МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»  
(ФГБОУ ВО «ВГТУ», ВГТУ)

Факультет информационных технологий и компьютерной безопасности

Кафедра компьютерных интеллектуальных технологий проектирования

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6

По дисциплине: «Системы хранения и обработки данных»

Тема: «Развертывание СУБД Postgres с использованием средств автоматизации развертывания и управления приложениями»

Выполнил работу студент группы змИИВТ-241: Авсянкина М.А.

подпись, дата

Принял: Короленко В.В.

подпись, дата

Воронеж 2024

Цель работы: изучить основы работы программного средства для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации Docker на примере решения задачи развёртывания СУБД Postgres.

Основные задачи:

• установка приложения для работы с Docker-контейнерами;

• установка программного средства для работы с СУБД DBeaver;

• скачивание docker-образа с СУБД postgres;

• создание Dockerfile;

• создание образа на основе Dockerfile;

• запуск контейнера для развёртывания СУБД Postgres;

• подключение к работающему контейнеру и запуск интерфейса psql;

• создание контейнера с томом (volume);

• создание контейнера с использованием файла docker-compose.yml

Ход работы:

Docker – это мощный инструмент для управления контейнерами, который позволяет разработчикам и системным администраторам эффективно развертывать и масштабировать приложения. Docker предоставляет удобный способ упаковки приложений вместе с их зависимостями в изолированные контейнеры, что делает их легкими для переноса и запуска в различных средах.

Начнем работу в терминале, с открытой программой docker.desktop. Для начала скачиваем образ, с помощью команды:

docker pull postgres

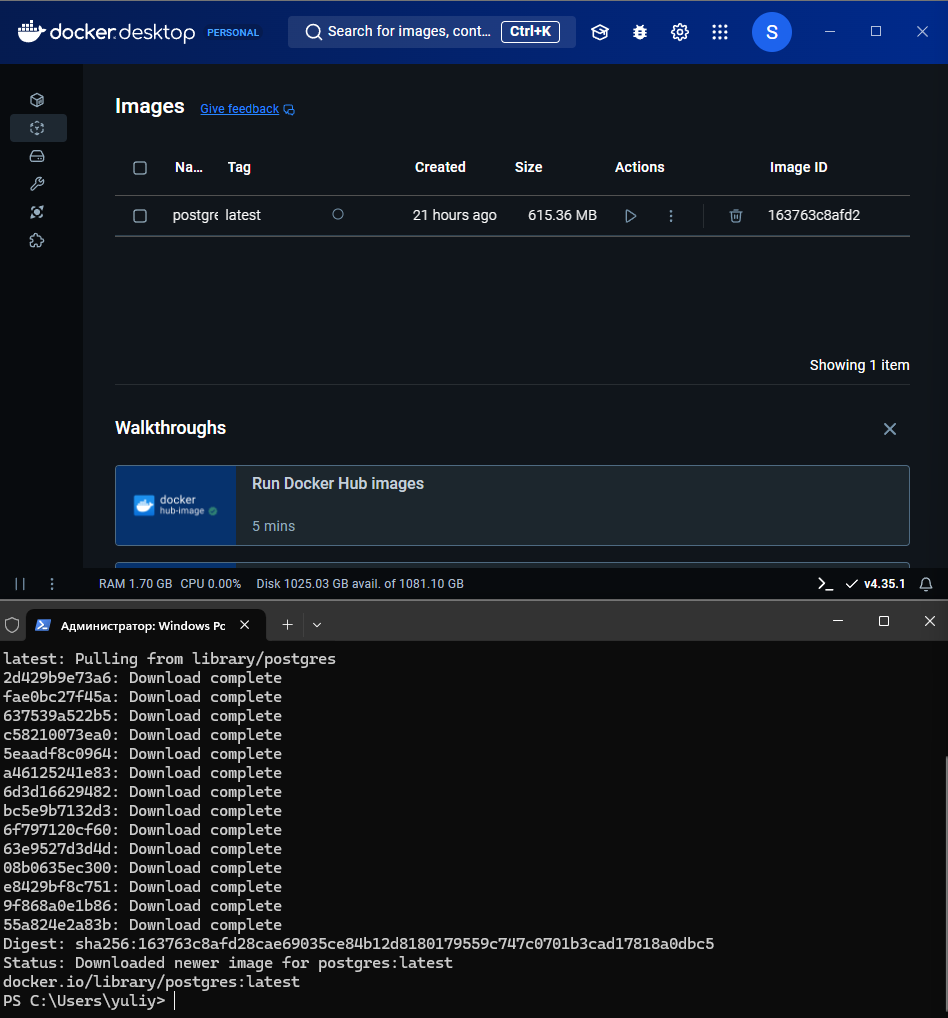


Рисунок 1 – Скачивание образа через терминал

Как видно, из рисунка 1, после запуска команды в терминале, в докере скачался образ “postgres”.

Создаем Dockerfile со следующим содержанием:

FROM postgres:latest

ENV POSTGRES\_PASSWORD=dbpass

ENV POSTGRES\_USER=dbuser

ENV POSTGRES\_DB=dbname

COPY init\_scripts/init.sql /docker-entrypoint-initdb.d/init.sql

Описание каждой строчки Dockerfile, что она означает и какую задачу помогает решать:

1. FROM postgres:latest
   * Определяет образ, который будет использован в качестве основы для контейнера. В данном случае используется последний доступный образ PostgreSQL.
2. ENV POSTGRES\_PASSWORD=dbpass
   * Устанавливает переменную среды POSTGRES\_PASSWORD и присваивает ей значение dbpass. Переменная POSTGRES\_PASSWORD используется для установки пароля суперпользователя (postgres) в PostgreSQL.
3. ENV POSTGRES\_USER=dbuser
   * Устанавливает переменную среды POSTGRES\_USER и присваивает ей значение dbuser. Переменная POSTGRES\_USER используется для установки имени пользователя, который будет суперпользователем (postgres) в PostgreSQL.
4. ENV POSTGRES\_DB=dbname
   * Устанавливает переменную среды POSTGRES\_DB и присваивает ей значение dbname. Переменная POSTGRES\_DB используется для установки имени базы данных, которую будет использовать контейнер.
5. COPY init\_scripts/init.sql /docker-entrypoint-initdb.d/init.sql
   * Копирует файл init.sql из каталога init\_scripts в каталог /docker-entrypoint-initdb.d. Этот каталог используется Docker для автоматической инициализации базы данных при запуске контейнера.

