PostgreSQL w Dockerze na własnym komputerze: piaskownica do nauki

Poziom: podstawy **Dotyczy: PostgreSQL 18 w Dockerze** **Czas: ~25 min**

Zanim zaczniemy ćwiczyć `SELECT`, `JOIN` i transakcje z kolejnych rozdziałów, potrzebujemy bazy, na której wolno nam popełnić każdy błąd. Ta strona stawia własny PostgreSQL w kontenerze, który da się zniszczyć i odtworzyć od zera w minutę, bez ryzyka dla czegokolwiek produkcyjnego.

Skróty użyte na tej stronie (2)

Skrót	Co znaczy
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
Wi-Fi	Sieć bezprzewodowa: wygodna w biurze, ale w serwerowni dokłada kolejną możliwą przyczynę awarii

Fiszka

TYLKO ODCZYT

Instalacja Docker Desktop na Windows

```
docker.com/products/docker-desktop → Download for Windows → uruchomienie instalatora → restart komputera
```

Docker Desktop instaluje silnik kontenerów razem z graficznym panelem do podglądu uruchomionych kontenerów. Po restarcie sprawdzamy, czy działa, poleceniem niżej w dowolnym terminalu (PowerShell wystarczy).

□□ TYLKO ODCZYT

Sprawdzenie instalacji Dockera

```
docker --version
```

Przykład:

```
Docker version 27.3.1, build ce12230
```

□□ **docker** = narzędzie do kontenerów

□□ ZMIENIA SYSTEM

Instalacja Dockera na Rocky Linux

```
sudo dnf install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin
sudo systemctl enable --now docker
```

Na maszynie linuksowej (na przykład własnym laptopie z Rocky albo maszynie wirtualnej) instalujemy silnik Dockera z oficjalnego repozytorium producenta, a nie wersję `podman-docker` z repozytoriów dystrybucji, bo różni się poleceniami pomocniczymi. Repozytorium Dockera dla Rocky Linux dodajemy wcześniej poleceniem `dnf config-manager --add-repo`, zgodnie z instrukcją na `docs.docker.com`.

Przykład:

```
Installed:
  docker-ce-3:27.3.1-1.el9.x86_64  docker-ce-cli-1:27.3.1-1.el9.x86_64  ...
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service →
/usr/lib/systemd/system/docker.service.
```

> W naszym środowisku firmowym 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. Na własnym komputerze do nauki ta konwencja nie obowiązuje: wystarczy jeden zwykły katalog, na przykład `piaskownica-postgresql`, obok innych naszych projektów.

□□ **dnf** = Dandified YUM: menedżer pakietów RHEL/Rocky · **systemctl** = system control: sterowanie usługami systemd

ZMIENIA SYSTEM

Utworzenie plików piaskownicy

```
mkdir piaskownica-postgresql && cd piaskownica-postgresql
```

Zakładamy jeden katalog na dwa pliki: `compose.yaml` z definicją kontenera i `.env` z hasłem, które nigdy nie trafia do żadnego repozytorium kodu.

mkdir = make directory: utwórz katalog

PLIK KONFIGURACYJNY

Plik compose.yaml

```
services:
  db:
    image: postgres:18.6
    restart: unless-stopped
    environment:
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: piaskownica
    ports:
      - "127.0.0.1:5432:5432"
    volumes:
      - dane_postgres:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 5s
      timeout: 5s
      retries: 5

volumes:
  dane_postgres:
```

Obraz `postgres:18.6` to oficjalny obraz PostgreSQL w wersji 18 (sprawdzone na `hub.docker.com/_/postgres` 28 września 2026), przypięty do konkretnej wersji zamiast `latest`, żeby kolejne `docker compose up` nie podmieniło nam wersji silnika bez ostrzeżenia. Port wystawiamy wyłącznie na `127.0.0.1`, nie na wszystkie interfejsy, żeby nikt inny w tej samej sieci Wi-Fi nie zapukał do naszej piaskownicy.

