Instalacja Dockera na Rocky Linux 9 z osobnym wolumenem /docker

Typ: procedura **Dotyczy: Rocky Linux 9, Docker CE 27+, wtyczka Compose**

Czas: ~45 min

Efekt: działający Docker Engine na Rocky Linux 9, z własnymi danymi na osobnym wolumenie `vgDocker/lvDocker` zamiast na dysku systemowym, gotowy na pierwszą aplikację opisaną na stronie Pierwsza aplikacja w Compose krok po kroku.

Skróty użyte na tej stronie (5)

Skrót	Co znaczy
CE	Community Edition: darmowa, społecznościowa wersja Dockera, w odróżnieniu od płatnej wersji Enterprise
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
UUID	Universally Unique Identifier: długi, praktycznie niepowtarzalny identyfikator
XFS	system plików stworzony przez Silicon Graphics, domyślny w Rocky Linux i RHEL

Zanim zaczniemy

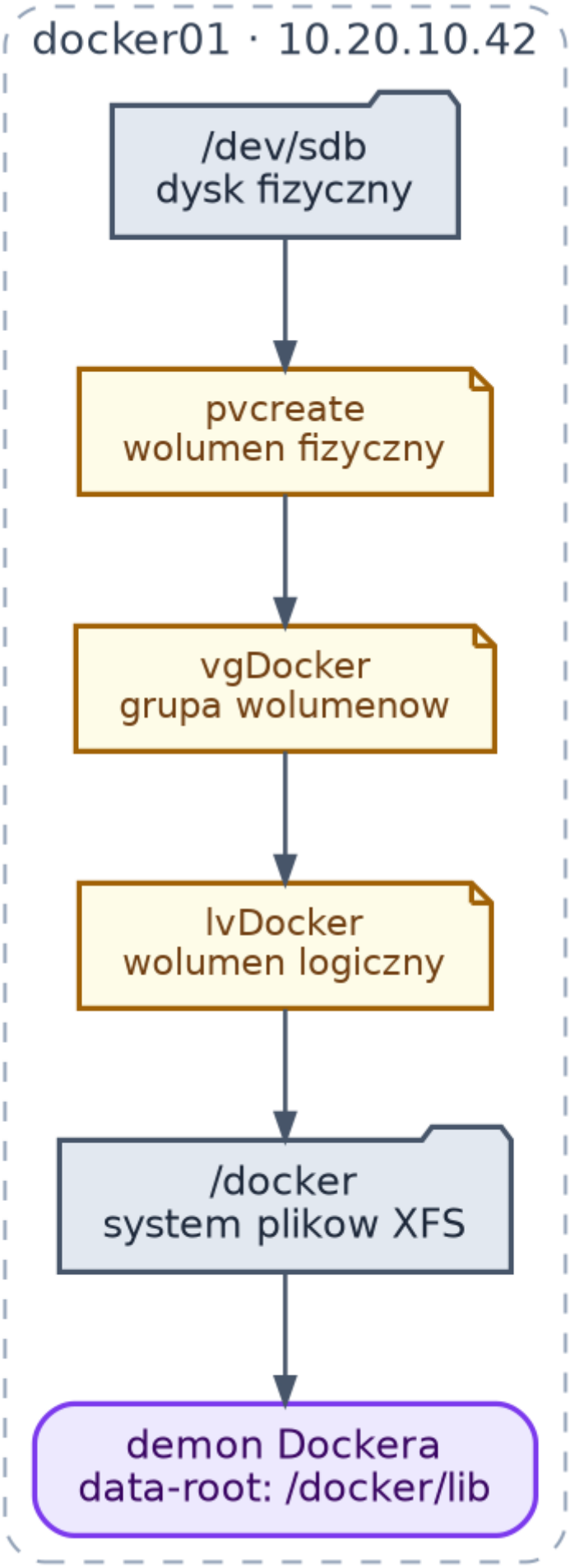
> W naszym środowisku dane aplikacji trzymamy w `/docker/apps/<nazwa>`, a plik Compose w `/docker/compose/<nazwa>/compose.yaml`. To konwencja, nie wymóg Dockera: można ją zmienić, byle w całej firmie była jedna. Katalog `/docker` to osobny wolumen `vgDocker/lvDocker`, żeby pełny dysk aplikacji nie zatrzymał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

Procedura zakłada gotowy system Rocky Linux 9 z jednym wolnym dyskiem pod przyszły `/docker`, zainstalowany według wzorca z Instalacja Rocky Linux z własnym nazewnictwem grup wolumenów:

system operacyjny na `vgOS/lvOS`, `/var` na `vgVAR/lvVAR`. Komendy wykonujemy z CLI bash z uprawnieniami root (sudo).