По этой же системе описываем создание файла init.sql:

CREATE TABLE IF NOT EXISTS public.index\_mass (

user\_id BIGINT,

weight BIGINT,

height BIGINT

);

INSERT INTO public.index\_mass (user\_id, weight, height) VALUES

(1, 75, 175),

(2, 60, 182),

(3, 93, 181);

1. CREATE TABLE IF NOT EXISTS public.index\_mass (

user\_id BIGINT,

weight BIGINT,

height BIGINT

);

Эта команда создает таблицу index\_mass в базе данных public, если она еще не существует. Таблица имеет три столбца: user\_id типа BIGINT, weight типа BIGINT и height типа BIGINT.

1. INSERT INTO public.index\_mass (user\_id, weight, height) VALUES

(1, 75, 175),

(2, 60, 182),

(3, 93, 181);

Эта команда вставляет три записи в таблицу index\_mass. Каждая запись состоит из трех значений: user\_id, weight и height.

После чего, с помощью команды ниже, которую мы запускаем в терминале из каталога, в котором находится Dockerfile.

docker build -t d\_img:latest

Создаем непосредственно образ, с именем d\_img (рисунок 2-3).

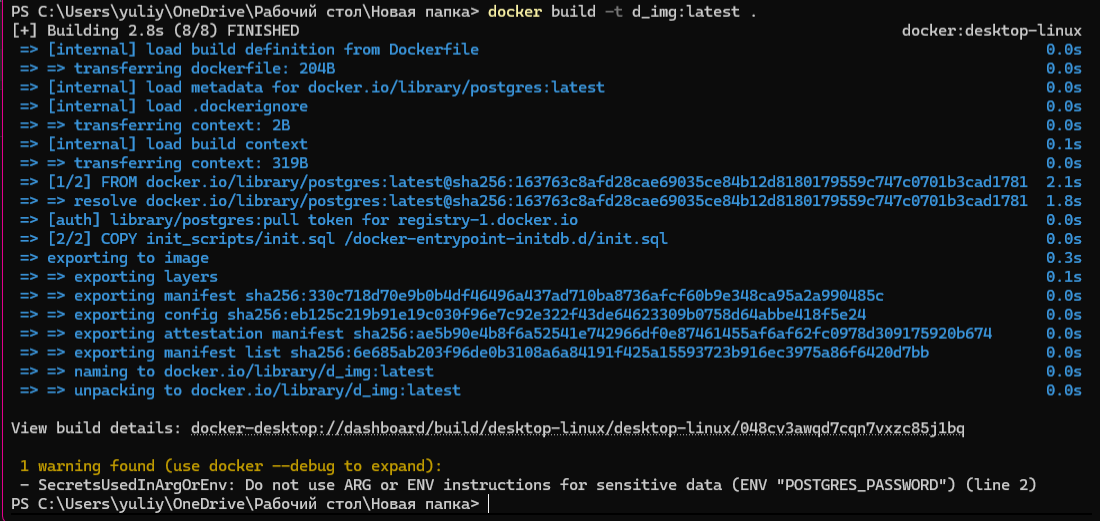


Рисунок 2 – Создание образа через терминал

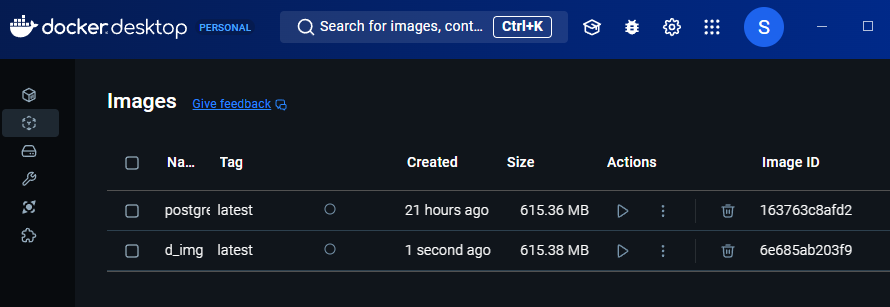


Рисунок 3 – Отображение созданного образа в docker

Чтобы воспользоваться созданной СУБД необходимо запустить контейнер с именем «d\_cont» на основе образа «d\_img:latest» с помощью команды: «docker run -d -p 5432:5432 --name d\_cont d\_img:latest».

Команда **docker run -d -p 5432:5432 --name d\_cont d\_img:latest** состоит из следующих компонентов:

1. **docker run –** Основная команда Docker, которая запускает контейнер на основе указанного образа.
2. **-d –** Запуск контейнера в фоновом режиме (detached mode). Контейнер продолжит выполнение в фоне, а терминал освободится для дальнейших команд.
3. **-p 5432:5432 –** Привязка порта. Здесь порт 5432 на хост-машине привязывается к порту 5432 внутри контейнера. Это позволяет обращаться к сервисам внутри контейнера через указанный порт на хосте.
4. **--name d\_cont –** Указание имени для контейнера. Имя контейнера должно быть уникальным среди запущенных контейнеров. Это удобно для дальнейшего обращения к контейнеру по имени вместо идентификатора.
5. **d\_img:latest –** Образ контейнера, на основе которого создается и запускается контейнер. В данном случае используется образ с именем d\_img и тегом latest. Тег latest указывает на последнюю доступную версию образа.