□□ **Wi-Fi** = Sieć bezprzewodowa: wygodna w biurze, ale w serwerowni dokłada kolejną możliwą przyczynę awarii

□□ TYLKO ODCZYT

Plik .env z hasłem

```
POSTGRES_PASSWORD=zmien-to-haslo-na-swoje
```

Hasło stoi osobno od `compose.yaml`, żeby dało się bezpiecznie wrzucić plik Compose do repozytorium bez wycieku hasła. Plik `.env` dopisujemy do `.gitignore`, jeśli katalog w ogóle trafia pod kontrolę wersji.

□□ ZMIENIA SYSTEM

Pierwsze uruchomienie kontenera

```
docker compose up -d
```

Flaga `-d` uruchamia kontener w tle, żeby terminal został wolny do dalszej pracy. Przy pierwszym starcie Docker pobiera obraz z internetu, co potrwa dłużej niż każde kolejne uruchomienie.

Przykład:

```
[+] Running 2/2
 ✔ Volume "piaskownica-postgresql_dane_postgres"  Created
 ✔ Container piaskownica-postgresql-db-1          Started
```

□□ **docker** = narzędzie do kontenerów

□□ TYLKO ODCZYT

Sprawdzenie stanu kontenera

```
docker compose ps
```

Kolumna `STATUS` pokazuje `healthy` dopiero, gdy healthcheck z `compose.yaml` potwierdzi, że PostgreSQL faktycznie przyjmuje połączenia, nie tylko że proces wystartował.

Przykład:

```
NAME                          IMAGE           STATUS
piaskownica-postgresql-db-1   postgres:18.6   Up 12 seconds (healthy)
```

docker = narzędzie do kontenerów

TYLKO ODCZYT

Połączenie z bazą przez docker compose exec

```
docker compose exec db psql -U postgres -d piaskownica
```

Wchodzimy do wnętrza kontenera i od razu uruchamiamy tam `psql`, klienta wiersza poleceń opisanego na stronie psql: klient wiersza poleceń, bez instalowania go osobno na naszym komputerze.

Przykład:

```
psql (18.6)
Type "help" for help.


piaskownica=#
```

docker = narzędzie do kontenerów

ZMIENIA SYSTEM

Zatrzymanie piaskownicy z zachowaniem danych

```
docker compose down
```

Usuwa kontener, ale zostawia wolumen `dane_postgres` nietknięty; kolejne `docker compose up -d` odtworzy kontener z tymi samymi danymi co przed zatrzymaniem.

docker = narzędzie do kontenerów

ZMIENIA SYSTEM

Zniszczenie piaskownicy razem z danymi

```
docker compose down -v
```

Flaga `-v` usuwa razem z kontenerem także wolumen, czyli wszystkie dane wewnątrz. To jest dokładnie ten reset do zera, po który sięgamy, kiedy chcemy zacząć ćwiczenie od czystej bazy, bez śladu poprzednich prób.

Przykład:

```
[+] Running 3/3
 ✔ Container piaskownica-postgresql-db-1          Removed
 ✔ Volume piaskownica-postgresql_dane_postgres    Removed
 ✔ Network piaskownica-postgresql_default         Removed
```

