

Docker

- Konfiguracja silnika
 - Instalacja Docker ze zmianą domyślnej lokalizacji
- Docker Compose
 - Możliwości docker-compose.yml

Konfiguracja silnika

Instalacja Docker ze zmianą domyślnej lokalizacji

Instalacja repozytorium

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

Instalacja Docker Engine:

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

Zmiana domyślnej lokalizacji plików dockera:

Utworzenie pliku `/etc/docker/daemon.json`, w którym wskażemy inną ścieżkę "data-root"

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

W pliku wklejamy odpowiednią konfigurację

```
{  
  "data-root": "/docker/docker-data"  
}
```

Następnie startujemy usługę dockera:

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

Sprawdzamy, czy docker używa nowej lokalizacji:

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

Uruchomienie testowego kontenera w celu weryfikacji poprawności instalacji:

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

Output powinien wyglądać tak:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
719385e32844: Pull complete
Digest: sha256:926fac19d22aa2d60f1a276b66a20eb765fbbea2db5dbdaafeb456ad8ce81598
Status: Downloaded newer image for hello-world:latest
```

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

<https://hub.docker.com/>

For more examples and ideas, visit:

<https://docs.docker.com/get-started/>

Docker Compose

Możliwości docker-compose.yml

Docker Compose (Docker Compose) umożliwia uruchamianie wielu kontenerów opartych na gotowych obrazach (np. `postgres`, `redis`, `nginx`) przy użyciu jednego pliku `docker-compose.yml`. Konfiguracja polega na zadeklarowaniu usług (`services`), sieci (`networks`) oraz wolumenów (`volumes`). Poniżej znajduje się praktyczna instrukcja skupiona na pracy z gotowymi obrazami, komunikacją między kontenerami oraz szczegółowym wykorzystaniem wolumenów.

Podstawowa konfiguracja usług opartych na gotowych obrazach wygląda następująco:

```
version: "3.9"

services:
  app:
    image: nginx:latest
    ports:
      - "8080:80"

  db:
    image: postgres:15
    environment:
      POSTGRES_PASSWORD: example
```

Po uruchomieniu polecenia `docker compose up -d` Compose pobiera obrazy, tworzy kontenery i domyślną sieć. Każda usługa jest dostępna dla innych usług pod nazwą swojej sekcji, czyli `db` jest osiągalna jako host `db`.

Sieci kontrolują komunikację między usługami. Jeśli nie zostaną jawnie zdefiniowane, Compose tworzy jedną sieć typu bridge i podłącza do niej wszystkie kontenery. W bardziej złożonych przypadkach należy definiować własne sieci, aby ograniczyć dostęp między komponentami:

```
services:
  frontend:
```

```
image: nginx
```

```
networks:
```

```
- front
```

```
backend:
```

```
image: nginx
```

```
networks:
```

```
- front
```

```
- back
```

```
db:
```

```
image: postgres
```

```
networks:
```

```
- back
```

```
networks:
```

```
front:
```

```
back:
```

W tym układzie `frontend` nie ma dostępu do `db`, ponieważ nie współdzieli sieci. `backend` pełni rolę pośrednika. Jedna usługa może należeć do wielu sieci. Sieć może być oznaczona jako `internal: true`, co blokuje ruch wychodzący poza sieć Dockera:

```
networks:
```

```
back:
```

```
internal: true
```

Wolumeny odpowiadają za trwałość danych i stanowią jeden z najważniejszych elementów konfiguracji. Bez wolumenów dane zapisane w kontenerze zostaną utracone po jego usunięciu. Docker Compose obsługuje kilka sposobów montowania danych.

Najprostszy wariant to bind mount, czyli mapowanie katalogu z hosta:

```
services:
```

```
db:
```

```
image: postgres
```

```
volumes:
```

```
- ./data:/var/lib/postgresql/data
```

Dane są zapisywane bezpośrednio w katalogu `./data`. Rozwiązanie to jest użyteczne w środowiskach developerskich, gdzie wymagany jest bezpośredni dostęp do plików.

Drugim podejściem są named volumes zarządzane przez Dockera:

```
services:
  db:
    image: postgres
    volumes:
      - db_data:/var/lib/postgresql/data

volumes:
  db_data:
```

W tym przypadku Docker przechowuje dane w swojej przestrzeni (`/var/lib/docker/volumes/...`). Named volumes są niezależne od struktury katalogów hosta i bardziej przenośne.

Możliwe jest także użycie wolumenów anonimowych:

```
services:
  app:
    image: nginx
    volumes:
      - /data
```

Docker tworzy wtedy wolumen bez nazwy i przypisuje go do kontenera. Rozwiązanie to jest rzadziej stosowane, ponieważ utrudnia zarządzanie.

Wolumeny mogą być współdzielone między usługami:

```
services:
  db:
    image: postgres
    volumes:
      - db_data:/data

  backup:
    image: alpine
    volumes:
      - db_data:/data
```

Pozwala to na wykonywanie operacji pomocniczych, takich jak backup lub migracje danych.

Dostęp do wolumenów można ograniczyć do trybu tylko do odczytu:

```
services:
  app:
    image: nginx
    volumes:
      - db_data:/data:ro
```

W tym przypadku kontener nie może modyfikować danych.

Możliwe jest także bardziej szczegółowe definiowanie wolumenów przy użyciu składni rozszerzonej:

```
services:
  app:
    image: nginx
    volumes:
      - type: volume
        source: db_data
        target: /data
        read_only: true

volumes:
  db_data:
```

Wolumeny mogą korzystać z różnych sterowników (`driver`), co umożliwia integrację np. z systemami zewnętrznymi:

```
volumes:
  db_data:
    driver: local
```

Dodatkowe opcje sterownika można przekazać przez `driver_opts`, np. przy mountowaniu zasobów sieciowych.

Ważnym przypadkiem są wolumeny zewnętrzne (`external`), które nie są tworzone przez Compose, lecz muszą istnieć wcześniej:

```
volumes:
  db_data:
    external: true
```

Pozwala to współdzielić dane między różnymi projektami lub zachować je niezależnie od cyklu życia Compose.

Typowym wzorcem jest inicjalizacja danych przy pierwszym uruchomieniu kontenera (np. w obrazach takich jak `postgres`), gdzie katalog danych jest pusty — wolumen zapewnia, że inicjalizacja nie zostanie powtórzona przy kolejnym uruchomieniu.

Zachowanie wolumenów przy usuwaniu środowiska:

- `docker compose down` usuwa kontenery i sieci, ale pozostawia wolumeny
- `docker compose down -v` usuwa również wolumeny
- `docker volume ls` pozwala zobaczyć dostępne wolumeny
- `docker volume rm` usuwa wskazany wolumen

Pełny przykład konfiguracji z sieciami i wolumenami:

```
version: "3.9"

services:
  app:
    image: nginx
    ports:
      - "8080:80"
    networks:
      - frontend
      - backend

  db:
    image: postgres:15
    environment:
      POSTGRES_PASSWORD: example
    volumes:
      - db_data:/var/lib/postgresql/data
    networks:
      - backend

  cache:
    image: redis:7
    networks:
      - backend

networks:
```

frontend:

backend:

volumes:

db_data:

W tej konfiguracji `app` komunikuje się z `db` i `cache` przez sieć `backend`, natomiast `db` nie jest dostępna z zewnątrz. Dane bazy są przechowywane w wolumenie `db_data`, który pozostaje po usunięciu kontenerów.