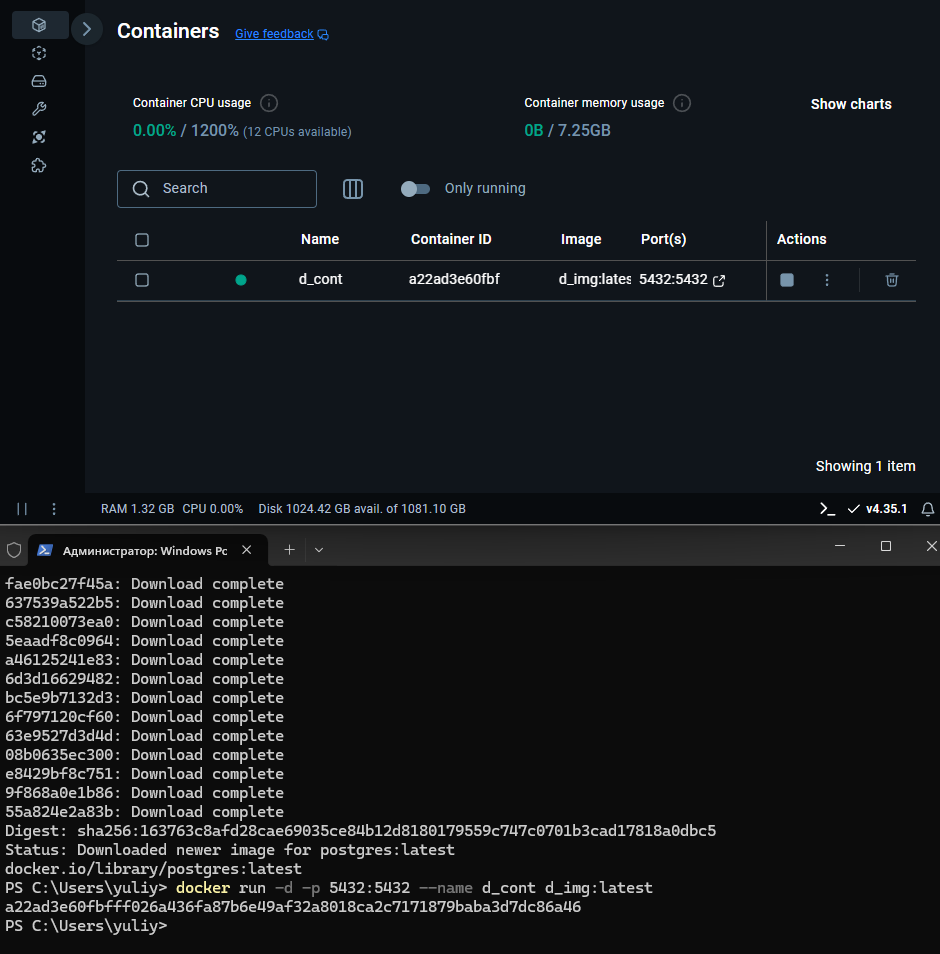


Рисунок 4 – Созданный Docker-контейнер

Далее подключаем базу данных с параметрами, указанными в Dockerfile (рисунок 4), чтобы проверить развернутую СУБД в DBeaver.