Wymaganie	Uwaga
Wolny dysk pod `vgDocker/lvDocker`	dołożenie dysku po instalacji systemu, patrz krok 1
Rocky Linux 9 z dostępem do internetu albo lokalnego lustra pakietów	repozytorium Docker CE pobiera pakiety przez sieć
Konto z prawem sudo	do instalacji pakietów i edycji `/etc/fstab`

Droga od gołego dysku do działającego demona Dockera, krok po kroku:


docker01 · 10.20.10.42
/dev/sdb
dysk fizyczny
pvcreate
wolumen fizyczny
vgDocker
grupa wolumenow
lvDocker
wolumen logiczny
/docker
system plikow XFS
demon Dockera
data-root: /docker/lib

Dysk fizyczny zamieniony przez pvcreate, vgcreate i lvcreate w wolumen logiczny, zamontowany jako /docker i wskazany demonowi Dockera jako data-root (kliknięcie otwiera pełny rozmiar)

Przebieg

1. Weryfikacja wolnego dysku

```
lsblk
```

Przykład:

```
NAME         MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda            8:0    0    60G  0 disk
├─sda1         8:1    0     1G  0 part /boot
└─sda2         8:2    0    59G  0 part
  └─vgOS-lvOS 253:0    0    50G  0 lvm  /
sdb            8:16   0   500G  0 disk
```

Dysk `/dev/sdb` na przykładzie wyżej to wolny dysk pod `vgDocker`, bez żadnej partycji ani systemu plików.

2. Utworzenie wolumenu fizycznego i grupy wolumenów

```
sudo pvcreate /dev/sdb
```

```
sudo vgcreate vgDocker /dev/sdb
```

Zostawiamy zapas w grupie zamiast od razu zająć całą przestrzeń wolumenem logicznym:

```
sudo lvcreate -L 400G -n lvDocker vgDocker
```

> Nie wykorzystujemy `100%FREE` przy zakładaniu wolumenu, z tego samego powodu co przy każdym innym wolumenie w Zielonej: wolne miejsce w grupie pozwala później powiększyć `lvDocker` jedną komendą, bez dokładania kolejnego

> dysku.

3. Utworzenie systemu plików XFS

```
sudo mkfs.xfs /dev/mapper/vgDocker-lvDocker
```

Przykład:

```
meta-data=/dev/mapper/vgDocker-lvDocker isize=512    agcount=4, agsize=26214400 blks
data     =                       bsize=4096   blocks=104857600, imaxpct=25
naming   =version 2              bsize=4096   ascii-ci=0, ftype=1
log      =internal log           bsize=4096   blocks=51200, version=2
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

XFS jest domyślnym systemem plików Rocky Linux 9 i tego samego używamy na pozostałych wolumenach opisanych w Instalacja Rocky Linux z własnym nazewnictwem grup wolumenów: jeden system plików w całej firmie, żeby nie pamiętać dwóch zestawów narzędzi diagnostycznych.

4. Wpis w /etc/fstab i montowanie

Sprawdzamy UUID nowego systemu plików:

```
sudo blkid /dev/mapper/vgDocker-lvDocker
```

Przykład:

```
/dev/mapper/vgDocker-lvDocker: UUID="3f7a9c1e-8b2d-4e5f-9a1c-6d8e2f4b7a9c" TYPE="xfs"
```

Dopisujemy wpis do `/etc/fstab`:

```
UUID=3f7a9c1e-8b2d-4e5f-9a1c-6d8e2f4b7a9c  /docker  xfs  defaults  0 2
```

> (gdzie `UUID` to wartość odczytana w poprzednim kroku, indywidualna dla każdego wolumenu)

Zakładamy punkt montowania i montujemy bez restartu:

```
sudo mkdir -p /docker
sudo mount -a
```

5. Weryfikacja montowania

```
df -h /docker
```

Przykład:

```
Filesystem                     Size  Used Avail Use% Mounted on
/dev/mapper/vgDocker-lvDocker  400G   30M  400G   1% /docker
```

6. Repozytorium Docker CE

```
sudo dnf config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo
```

> **Pojęcie: Docker CE.** Community Edition, darmowa wersja Docker Engine, w odróżnieniu od płatnej wersji Enterprise sprzedawanej z dodatkowym wsparciem i narzędziami do dużych wdrożeń. W Zielonej korzystamy wyłącznie z CE.

7. Instalacja Docker Engine z wtyczkami

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

8. Konfiguracja demona: data-root i rotacja logów

Zanim pierwszy raz uruchomimy usługę, wskazujemy demonowi własną lokalizację danych i limit logów, żeby nie trzeba było tego robić później na hoście z uruchomionymi już kontenerami:

```
sudo mkdir -p /docker/lib
```

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

