Неделя 5. Лаба 3. Данные и Compose: PostgreSQL для сервиса задач

Часть 1 из 2: разделы «Часть 1. БД в контейнере» — «Часть 2. Сервис + БД».

Цель: подключить сервису настоящую БД — PostgreSQL в контейнере с томом и учебными данными, — запустить сервис-контейнер рядом с БД-контейнером и описать всё окружение в `compose.yaml`. **Время:** 150 мин (удобно разбить на два занятия: части 1–2 и часть 3). **Теория:** лекция 3. **Инструменты:** Docker Desktop, Visual Studio 2026, PowerShell 7.

Подготовка. С этой лабы нужны готовые файлы из архива `kit.zip`: шаблон `.env`, позже — конфигурации GitHub, CI/CD и мониторинга. Распакуйте его, например, в `C:\dev\kit`. Путь в командах лаб обозначается `$K`:

```
cd C:\dev\demo-api
$K = "C:\dev\kit"                          # переменная живёт до закрытия терминала
Test-Path $K\.env.example                  # True
Get-ChildItem -Recurse $K | Unblock-File   # снять пометку «скачано из интернета», иначе скрипты из kit не запустятся
```

В следующих лабах, где нужен `$K`, задавайте его заново первой строкой.

Остановите сервис в Visual Studio (Shift+F5). Посмотрите учебные данные, которые загрузим в БД: `Get-Content db\seed.sql`.

Проверьте, свободен ли порт 5432 — его займёт PostgreSQL в контейнере:

```
Get-NetTCPConnection -State Listen -LocalPort 5432 -ErrorAction SilentlyContinue |
  Select-Object LocalPort, @{ n = 'Process'; e = { (Get-Process -Id $_.OwningProcess).ProcessName } }
```

Пусто — свободен. Часто его занимает PostgreSQL, установленный прямо в Windows. **Порт 5432 занят?** Везде в этой лабе используйте `127.0.0.1:15432:5432` вместо `127.0.0.1:5432:5432`, в `appsettings.Development.json` — `Port=15432`, а в `.env` (часть 3) — строку `DB_PORT=15432`.

Часть 1. БД в контейнере

1. БД без тома: данные пропадают

```
docker run -d --name demo-db -e POSTGRES_DB=demo -e POSTGRES_USER=demo -e POSTGRES_PASSWORD=demo postgres:18-alpine
do { Start-Sleep 1; docker exec demo-db pg_isready -h 127.0.0.1 -U demo -d demo *> $null } until ($LASTEXITCODE -eq 0)
ждём готовности
docker exec demo-db psql -U demo -d demo -c "CREATE TABLE notes (text text); INSERT INTO notes VALUES ('check');"
docker rm -f demo-db
docker run -d --name demo-db -e POSTGRES_DB=demo -e POSTGRES_USER=demo -e POSTGRES_PASSWORD=demo postgres:18-alpine
do { Start-Sleep 1; docker exec demo-db pg_isready -h 127.0.0.1 -U demo -d demo *> $null } until ($LASTEXITCODE -eq 0)
docker exec demo-db psql -U demo -d demo -c "SELECT * FROM notes;"      # ERROR: relation "notes" does not exist
docker rm -f demo-db
```

Таблицы нет. Почему? Запишите ответ.

2. Том и учебные данные

```
docker run -d --name demo-db `
  -e POSTGRES_DB=demo -e POSTGRES_USER=demo -e POSTGRES_PASSWORD=demo `
  -v demo-pgdata:/var/lib/postgresql `
  -v "${PWD}\db:/docker-entrypoint-initdb.d:ro" `
  -p 127.0.0.1:5432:5432 `
  postgres:18-alpine
do { Start-Sleep 1; docker exec demo-db pg_isready -h 127.0.0.1 -U demo -d demo *> $null } until ($LASTEXITCODE -eq 0)
docker logs demo-db 2>&1 | Select-String 'docker-entrypoint-initdb.d'     # running /docker-entrypoint-initdb.d/seed.s
docker exec demo-db psql -U demo -d demo -c "SELECT p.name, t.title, t.status FROM tasks t JOIN projects p ON p.id =
t.project_id;"
```

Флаг	Зачем
`-v demo-pgdata:/var/lib/postgresql`	**Том.** Файлы БД хранятся вне контейнера
`-v "${PWD}\db:/docker-entrypoint-initdb.d:ro"`	**Bind mount.** Папка проекта `db/` видна внутри контейнера; образ выполняет из неё скрипты при первой инициализации
`-p 127.0.0.1:5432:5432`	К БД сможет подключиться Visual Studio с вашего ПК

3. Данные переживают контейнер

```
docker exec demo-db psql -U demo -d demo -c "INSERT INTO tasks (project_id, title) VALUES (1, 'Задача из лабы 3');"
docker rm -f demo-db
docker run -d --name demo-db `
  -e POSTGRES_DB=demo -e POSTGRES_USER=demo -e POSTGRES_PASSWORD=demo `
  -v demo-pgdata:/var/lib/postgresql `
  -v "${PWD}\db:/docker-entrypoint-initdb.d:ro" `
  -p 127.0.0.1:5432:5432 `
  postgres:18-alpine
do { Start-Sleep 1; docker exec demo-db pg_isready -h 127.0.0.1 -U demo -d demo *> $null } until ($LASTEXITCODE -eq 0)
docker exec demo-db psql -U demo -d demo -c "SELECT id, title FROM tasks ORDER BY id;"    # ваша задача на месте
docker logs demo-db 2>&1 | Select-String 'Skipping initialization'
```

Контейнер новый, данные на месте, а `seed.sql` **не** выполнился повторно. **Почему?** Запишите ответ.