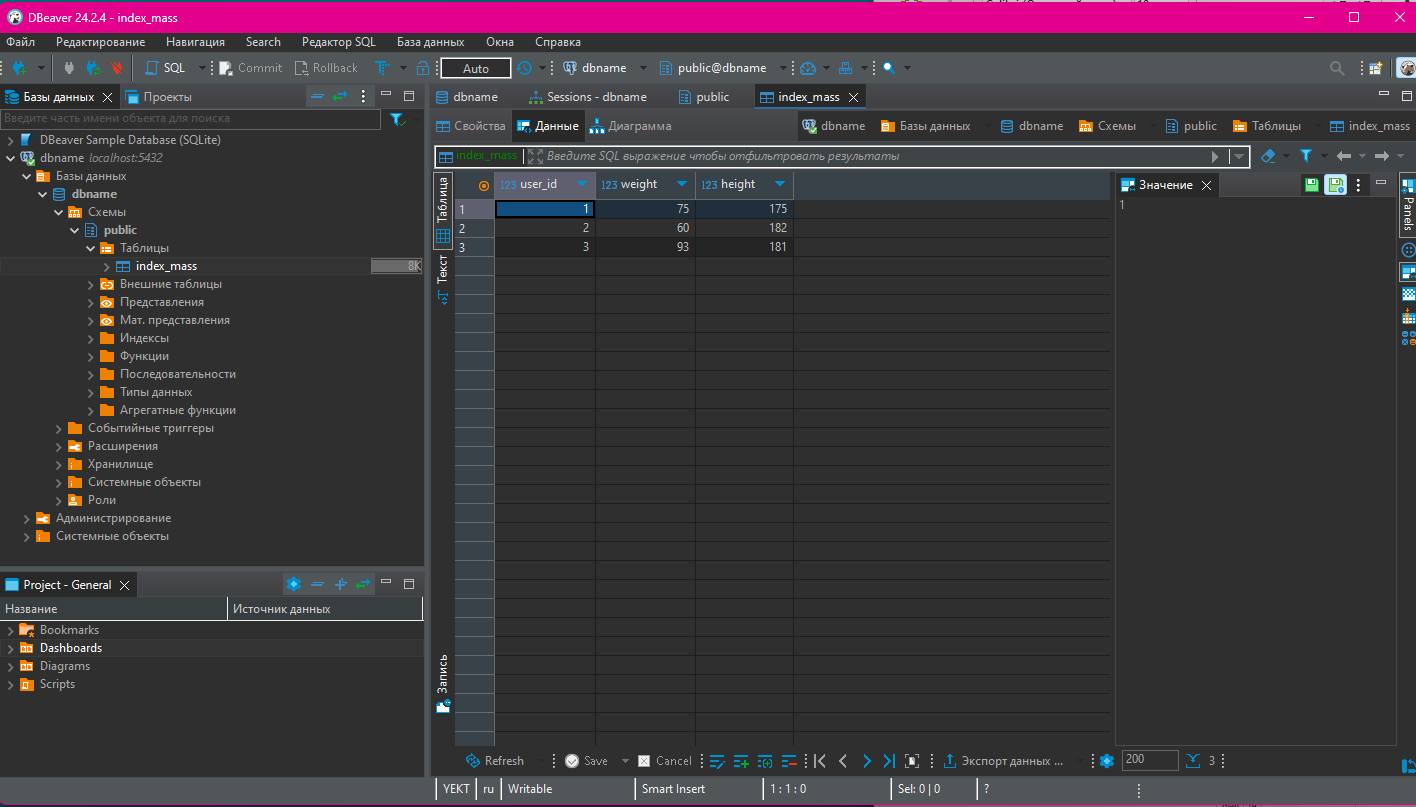


Рисунок 5 – Развернутая СУБД через DBeaver

Теперь подключаемся к работающему контейнеру, чтобы иметь возможность запускать интерфейс psql и вносить новые данные сразу же:

«docker exec -it d\_cont psql -d -U usr dbn».

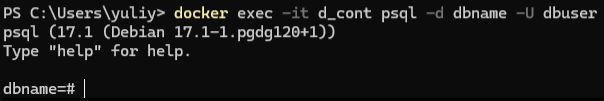


Рисунок 6 – Подключение к работающему контейнеру

Команда **docker exec -it d\_cont psql -d dbname -U dbuser** выполняет следующее:

1. **docker exec** – Выполняет команду внутри работающего контейнера Docker. В данном случае мы выполняем команду внутри контейнера с именем d\_cont.
2. **-it** – это комбинация флагов:

-i (interactive): Включает интерактивный режим, позволяющий передавать сигналы стандартного ввода и вывода между контейнером и терминалом.

-t (tty): Выделяет псевдотерминал, что делает взаимодействие более удобным, особенно при работе с инструментами, требующими взаимодействия с пользователем.

1. **d\_cont** – Имя контейнера, внутри которого будет выполнена команда. В вашем случае это d\_cont, которое было задано при запуске контейнера.
2. **Psql** – Утилита командной строки для взаимодействия с базой данных PostgreSQL. Она позволяет выполнять SQL-команды, просматривать структуры таблиц, изменять данные и многое другое.
3. **-d dbname** – Указывает имя базы данных, к которой нужно подключиться. В данной команде dbname — это имя базы данных.
4. **-U dbuser** – Задает имя пользователя, под которым нужно подключаться к базе данных. В этом случае используется пользователь dbuser.

Теперь мы подключились к контейнеру. Чтобы работать в интерактивной среде, надо ввести команду «\l» и выводим на терминале список баз данных.

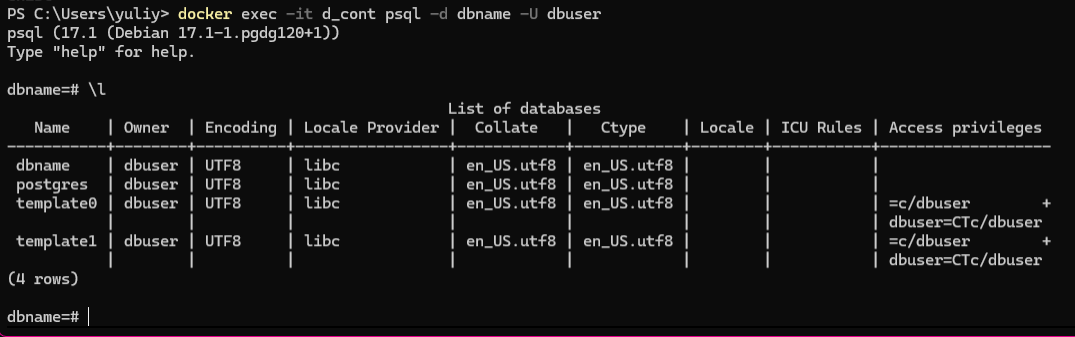


Рисунок 7 – Список баз данных

Рассмотрим такую ситуацию, когда нам надо удалить контейнер, с измененными данными в процессе работы, но при этом, необходимо, чтобы они были доступны после удаления. Для решения такой задачи, удобно использовать созданный том (volume).

Для этого, в Dockerfile необходимо добавить инструкцию вызова создания контейнера в volume, используем команду:

VOLUME /d\_data:/var/lib/postgresql/data

Теперь можно удалять контейнер. Для этого, сначала останавливаем контейнер (рисунок 8).