docker = narzędzie do kontenerów

Z życia wzięte

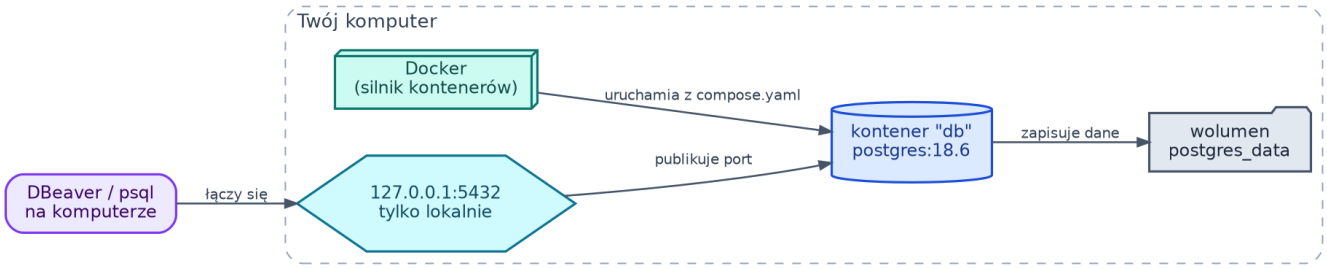
Z ŻYCIA WZIĘTE

Tomek, drugi tydzień w pracy, chce sprawdzić, co się stanie, jeśli wykona `DELETE FROM zamowienia;` bez `WHERE`. Zamiast ryzykować na `pg01`, stawia tę samą piaskownicę na swoim laptopie, wykonuje polecenie, patrzy na pustą tabelę, uczy się czegoś ważnego i jednym `docker compose down -v` sprząta po sobie, zanim ktokolwiek zdąży zapytać, co robił.

Jak to działa

Rozwiń: jak to działa pod spodem

Pojęcie: piaskownica. Środowisko odizolowane od tego, co produkcyjne, w którym błąd nie kosztuje nic poza czasem na jego naprawienie. Piaskownica w Dockerze różni się od bazy zainstalowanej wprost na naszym systemie tym, że mieszka w całości wewnątrz jednego kontenera i jednego wolumenu, więc jej usunięcie nie zostawia żadnych plików porozrzucanych po dysku.



Droga połączenia od klienta przez Dockera do kontenera PostgreSQL i wolumenu z danymi (kliknięcie otwiera pełny rozmiar)

Wolumen `dane_postgres` to osobny byt od samego kontenera: kontener można usunąć i stworzyć od nowa (`docker compose down` bez `-v`, potem `docker compose up -d`) i dane przetrwają, bo mieszkają w wolumenie, nie w kontenerze. Dopiero jawne usunięcie wolumenu (`-v` albo `docker volume rm`) kasuje dane naprawdę. To rozróżnienie, kontener kontra wolumen, jest tym samym rozróżnieniem, co instancja i baza danych ze strony Baza danych, silnik, instancja, schemat: proces można zrestartować, a dane, które ten proces obsługuje, żyją niezależnie od niego.

