Pierwsza aplikacja w Compose krok po kroku

Typ: procedura **Dotyczy: Docker Compose v2, obraz nginx:1.27-alpine**

Czas: ~25 min

Efekt: własna, prosta strona WWW uruchomiona w kontenerze przez Docker Compose, od założenia katalogów po zatrzymanie i usunięcie, z każdym krokiem sprawdzonym wynikiem, zanim przejdziemy do prawdziwych aplikacji dalej w tej książce.

Skróty użyte na tej stronie (3)

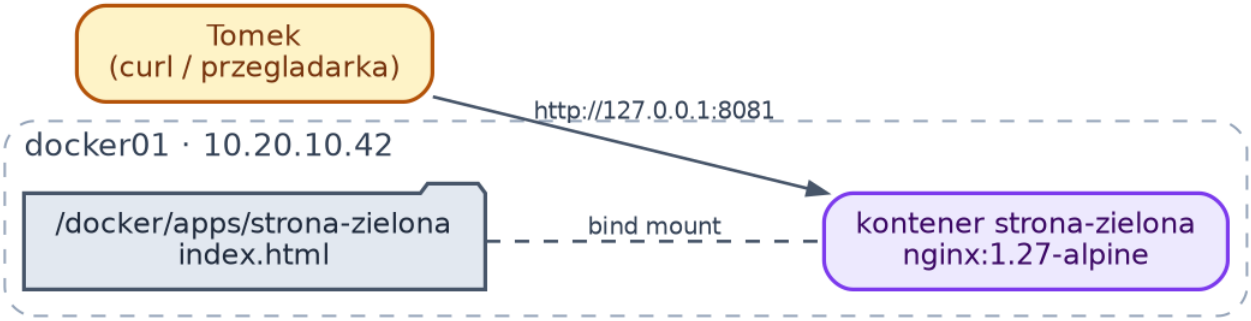
Skrót	Co znaczy
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
HTML	HyperText Markup Language: format tekstowy stron WWW, tu używany jako format raportu GPO do przeglądania w przeglądarce
WWW	World Wide Web: strony i aplikacje otwierane w przeglądarce po HTTP albo HTTPS

Zanim zaczniemy

Procedura zakłada działający Docker Engine z wtyczką Compose, zainstalowany według strony Instalacja Dockera na Rocky Linux 9 z osobnym wolumenem /docker, i katalogi zakładane zgodnie z konwencją opisaną na stronie Jak trzymamy aplikacje w Dockerze: konwencje Zielonej. Komendy wykonujemy z CLI bash jako user z uprawnieniami sudo albo jako user należący do grupy `docker`.

> 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.

Aplikacją testową jest zwykły serwer WWW Nginx, serwujący jedną stronę HTML z krótką informacją o Zielonej: wystarczająco prosty, żeby skupić się na samym Compose, a nie na konfiguracji aplikacji.



Katalog na hoście podpięty bind mountem do kontenera nginx, odpowiadający na zapytanie curl przez port 127.0.0.1 (kliknięcie otwiera pełny rozmiar)

Przebieg

1. Założenie katalogów

```
sudo mkdir -p /docker/apps/strona-zielona /docker/compose/strona-zielona
```

Przykład:

```
docker01:~$ ls -ld /docker/apps/strona-zielona /docker/compose/strona-zielona
drwxr-xr-x. 2 root root 6 wrz 27 11:00 /docker/apps/strona-zielona
drwxr-xr-x. 2 root root 6 wrz 27 11:00 /docker/compose/strona-zielona
```

2. Plik strony HTML

> **Pojęcie: bind mount.** Bezpośrednie podpięcie katalogu z hosta do wnętrza kontenera: to, co leży w `/docker/apps/strona-zielona` na hoście, kontener widzi pod wskazaną ścieżką u siebie, i odwrotnie. Zmiana pliku na hoście jest widoczna w kontenerze natychmiast, bez ponownego budowania czegokolwiek.

```
sudo tee /docker/apps/strona-zielona/index.html > /dev/null << 'EOF'
<!DOCTYPE html>
```

```
<html lang="pl">
<head><meta charset="utf-8"><title>Zielona sp. z o.o.</title></head>
<body>
<h1>Zielona sp. z o.o.</h1>
<p>Hurtownia artykulow spozywczych, centrala w Pamiatkowie, magazyn w Swarzedzu.</p>
</body>
</html>
EOF
```

3. Plik compose.yaml

```
sudo vim /docker/compose/strona-zielona/compose.yaml
```

```
services:
  server:
    image: nginx:1.27-alpine
    container_name: strona-zielona
    restart: unless-stopped
    volumes:
      - /docker/apps/strona-zielona/index.html:/usr/share/nginx/html/index.html:ro
    ports:
      - "127.0.0.1:8081:80"
```

> **Pojęcie: obraz.** Gotowy szablon systemu plików aplikacji, tutaj `nginx:1.27-alpine`: pobrany raz z internetowego rejestru, leży potem lokalnie na hoście i służy do uruchamiania dowolnej liczby kontenerów bez ponownego pobierania. Numer `1.27-alpine` to przypięta wersja: dokładnie ten sam obraz przy każdym kolejnym `pull`, zgodnie z konwencją z poprzedniej strony.