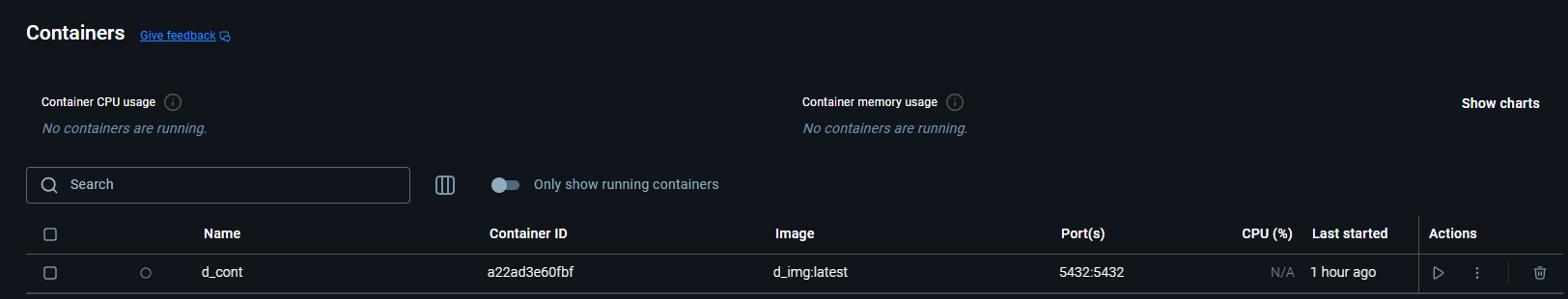


Рисунок 8 – Остановка контейнера

После чего, выполняем само удаление контейнера, с помощью команды **rm d\_cont**. Как видно из рисунка 9, контейнер удален и не отображается в docker.

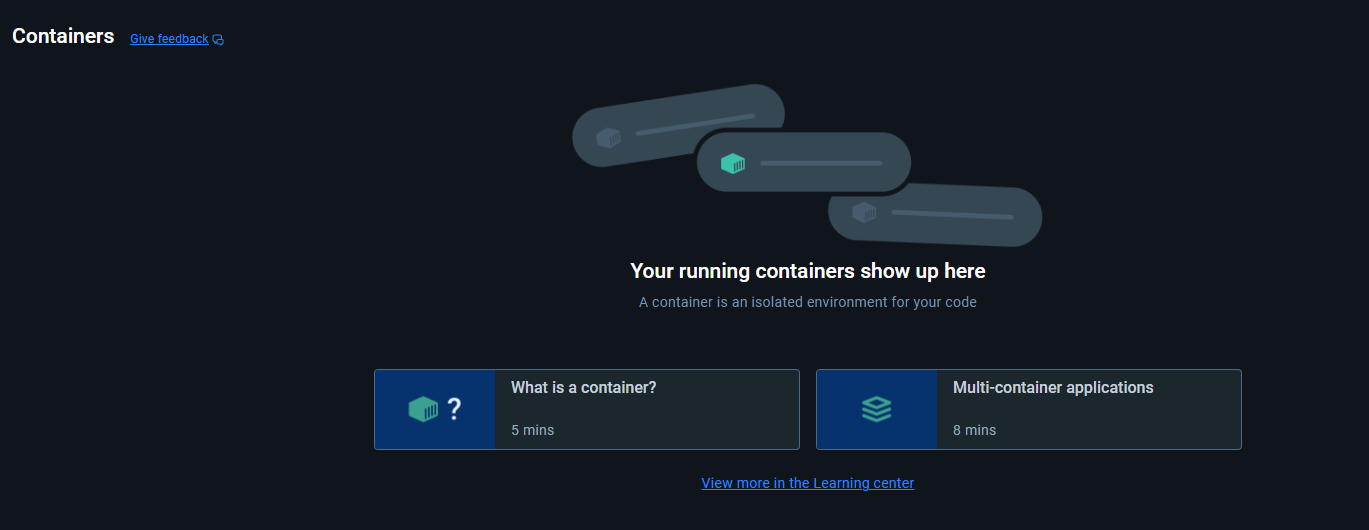


Рисунок 9 – Проверка удаления контейнера

И теперь запускаем создание нового контейнера (рисунок 10-11), с образом d\_img:latest, командой docker run -d -p 5432:5432 --name d\_cont d\_img:latest -v /d\_data:/var/lib/postgresql/data, и не беспокоиться, что какие-либо внесенные изменения в данные, будут утеряны.

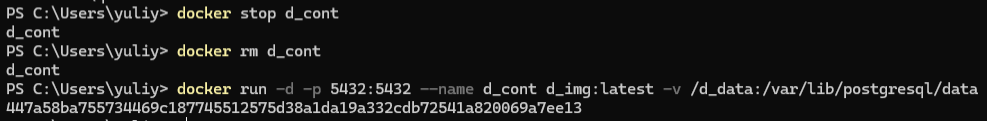


Рисунок 10 – Создание контейнера с томом из терминала

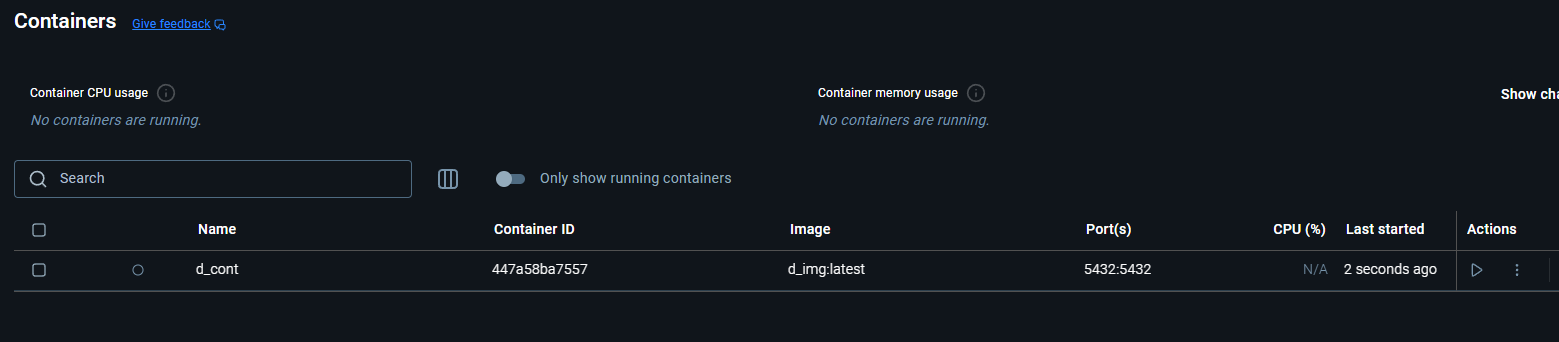


Рисунок 11 – Новый созданный контейнер

Команда **docker run -d -p 5432:5432 --name d\_cont d\_img:latest -v /d\_data:/var/lib/postgresql/data**