Załadowanie przykładowych tabel

Piaskownica jest pusta zaraz po starcie. Wklejamy poniższy blok do sesji `psql` otwartej poleceniem `docker compose exec db psql -U postgres -d piaskownica` z Fiszki wyżej, żeby mieć te same tabele, które wracają w kolejnych rozdziałach tej książki.

```
CREATE TABLE klienci (
    id SERIAL PRIMARY KEY,
    nazwa VARCHAR(200) NOT NULL,
    miasto VARCHAR(100),
    nip VARCHAR(10) UNIQUE
);

CREATE TABLE produkty (
    id SERIAL PRIMARY KEY,
    nazwa VARCHAR(200) NOT NULL,
    kategoria VARCHAR(50) NOT NULL,
    cena NUMERIC(10, 2) NOT NULL,
    dostepny BOOLEAN NOT NULL DEFAULT true
```

```
);

CREATE TABLE zamowienia (
    id SERIAL PRIMARY KEY,
    klient_id INTEGER NOT NULL REFERENCES klienci(id),
    data_zlozenia DATE NOT NULL DEFAULT CURRENT_DATE
);

CREATE TABLE pozycje (
    id SERIAL PRIMARY KEY,
    zamowienie_id INTEGER NOT NULL REFERENCES zamowienia(id),
    produkt_id INTEGER NOT NULL REFERENCES produkty(id),
    ilosc INTEGER NOT NULL CHECK (ilosc > 0)
);

CREATE TABLE pracownicy (
    id SERIAL PRIMARY KEY,
    imie VARCHAR(100) NOT NULL,
    nazwisko VARCHAR(100) NOT NULL,
    dzial VARCHAR(50) NOT NULL,
    przelozony_id INTEGER REFERENCES pracownicy(id)
);

INSERT INTO klienci (nazwa, miasto, nip) VALUES
    ('Nowak i Wspólnicy Sp. z o.o.', 'Warszawa', '5261234567'),
    ('Sklep Zielony Kącik', 'Poznań', '7791234567'),
    ('Hurtownia Pod Dębem', 'Swarzędz', '7771234567');

INSERT INTO produkty (nazwa, kategoria, cena, dostepny) VALUES
    ('Kawa mielona 500g', 'Spożywcze', 24.90, true),
    ('Herbata czarna 100 torebek', 'Spożywcze', 14.50, true),
    ('Woda mineralna 1,5l', 'Napoje', 2.30, true),
    ('Skrzynka na butelki', 'Opakowania', 18.00, false);

INSERT INTO zamowienia (klient_id, data_zlozenia) VALUES
    (1, '2026-09-14'),
    (1, '2026-09-21'),
    (2, '2026-09-20');
```

```
INSERT INTO pozycje (zamowienie_id, produkt_id, ilosc) VALUES
    (1, 1, 10), (1, 3, 24),
    (2, 2, 5),
    (3, 1, 4), (3, 4, 2);

INSERT INTO pracownicy (imie, nazwisko, dzial, przelozony_id) VALUES
    ('Ewa', 'Kowalczyk', 'Sprzedaż', NULL),
    ('Marek', 'Zawadzki', 'Sprzedaż', 1),
    ('Tomasz', 'Wróbel', 'Sprzedaż', 1),
    ('Anna', 'Sikora', 'Sprzedaż', 2),
    ('Paweł', 'Górski', 'Sprzedaż', 2),
    ('Katarzyna', 'Baran', 'Sprzedaż', 3);
```

Pięć tabel odtwarza dokładnie te definicje, które widzieliśmy przy okazji kluczy, relacji i typów danych we wcześniejszych rozdziałach: `klienci` i `produkty` jako tabele niezależne, `zamowienia` i `pozycje` połączone kluczami obcymi w relację 1:N, `pracownicy` z kolumną `przelozony_id` wskazującą samą siebie, do której wracamy przy `SELF JOIN` i zapytaniach rekurencyjnych.

Połączenie DBeaverem albo pgAdminem do piaskownicy

Piaskownica słucha na `127.0.0.1:5432`, więc w DBeaverze zakładamy nowe połączenie PostgreSQL dokładnie tak, jak na stronie Pierwsze połączenie z bazą myszką: DBeaver i SSMS, tylko z Host `localhost`, Database `piaskownica`, Username `postgres` i hasłem z pliku `.env`. Kto woli natywne narzędzie od samych twórców PostgreSQL, może zamiast DBeavera postawić kontener `dpage/pgadmin4` obok `db` w tym samym pliku `compose.yaml`.

> 🖼 Zrzut do uzupełnienia: DBeaver, nowe połączenie PostgreSQL z Host `localhost`, Port `5432`, Database `piaskownica`, po udanym „Test Connection".

