Неделя 6. Лаба 3. Данные и Compose: PostgreSQL для сервиса задач

Часть 2 из 2: разделы «Часть 3. Docker Compose» — «Приложение: эталон `compose.yaml`». Первая часть лабы — в материалах предыдущей недели; перед продолжением выполните подготовку ниже.

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 3. Docker Compose

6. Разбираем ручной стенд

Посчитайте, сколько команд и флагов понадобилось в частях 1–2. Теперь всё это опишет один файл.

```
docker rm -f api demo-db
docker network rm demo-net
docker volume rm demo-pgdata     # compose создаст свой том
```

7. `compose.yaml` — напишите сами

Создайте `compose.yaml` в корне репозитория. Требования и зачем каждое нужно:

Проект: `name: demo` — чтобы имена контейнеров и томов не зависели от названия папки.

Сервис `api`:

Требование	Зачем
`image: ${API_IMAGE:-demo-api}:${API_TAG:-local}` и `build: .`	Локально образ собирается из Dockerfile; в лабе 5 пайплайн подставит образ из GHCR
Порт `8080:8080`	Доступ к API с вашего ПК
env `Storage: Postgres`	Хранилище — PostgreSQL
env `ConnectionStrings__Db` к хосту `db`, пароль `${DB_PASSWORD:-demo}`	Строка подключения из окружения; пароль не в файле
env `Chaos__Enabled: "true"`	Эндпоинты учебных сбоев для лаб 6–8
`depends_on: db: condition: service_healthy`	Сервис не стартует, пока БД не готова
`restart: unless-stopped`	Упавший контейнер поднимется сам (лаба 6)
Лимиты 1 CPU и 256M + `memswap_limit: 256M`	Контейнер не съест память всего ПК

Логи `json-file`, `max-size: 10m`, `max-file: 3`	Логи не заполнят диск

Сервис `db`:

Требование	Зачем
`postgres:18-alpine`, БД `demo`, пользователь `demo`, пароль `${DB_PASSWORD:-demo}`	Та же БД, что в части 1
Том `pgdata` → `/var/lib/postgresql`	Данные переживут пересоздание контейнера
`./db` → `/docker-entrypoint-initdb.d`, только чтение	Учебные данные при первом запуске
healthcheck `pg_isready -h 127.0.0.1 -U demo -d demo` раз в 5 с	По нему `depends_on` понимает, что БД готова
Порт `127.0.0.1:${DB_PORT:-5432}:5432`	Visual Studio и тесты подключаются с вашего ПК
`restart: unless-stopped`	Как у сервиса

Застряли больше 15 минут — сверьтесь с приложением в конце лабы.

8. Секреты и запуск

```
Copy-Item $K\.env.example .env
notepad .env                                # посмотрите содержимое
docker compose config                       # итоговый файл с подставленными значениями
docker compose up -d --build --wait
docker compose ps                           # оба сервиса (healthy)
irm http://localhost:8080/api/projects      # учебные проекты
```

Пароль оставьте `demo`: тот же пароль прописан в `appsettings.Development.json`, с ним к этой БД подключаются Visual Studio и тесты. В реальном проекте пароль разработчика хранят в User Secrets, а не в JSON.

9. Жизнь данных

```
irm -Method Post http://localhost:8080/api/projects -ContentType application/json -Body '{"name":"Compose"}'
(irm http://localhost:8080/api/projects).Count
docker compose down;    docker compose up -d --wait;  (irm http://localhost:8080/api/projects).Count   # ?
docker compose down -v; docker compose up -d --wait;  (irm http://localhost:8080/api/projects).Count   # ?
```

Запишите три числа и объясните каждое, в том числе откуда берутся проекты после `down -v`.

10. Сеть и готовность

```
docker network ls                                                   # demo_default
docker compose exec api getent hosts db                             # IP сервиса db
docker compose stop db
curl.exe -s -w " %{http_code}`n" http://localhost:8080/health/live   # Healthy 200
curl.exe -s -w " %{http_code}`n" http://localhost:8080/health/ready  # Unhealthy 503
curl.exe -s -o NUL -w "%{http_code}`n" http://localhost:8080/api/tasks   # 500
docker compose ps                                                   # api всё ещё healthy
docker compose start db
```

Почему сервис считается healthy, если БД лежит? Ответ — в лекции 6. Сформулируйте свой вариант.

11. Разработка с compose

Рабочий режим разработчика: БД в контейнере, код в IDE.

```
docker compose stop api          # освободить порт 8080 для Visual Studio
```

F5 в Visual Studio — сервис подключается к БД из compose. Выполните пару запросов, затем запустите тесты. Вернуть весь стек: остановите отладку и выполните `docker compose up -d --wait`.

Сдача

- [ ] Ответы на вопросы из пп. 1 и 3
- [ ] Скриншот: `Хранилище: PostgreSQL` в консоли и данные, пережившие перезапуск сервиса (п. 4); 24 зелёных теста
- [ ] Строки ошибок из попыток 1 и 2 (п. 5) и объяснение каждой
- [ ] Ваш `compose.yaml`
- [ ] Три числа из п. 9 с объяснением
- [ ] Коды live, ready и `/api/tasks` при остановленной БД
- [ ] `.env` не попадёт в git: `Select-String '^\.env$' .gitignore` находит строку `.env`

☆ Дополнительно

Добавьте сервис `adminer` (образ `adminer`, порт `8083:8080`) и посмотрите таблицы `projects` и `tasks` через веб-интерфейс. Какой хост указать в форме входа и почему не `localhost`?

Приложение: эталон `compose.yaml`

Открывайте только после своей попытки.

```yaml
name: demo

services:
  api:
    image: ${API_IMAGE:-demo-api}:${API_TAG:-local}
    build:
      context: .
      args:
        APP_VERSION: ${API_TAG:-local}
    ports:
      - "8080:8080"
    environment:
      Storage: Postgres
      ConnectionStrings__Db: "Host=db;Database=demo;Username=demo;Password=${DB_PASSWORD:-demo}"
      Chaos__Enabled: "true"
    depends_on:
      db:
        condition: service_healthy
    restart: unless-stopped
    memswap_limit: 256M
    deploy:
      resources:
        limits:
          cpus: "1.0"
          memory: 256M
    logging:
      driver: json-file
      options:
        max-size: "10m"
        max-file: "3"

  db:
    image: postgres:18-alpine
    environment:
      POSTGRES_DB: demo
      POSTGRES_USER: demo
      POSTGRES_PASSWORD: ${DB_PASSWORD:-demo}
    ports:
      - "127.0.0.1:${DB_PORT:-5432}:5432"
    volumes:
      - pgdata:/var/lib/postgresql
      - ./db:/docker-entrypoint-initdb.d:ro
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -h 127.0.0.1 -U demo -d demo"]
      interval: 5s
      timeout: 3s
      retries: 10
    restart: unless-stopped

volumes:
  pgdata:
```