Состоит из следующих компонентов и выполняет такие функции как:

1. **docker run** – Основная команда Docker, которая запускает контейнер на основе указанного образа.
2. **-d** – Запуск контейнера в фоновом режиме (detached mode). Контейнер продолжит выполнение в фоне, а терминал освободится для дальнейших команд.
3. **-p 5432:5432** – Привязка порта. Здесь порт 5432 на хост-машине привязывается к порту 5432 внутри контейнера. Это позволяет обращаться к сервисам внутри контейнера через указанный порт на хосте.
4. **--name d\_cont** – Указание имени для контейнера. Имя контейнера должно быть уникальным среди запущенных контейнеров. Это удобно для дальнейшего обращения к контейнеру по имени вместо идентификатора.
5. **d\_img:latest** – Образ контейнера, на основе которого создается и запускается контейнер. В данном случае используется образ с именем d\_img и тегом latest. Тег latest указывает на последнюю доступную версию образа.
6. **-v /d\_data:/var/lib/postgresql/data** – Монтирование тома (volume). Здесь создается связь между папкой /d\_data на хост-компьютере и папкой /var/lib/postgresql/data внутри контейнера. Все изменения, происходящие в папке /var/lib/postgresql/data внутри контейнера, будут сохраняться в папке /d\_data на хост-системе. Это гарантирует, что данные будут сохранены даже после удаления контейнера.

Создаем текстовый файл cmd.txt, в котором описываем команды, которые необходимы для развертывания базы данных Postgres с помощью Dockerfile (рисунок 10).

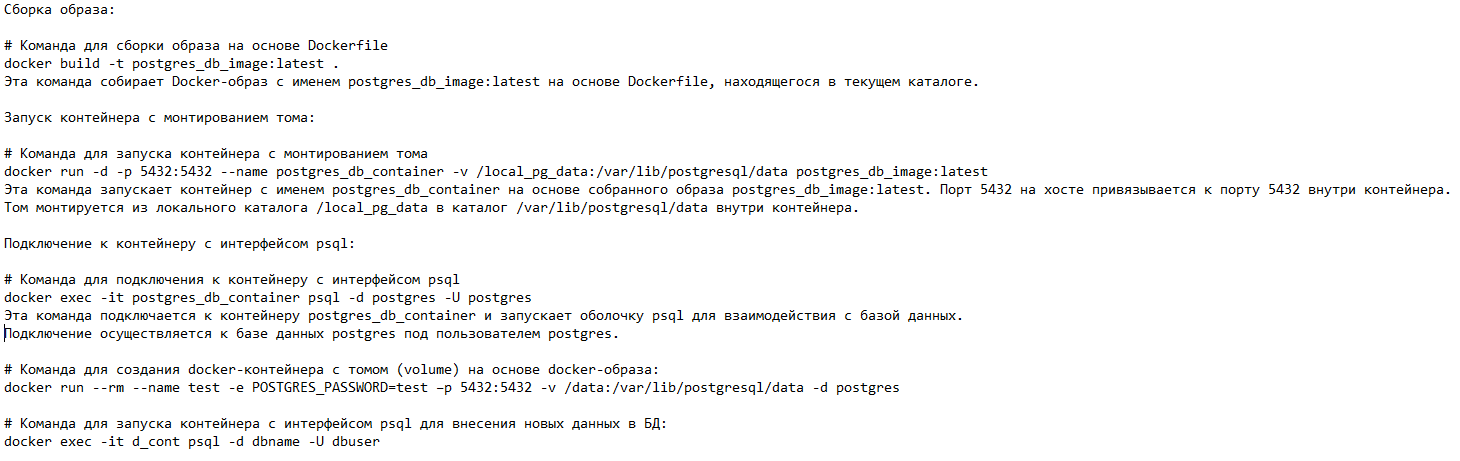


Рисунок 10 – Описание команд для развертывания базы данных Postgres

В данном Dockerfile описываем команды, для создания образа, запуска контейнера, создания тома. Также указываем дополнительно команду для создания docker-контейнера с томом (volume) на основе docker-образа и команду для запуска контейнера с интерфейсом psql для внесения новых данных в БД.

Создаем контейнер с использованием файла docker-compose.yml (рисунок 11-12).

