bez `sudo`, gdy user nie jest w grupie `docker`. Kończy się komunikatem `permission denied while trying to connect to the Docker daemon socket` — trwałym

rozwiązaniem jest dodanie użytkownika do grupy `docker` (`usermod -aG docker nazwa_usera`) i ponowne zalogowanie, nie wpisywanie `sudo` przed każdą komendą.

Mylenie `docker stop` z `docker rm`. Zatrzymany kontener nadal istnieje i zajmuje miejsce — nowa osoba bywa zaskoczona, widząc go dalej w `docker ps -a` po tym, co uważała za jego usunięcie.

Uruchamianie `docker system prune` bez wcześniejszego `docker system df`. Polecenie kasuje bez pytania o każdy element z osobna — sprawdzenie, co faktycznie zajmuje miejsce, zanim skasujemy, chroni przed usunięciem czegoś, co ktoś inny właśnie przygotowywał do użycia.

□□ Ćwiczenia

> Zadania sprawdzające.

1. Piszemy jedną komendę `docker run`, która uruchamia kontener `nginx:latest` w tle, pod nazwą `strona_glowna`, z portem 80 kontenera zmapowanym na port 8081 hosta i polityką restartu `unless-stopped`.
2. Kontener `app_db` przestał odpowiadać. Opisujemy kolejność komend, którymi sprawdzimy jego logi, zużycie zasobów i pełną konfigurację, zanim zdecydujemy się go zrestartować.
3. Wyjaśniamy różnicę między `docker stop` a `docker kill` i podajemy sytuację, w której świadomie wybieramy to drugie.
4. `docker system df` pokazuje 2.1 GB miejsca możliwego do odzyskania w obrazach. Opisujemy, jaką komendą sprawdzimy to dokładniej przed wykonaniem `prune`.

Powiązane: Obraz, kontener, warstwa, wolumen: cztery pojęcia · Docker Compose · Utrzymanie hosta dockerowego

---

Revision #37
Created 2026-09-25 07:39:58 UTC by Tomasz Konieczny
Updated 2026-10-26 14:09:23 UTC by Tomasz Konieczny