```
{
  "data-root": "/docker/lib",
  "storage-driver": "overlay2",
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
  }
}
```

> **Pojęcie: `/docker/lib`.** Katalog na własne dane demona Dockera: pobrane obrazy, warstwy, sieci, anonimowe wolumeny. To trzecie miejsce pod `/docker`, obok `/docker/apps` (dane aplikacji) i `/docker/compose` (pliki Compose), bo trzyma wszystko, czym zarządza sam Docker, a nie żadna konkretna aplikacja. Leży na tym samym wolumenie `vgDocker/lvDocker` co reszta, bo obrazy i warstwy potrafią zająć równie dużo miejsca co dane aplikacji.

> Na hoście postawionym przed ujednoliceniem tej konwencji mogła się przyjąć inna nazwa tego katalogu, na przykład `/docker/docker-data`. Nowe instalacje zakładamy z `/docker/lib`, starych hostów nie migrujemy bez potrzeby, patrz Utrzymanie hosta dockerowego.

Limit `max-size` i `max-file` w `log-opts` obowiązuje globalnie dla nowo tworzonych kontenerów: każdy log rotuje po 10 MB, trzymając maksymalnie trzy pliki historii, więc żaden pojedynczy kontener nie zapełni `/docker/lib` samymi logami.

9. Uruchomienie usługi

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

10. Weryfikacja nowej lokalizacji danych

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

Przykład:

```
/docker/lib
```

11. Dodanie użytkownika do grupy docker

```
sudo usermod -aG docker nazwa_usera
```

> Konto dodane do grupy `docker` ma w praktyce pełne uprawnienia roota na tym hoście, mimo że formalnie rootem nie jest: demon Dockera działa jako root, a każdy, kto może z nim rozmawiać, może zamontować dowolny katalog hosta do kontenera i odczytać go bez ograniczeń. Dodajemy do tej grupy wyłącznie konta, które faktycznie mają administrować kontenerami, nigdy odruchowo, „żeby nie pisać sudo".

Zmiana grupy wymaga ponownego zalogowania konta, żeby nowa przynależność została uwzględniona w bieżącej sesji.

12. Firewalld i Docker

```
sudo firewall-cmd --list-all
```

Docker zarządza własnymi regułami `iptables` niezależnie od `firewalld` i domyślnie omija strefy firewalld przy publikowaniu portów kontenerów: port opublikowany przez `-p` albo przez `ports:` w compose jest osiągalny z zewnątrz, nawet jeśli `firewalld` w ogóle go nie wymienia jako otwartego. Świadome ograniczenie ruchu do konkretnych portów i adresów robimy przez sam Docker, adresem `127.0.0.1` przy publikowaniu portu, zgodnie ze stroną Jak trzymamy aplikacje w Dockerze: konwencje Zielonej, nie przez dopisywanie reguł do `firewalld`, które i tak nie obejmą ruchu przekierowanego przez Docker do kontenera.

13. Test poprawności instalacji

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

Przykład:

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

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

Weryfikacja

```
docker compose version
```

Przykład:

```
Docker Compose version v2.29.1
```

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

Przykład:

```
/docker/lib
```

Wycofanie zmian

Odinstalowanie Dockera bez usuwania danych z `/docker`:

```
sudo systemctl disable --now docker
```

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

Wolumen `vgDocker/lvDocker` i wpis w `/etc/fstab` zostają nietknięte: ponowna instalacja pakietu i wskazanie tego samego `data-root` w `daemon.json` odzyskuje wszystkie obrazy i kontenery bez ponownego pobierania.

Gdy coś nie zadziała

Objaw	Przyczyna	Co robimy
`docker info` pokazuje `/var/lib/docker` mimo zmiany w `daemon.json`	plik `daemon.json` ma błąd składni JSON i demon zignorował całą konfigurację	`python3 -m json.tool /etc/docker/daemon.json`, poprawiamy i restartujemy usługę
Kontener publikuje port mimo zamkniętej strefy w firewalld	Docker dopisuje własne reguły do `iptables` z pominięciem firewalld	ograniczamy publikację portu do `127.0.0.1`, nie liczymy na firewalld
`mount -a` zwraca błąd po dopisaniu wpisu do fstab	literówka w UUID albo w nazwie systemu plików	`blkid` ponownie, porównujemy z wpisem w `/etc/fstab` znak po znaku

Objaw	Przyczyna	Co robimy
Po `usermod -aG docker` polecenie `docker ps` nadal zwraca `permission denied`	użytkownik nie wylogował się i nie zalogował ponownie	nowa sesja logowania, ewentualnie `newgrp docker` na czas bieżącej sesji

Powiązane: Jak trzymamy aplikacje w Dockerze: konwencje Zielonej · Instalacja Rocky Linux z własnym nazewnictwem grup wolumenów · Utrzymanie hosta dockerowego · Pierwsza aplikacja w Compose krok po kroku · Połączenie z serwerem z Windowsa: SSH, Windows Terminal i PuTTY

Źródła: docs.docker.com, rozdział „Install Docker Engine on RHEL" · docs.docker.com, rozdział „Configure the Docker daemon" · man 5 fstab

---

Revision #31
Created 2026-09-27 11:19:57 UTC by Tomasz Konieczny
Updated 2026-10-26 14:09:19 UTC by Tomasz Konieczny