Тема 3

Данные и Docker Compose: PostgreSQL, тома, сети



Даём сервису настоящую базу и описываем окружение одним файлом

План лекции

Сегодня

1. Почему данные пропадают
2. Том и bind mount
3. PostgreSQL в контейнере
4. Сеть: как контейнеры находят друг друга
5. Docker Compose



После лекции вы сможете

- Хранить данные контейнерных приложений правильно
- Переключить сервис на PostgreSQL
- Соединить контейнеры сетью
- Описать окружение в `compose.yaml`



Контейнер — одноразовый.
Данные — отдельно.

Контейнер сервиса можно удалить и создать заново в любой момент — так и работает деплой.
Данные должны жить там, где удаление контейнера их не заденет

Где хранить данные контейнера

Записываемый слой

Внутри контейнера. Пропадает при `docker rm`. Только для временных файлов

Том (volume)

Каталог под управлением Docker. Живёт после `docker rm`. Для данных БД: `demo-pgdata`

Bind mount

Папка вашего диска внутри контейнера. Для конфигов и скриптов: `db/` с учебными данными

PostgreSQL в контейнере


```
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
```

1. **Переменные**

   При первом старте создаются БД и пользователь

2. **Том**

   Файлы БД вне контейнера

3. **Bind mount**

   `seed.sql` создаёт таблицы и учебные данные

4. **Порт**

   Для Visual Studio, но не из сети

Инициализация — только на пустом томе

Первый старт

Том пуст: создаётся БД, выполняется `seed.sql`

Следующие старты

Том с данными: `Skipping initialization`

Ловушка

Поменяли пароль или seed — ничего не изменится

Хотите начать с нуля — удалите том. Данные при этом пропадут

Переключаем сервис на PostgreSQL

Две настройки: `Storage=Postgres` и строка подключения `ConnectionStrings:Db`

Из Visual Studio

- `appsettings.Development.json`
- `"Storage": "Postgres"`
- БД по адресу `localhost:5432` — через опубликованный порт

В контейнере

- Переменные окружения, образ не меняем
- `-e Storage=Postgres`
- `-e ConnectionStrings__Db=Host=demo-db;...`

Код сервиса не меняется: сработали слои и DI из темы 1

Сеть: localhost — это ловушка

`Host=localhost` в контейнере

- `localhost` внутри контейнера — сам контейнер
- PostgreSQL там нет
- Сервис падает: Failed to connect

Общая сеть `demo-net`

- Docker держит встроенный DNS
- Контейнеры находят друг друга по имени
- Сервис подключается к `demo-db:5432`

Visual Studio → `localhost:5432` (порт). Контейнер → `demo-db:5432` (сеть)

Лаба 3: три попытки запустить сервис с БД

Только Storage

«Не задана строка подключения»: в контейнере окружение Production

Host=localhost

«Failed to connect 127.0.0.1:5432»: localhost — сам контейнер

Сеть + Host=demo-db

healthy, в API учебные данные и проекты из Visual Studio

Две самые частые ошибки при запуске сервиса в контейнере — и как их читать по логам

Зачем Docker Compose

10+

флагов и команд, чтобы вручную поднять сервис с БД

1

файл `compose.yaml` в git рядом с кодом

1

команда: `docker compose up -d --build --wait`

Окружение описано декларативно: его можно отревьюить и поднять у любого члена команды

compose.yaml сервиса задач

```
name: demo
services:
  api:
    build: .
    ports: ["8080:8080"]
    environment:
      Storage: Postgres
      ConnectionStrings__Db: "Host=db;...;Password=$
{DB_PASSWORD}"
    depends_on:
      db: { condition: service_healthy }
    restart: unless-stopped
  db:
    image: postgres:18-alpine
    environment: { POSTGRES_DB: demo, POSTGRES_USER: demo,
                   POSTGRES_PASSWORD: "${DB_PASSWORD}" }
    volumes:
      - pgdata:/var/lib/postgresql
      - ./db:/docker-entrypoint-initdb.d:ro
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -h 127.0.0.1 -U demo"]
volumes:
  pgdata:
```

1. **Имя сервиса = DNS**
   `Host=db` — сеть создаётся сама
2. **depends_on healthy**
   Ждём готовности БД, а не старта
3. **Тома**
   Данные и учебные скрипты
4. **${DB_PASSWORD}**
   Секрет из `.env`, не из файла

Команды Compose

Команда	Что делает
`docker compose up -d --build --wait`	Собрать, запустить, дождаться healthy
`docker compose ps` / `logs -f api`	Состояние / логи
`docker compose exec db psql -U demo -d demo`	Консоль БД
`docker compose stop` / `start`	Остановить / запустить
`docker compose down`	Удалить контейнеры и сеть. Тома остаются
`docker compose down -v`	И тома тоже — данные пропадут
`docker compose config`	Итоговый файл с подставленными значениями

Контейнеры получают имена `demo-api-1`, `demo-db-1`

Compose в разработке

Режим разработчика

- `docker compose up -d --wait db`
- Только БД в контейнере
- Сервис — F5 в Visual Studio с отладчиком

Как в проде

- `docker compose up -d --build --wait`
- Весь стек в контейнерах
- Проверка перед PR

Зависимости — в контейнерах, свой код — в IDE

Типичные ошибки

- БД без тома: `docker rm` — и данные исчезли
- `Host=localhost` в строке подключения контейнера
- Порт 5432 занят локальным PostgreSQL: используйте 15432
- Поменяли пароль на непустом томе — БД ждёт старый
- `depends_on` без `service_healthy`: сервис падает на старте
- `down -v` «на всякий случай»

Проверьте себя

1. Почему том решает проблему потери данных?
2. Чем том отличается от bind mount?
3. Почему `seed.sql` не выполнится повторно?
4. Адрес БД из Visual Studio и из контейнера — почему разный?
5. Что переживёт `down`, а что — `down -v`?



Практика

Лаба 3. PostgreSQL для сервиса

Даём сервису настоящую БД и собираем окружение в Compose

- PostgreSQL в контейнере: том, учебные данные, порт
- Сервис из Visual Studio на PostgreSQL: данные переживают перезапуск
- Сервис-контейнер: две ошибки подключения и сеть
- Пишем `compose.yaml` и поднимаем всё одной командой