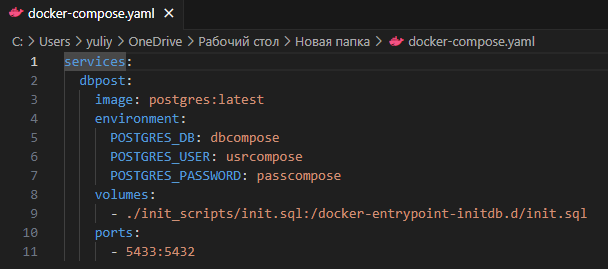


Рисунок 11 – Содержание файла docker-compose.yml

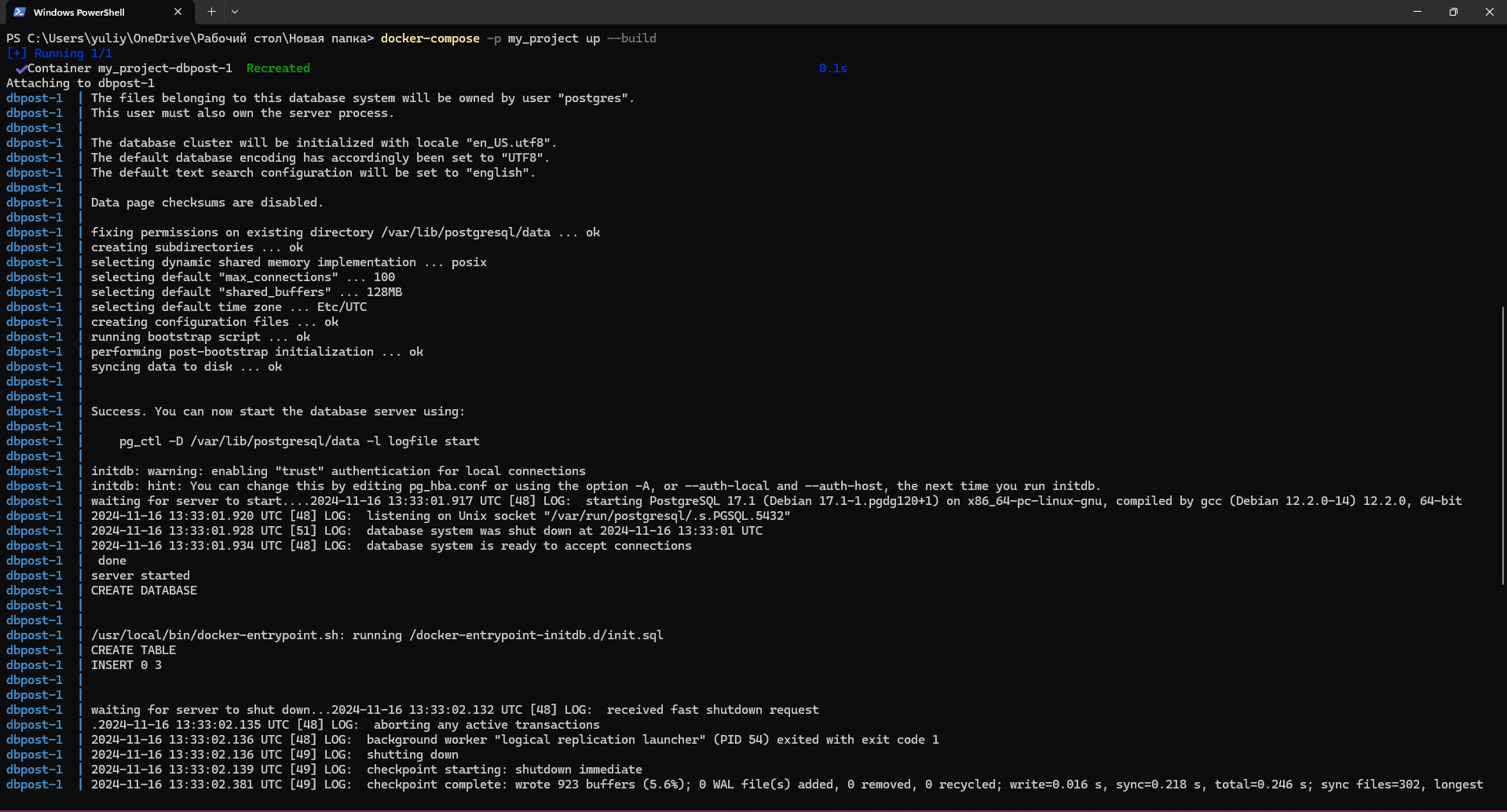


Рисунок 12 – Создаем новый контейнер

# Файл docker-compose.yml предназначен для описания конфигурации одного или нескольких сервисов (контейнеров) в формате YAML, которые будут запущены с помощью инструмента docker compose. Разбор файла по разделам:

**services:** Определяет сервисы (контейнеры), которые будут запущены. В данном случае определяется один сервис с именем dbpost.

**image: postgres:latest** Определяет Docker-образ, на основе которого будет создан контейнер. В данном случае используется образ postgres:latest, что означает использование последней версии образа PostgreSQL.

**environment:** Определяет переменные среды, которые будут использоваться в контейнере. В данном случае определяются три переменные:

**POSTGRES\_DB: dbcompose**

**POSTGRES\_USER: usrcompose**

**POSTGRES\_PASSWORD: passcompose** Эти переменные задают имя базы данных, имя пользователя и пароль для подключения к базе данных PostgreSQL.

**volumes:** Определяет тома (тома), которые будут использованы в контейнере. В данном случае определён один том:

**./init\_scripts/init.sql:/docker-entrypoint-initdb.d/init.sql** Этот том связывает файл init.sql из локального пути ./init\_scripts/init.sql с путём /docker-entrypoint-initdb.d/init.sql внутри контейнера. Это позволяет использовать файл init.sql для инициализации базы данных. Внутри контейнера этот файл будет использован для инициализации базы данных при первом запуске контейнера.

**ports**: **5433:5432** Определяет порты, которые будут привязаны к портам контейнера. В данном случае порт 5433 на хост-компьютере привязывается к порту 5432 внутри контейнера. Это позволяет подключаться к базе данных PostgreSQL, работающей в контейнере, через порт 5432 на хост-компьютере.

Процесс работы файла docker-compose.yaml

1. Запуск контейнера

Когда запускается команда docker-compose up, Docker Compose читает файл docker-compose.yaml и запускает контейнеры, указанные в файле. В данном случае будет запущен контейнер с именем dbpost, основанный на образе PostgreSQL, с привязанным портом 5432 и установленными переменными среды для подключения к базе данных.

1. Привязка порта

Порт 5432 на хост-компьютере привязывается к порту 5432 внутри контейнера. Это позволяет подключаться к базе данных PostgreSQL, работающей в контейнере, через порт 5432 на хост-компьютере.