Port publikujemy na `127.0.0.1`, nie na wszystkich interfejsach, z tego samego powodu co każda inna aplikacja w Zielonej: docelowo stanie przed nią Nginx frontowy z certyfikatem, a na czas tego ćwiczenia sprawdzamy stronę z samego hosta.

4. Pierwsze uruchomienie

```
cd /docker/compose/strona-zielona
docker compose up -d
```

Przykład:

```
[+] Running 2/2
 ✔ Network strona-zielona_default   Created
 ✔ Container strona-zielona         Started
```

5. Sprawdzenie stanu kontenera

```
docker compose ps
```

Przykład:

```
NAME             IMAGE               STATUS
strona-zielona    nginx:1.27-alpine   Up 8 seconds
```

6. Sprawdzenie odpowiedzi serwera

```
curl -s http://127.0.0.1:8081 | head -n 5
```

Przykład:

```
<!DOCTYPE html>
<html lang="pl">
<head><meta charset="utf-8"><title>Zielona sp. z o.o.</title></head>
<body>
<h1>Zielona sp. z o.o.</h1>
```

7. Podgląd logów

```
docker compose logs -f
```

Przykład:

```
strona-zielona  | 10.20.10.42 - - [27/Sep/2026:11:05:02 +0000] "GET / HTTP/1.1" 200 612 "-"
"curl/8.5.0"
```

Przerywamy podgląd kombinacją `Ctrl+C`, kontener działa dalej w tle.

8. Zmiana strony i ponowne zastosowanie

Zmieniamy treść pliku HTML na hoście:

```
sudo sed -i 's/Pamiatkowie/Pamiatkowie (od 1998 roku)/' /docker/apps/strona-zielona/index.html
```

Odświeżenie strony pokazuje zmianę natychmiast, bo plik jest bind mountem, nie kopią zamkniętą wewnątrz obrazu. `docker compose up -d` po samej zmianie pliku HTML nic nie zmienia w stanie kontenera, bo nie zmieniliśmy `compose.yaml`:

```
docker compose up -d
```

Przykład:

```
[+] Running 1/1
 ✔ Container strona-zielona  Running
```

Compose porównuje aktualną konfigurację z tą, z jaką kontener już działa, i nie robi nic, gdy nie ma różnicy: `up -d` jest bezpieczne do wywołania wielokrotnie.

9. Zmiana w compose.yaml i ponowne zastosowanie

Zmieniamy port hosta z `8081` na `8082` w `compose.yaml`, zapisujemy plik i uruchamiamy ponownie:

```
docker compose up -d
```

Przykład:

```
[+] Running 1/1
 ✔ Container strona-zielona  Recreated
```

Tym razem Compose wykrywa różnicę w `ports` i podmienia kontener na nowy, z nową konfiguracją: `Recreated` zamiast `Running` w wyniku.

10. Zatrzymanie i usunięcie

```
docker compose down
```

Przykład:

```
[+] Running 2/2
 ✔ Container strona-zielona       Removed
 ✔ Network strona-zielona_default Removed
```

Plik HTML w `/docker/apps/strona-zielona` przeżywa `down`, bo leży na hoście, nie w kontenerze: usuwamy go ręcznie, tylko jeśli faktycznie kończymy z tym ćwiczeniem.

```
sudo rm -rf /docker/apps/strona-zielona /docker/compose/strona-zielona
```

Weryfikacja

```
docker compose ps
```

Przykład:

```
NAME   IMAGE   STATUS
```

Pusty wynik, bez żadnego wiersza, potwierdza, że żadna usługa z tego katalogu już nie działa.

Wycofanie zmian

Zatrzymanie bez usuwania danych, jeśli chcemy tylko chwilowo wyłączyć stronę:

```
docker compose stop
```

Ponowne uruchomienie tego samego kontenera, bez odtwarzania niczego od zera:

```
docker compose start
```

Gdy coś nie zadziała

Objaw	Przyczyna	Co robimy

`curl` zwraca `Connection refused` na `127.0.0.1:8081`	kontener jeszcze się uruchamia albo port zajęty przez inną usługę	`docker compose logs`, `ss -tlnp \`
Zmiana w `index.html` niewidoczna po odświeżeniu w przeglądarce	przeglądarka pokazuje stronę z pamięci podręcznej	twarde odświeżenie kombinacją `Ctrl+Shift+R` albo sprawdzenie przez `curl` z terminala
`docker compose up -d` po zmianie portu nie tworzy kontenera na nowo	literówka w pliku, Compose wczytał starą, niezmienioną wersję	`docker compose config`, sprawdzamy, co Compose faktycznie odczytał
`Permission denied` przy zapisie do `index.html` przez `sed`	plik należy do innego użytkownika niż ten, który wykonuje komendę	`sudo` przed komendą edytującą plik na hoście

Powiązane: Jak trzymamy aplikacje w Dockerze: konwencje Zielonej · Docker Compose · Codzienne komendy

Źródła: docs.docker.com, rozdział „Compose file reference" · hub.docker.com, obraz `nginx`

Revision #31
Created 2026-09-27 11:19:58 UTC by Tomasz Konieczny
Updated 2026-10-26 14:09:20 UTC by Tomasz Konieczny