Częste błędy

Objaw	Przyczyna	Co robimy

`docker compose up` kończy się błędem „port is already allocated"	Coś innego na komputerze już zajmuje port 5432, na przykład lokalnie zainstalowany PostgreSQL	Zmieniamy w `compose.yaml` mapowanie na inny port hosta, na przykład `127.0.0.1:5433:5432`, i łączymy się klientem na port 5433
Po `docker compose down -v` dane zniknęły, choć mieliśmy je jeszcze użyć	Flaga `-v` usuwa wolumen razem z kontenerem, nie tylko sam kontener	Do zwykłego zatrzymania używamy `docker compose down` bez `-v`; `-v` zostawiamy świadomie na moment, kiedy naprawdę chcemy zacząć od zera
DBeaver z zewnątrz komputera nie widzi piaskownicy	Port wystawiony tylko na `127.0.0.1`, zgodnie z zaleceniem tej strony, jest niewidoczny spoza tego samego komputera	To zamierzone dla piaskownicy do nauki; jeśli naprawdę potrzebujemy dostępu z innego urządzenia, świadomie zmieniamy mapowanie na `0.0.0.0:5432:5432`, mając na uwadze, że wtedy widzi to też reszta naszej sieci
`docker compose exec db psql` zwraca „the input device is not a TTY"	Polecenie uruchomione w środowisku bez terminala interaktywnego, na przykład w niektórych zdalnych sesjach CI	Dodajemy flagę `-T` do `docker compose exec -T db psql ...` w skryptach automatyzujących, gdzie nikt nie patrzy na ekran na żywo
Zmiana hasła w `.env` nie działa po ponownym `docker compose up -d`	`POSTGRES_PASSWORD` ustawia hasło tylko przy pierwszym starcie, gdy wolumen jest jeszcze pusty	Żeby zmienić hasło na istniejącej piaskownicy, łączymy się i wykonujemy `ALTER USER postgres WITH PASSWORD 'nowe';`, albo świadomie robimy `docker compose down -v` i zaczynamy od zera

□□ Ćwiczenia

1. Stawiamy piaskownicę od zera, ładujemy pięć tabel z tej strony i sprawdzamy w DBeaverze albo `psql`, ile wierszy ma tabela `pozycje`.

Odpowiedź

`SELECT count(*) FROM pozycje;` zwraca `5`, zgodnie z pięcioma wierszami wstawionymi w bloku `INSERT INTO pozycje` powyżej.

1. Zatrzymujemy piaskownicę poleceniem `docker compose down` (bez `-v`), uruchamiamy ją ponownie i sprawdzamy tabelę `klienci`. Czy dane wciąż tam są? Dlaczego to samo pytanie po `docker compose down -v` da inną odpowiedź?

Odpowiedź

Po zwykłym `docker compose down` dane zostają, bo żyją w wolumenie `dane_postgres`, niezależnym od usuniętego kontenera; `docker compose up -d` tworzy nowy kontener i podłącza go do tego samego, istniejącego już wolumenu. Po `docker compose down -v` wolumen zostaje usunięty razem z kontenerem, więc kolejny `docker compose up -d` startuje z zupełnie pustym wolumenem i tabela `klienci` w ogóle nie istnieje, dopóki nie wykonamy `CREATE TABLE` od nowa.

1. Dlaczego port piaskownicy mapujemy jako `127.0.0.1:5432:5432`, a nie po prostu `5432:5432` ?

Odpowiedź

Zapis `5432:5432` bez adresu wystawia port na wszystkich interfejsach sieciowych komputera, więc każdy w tej samej sieci lokalnej (biurowe Wi-Fi, hotel, sieć domowa) może spróbować się połączyć, jeśli zna adres IP naszego komputera. Dopisanie `127.0.0.1` ogranicza dostęp wyłącznie do procesów uruchomionych na tym samym komputerze, co dla piaskownicy do nauki jest jedynym rozsądnym ustawieniem.

Powiązane: Baza danych, silnik, instancja, schemat · psql: klient wiersza poleceń · Pierwsze połączenie z bazą myszką: DBeaver i SSMS · Klucze: główny, obcy, unikalny, naturalny, sztuczny

Źródła: hub.docker.com/_/postgres (sprawdzone 28 września 2026) · docs.docker.com/engine/install/rockylinux · docs.docker.com/compose/compose-file · postgresql.org/docs/current/app-psql.html

Revision #21
Created 2026-09-28 17:51:23 UTC by Tomasz Konieczny
Updated 2026-10-26 14:10:42 UTC by Tomasz Konieczny