Часть 2. Сервис + БД

4. Сервис из Visual Studio → PostgreSQL

Сервису нужны две настройки: **какое хранилище** и **куда подключаться**. Обе — в `src/DemoApi/appsettings.Development.json`. Приведите эти строки к виду (остальное в файле не трогайте):

```
"Storage": "Postgres",
"ConnectionStrings": {
  "Db": "Host=localhost;Port=5432;Database=demo;Username=demo;Password=demo"
},
```

Часть строки подключения	Откуда значение
`Host=localhost;Port=5432`	Опубликованный порт контейнера `demo-db` из п. 2. Публиковали `15432` — пишите `Port=15432`
`Database=demo;Username=demo;Password=demo`	Переменные `POSTGRES_DB`, `POSTGRES_USER`, `POSTGRES_PASSWORD` из п. 2

Второй способ — переменные окружения. Те же настройки можно не писать в JSON, а задать в профиле запуска Visual Studio: `src/DemoApi/Properties/launchSettings.json` → профиль `http` → `environmentVariables`:

```
"environmentVariables": {
  "ASPNETCORE_ENVIRONMENT": "Development",
  "Storage": "Postgres",
  "ConnectionStrings__Db": "Host=localhost;Port=5432;Database=demo;Username=demo;Password=demo"
}
```

Переменные окружения переопределяют `appsettings*.json` (лекция 1, п. 11). Двойное подчёркивание `__` в имени — это уровень вложенности: `ConnectionStrings__Db` = `ConnectionStrings:Db`. Точно так же сервис настраивают в контейнере (п. 5) и в CI (тема 5).

> Сервис упал с `28P01: password authentication failed for user "demo"`? Он подключился не к той БД — чаще всего к PostgreSQL, установленному прямо в Windows на порту 5432. Проверьте порт: `docker ps --filter name=demo-db` покажет, какой порт опубликован у контейнера, и он должен совпадать с `Port=` в строке.

F5. В консоли сервиса — `Хранилище: PostgreSQL`. На странице Swagger:

- `GET /api/projects` — учебные проекты из `seed.sql`;
- создайте свой проект;
- перезапустите сервис (Shift+F5, F5) и снова `GET /api/projects` — **ваш проект на месте**. Сравните с лабой 1, п. 4.

Остановите сервис и запустите тесты (**Test Explorer → Run All**). Ожидаемо: **24 теста**, все зелёные. Теперь интеграционные тесты работают с PostgreSQL, и SQL-версия статистики из лабы 1 впервые проверена по-настоящему. Если её тест упал — исправьте SQL в `ProjectRepository.GetStatsAsync`.

Зачем: контроллеры, сервисы и тесты не менялись — поменялась одна настройка. Так работает разделение на слои и DI.

5. Сервис-контейнер → БД-контейнер

Соберите свежий образ — в нём ваша функция статистики:

```
docker build -t demo-api:2.0 .
```

Попытка 1 — включить PostgreSQL:

```
docker run -d --name api -p 8080:8080 -e Storage=Postgres demo-api:2.0
Start-Sleep 5
docker ps -a --filter name=api                                          # Exited
docker logs api 2>&1 | Select-String 'Unhandled exception'              # первая строка ошибки
```

Причина — в строке ошибки. Подсказка: в контейнере окружение `Production`, а строка подключения была только в `appsettings.Development.json`.

Попытка 2 — передать строку подключения к `localhost`:

```
docker rm api
docker run -d --name api -p 8080:8080 -e Storage=Postgres `
  -e "ConnectionStrings__Db=Host=localhost;Database=demo;Username=demo;Password=demo" `
  demo-api:2.0
Start-Sleep 5
docker logs api 2>&1 | Select-String 'Unhandled exception'
```

Снова падение, теперь ошибка подключения. **`localhost` внутри контейнера — это сам контейнер**, а не ваш ПК и не `demo-db`.

Попытка 3 — общая сеть и обращение по имени:

```
docker rm api
docker network create demo-net
docker network connect demo-net demo-db
docker run -d --name api --network demo-net -p 8080:8080 -e Storage=Postgres `
  -e "ConnectionStrings__Db=Host=demo-db;Database=demo;Username=demo;Password=demo" `
  demo-api:2.0
Start-Sleep 20
docker ps --filter name=api                    # (healthy)
irm http://localhost:8080/api/projects         # учебные проекты и ваш проект из п. 4
```

Сервис в контейнере и сервис в Visual Studio видят одни и те же данные: они теперь в БД, а не в процессе.

Проверьте ещё раз то, что в лабе 2 теряло данные:

```
docker stop api; docker start api; Start-Sleep 15
irm http://localhost:8080/api/projects         # данные на месте
```

Зачем столько попыток: это две самые частые ошибки при запуске сервиса в контейнере. Вы увидели обе и знаете, как их диагностировать по логам.