1. Монтирование тома

Файл init.sql из локального пути ./init\_scripts/init.sql монтируется внутрь контейнера в путь /docker-entrypoint-initdb.d/init.sql. Это позволяет использовать файл init.sql для инициализации базы данных. Внутри контейнера этот файл будет использован для инициализации базы данных при первом запуске контейнера.

Результат работы файла docker-compose.yaml

1. Запуск контейнера Запускается контейнер с именем dbpost, основанный на образе PostgreSQL, с привязанными портами и томами, а также с установленными переменными среды для подключения к базе данных.
2. Инициализация базы данных В первый раз, когда контейнер запускается, файл init.sql используется для инициализации базы данных. В дальнейшем контейнер продолжает работать с существующей базой данных.
3. Доступ к базе данных Доступ к базе данных возможен через порт 5432 на хост-компьютере. Пользователи могут подключаться к базе данных, работающей в контейнере, используя PostgreSQL.

Проверяем в docker.desktop, появился ли у нас новый контейнер (рисунок 13).

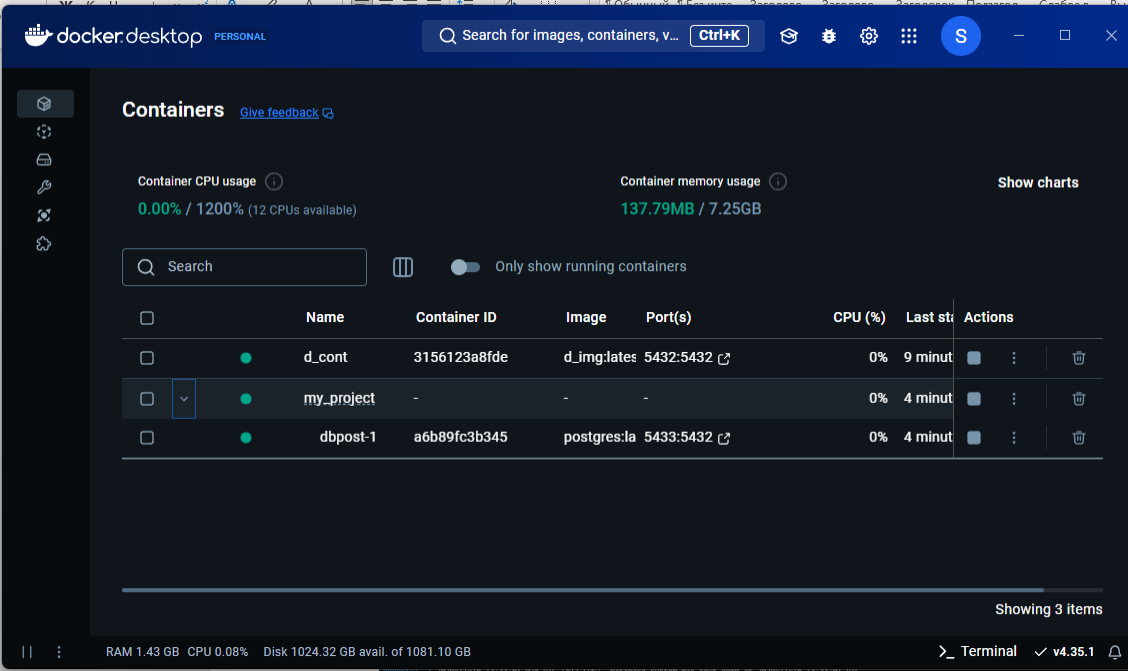


Рисунок 13 – Новый контейнер my\_project

Как видно из рисунка 13, был создан новый контейнер, из файла docker-compose.yml.

Вывод: таким образом, мы изучили основы работы программного средства для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации Docker на примере решения задачи развёртывания СУБД Postgres.

Ответы на контрольные вопросы:

1. Что такое Docker?

Docker — это платформа для разработки, доставки и запуска приложений в виде контейнеров. Контейнеры позволяют упаковывать приложение со всеми необходимыми зависимостями и библиотеками в изолированную среду, которая может быть легко развернута на любой машине, поддерживающей Docker.

2. Зачем нужен Docker?

Docker помогает решать множество задач:

Обеспечивает переносимость приложений: контейнеры могут быть запущены на разных операционных системах и машинах без изменений в коде приложения.

Упрощает управление зависимостями: каждая зависимость включена прямо в контейнер, что исключает конфликты версий.

Улучшает изоляцию: каждый контейнер работает в своей среде, что снижает риск влияния одного приложения на другое.

Повышает производительность: контейнеры требуют меньше ресурсов, чем виртуальные машины, поскольку они используют ядро хоста напрямую.

3. Что такое docker-образ?

Docker-образ — это шаблон, используемый для создания контейнеров. Образ включает в себя все файлы, зависимости и инструкции, необходимые для запуска приложения. Образы могут содержать слои, что позволяет эффективно управлять ими и оптимизировать хранение.

4. Что такое docker-контейнер?

Docker-контейнер — это исполняемая единица, созданная на основе образа. Контейнер представляет собой изолированное окружение, в котором выполняется ваше приложение. Он содержит все необходимое для работы приложения, включая код, библиотеки и другие зависимости.

5. Что такое volume и зачем он нужен?

Volume (том) в Docker — это механизм для хранения данных вне контейнера. Том монтируется внутрь контейнера и позволяет сохранять данные даже после удаления самого контейнера. Это особенно важно для приложений, которые работают с данными, такими как базы данных или файловые системы.

6. Что такое docker-compose?

Docker Compose — это инструмент для определения и управления несколькими связанными контейнерами в одном проекте. С помощью файла docker-compose.yml можно описать всю архитектуру приложения, включающую несколько сервисов, и запускать их одной командой. Это упрощает работу с многоуровневыми приложениями.

7. В чем разница между Dockerfile и docker-compose?

Dockerfile описывает процесс сборки отдельного образа. В нем указываются команды для установки зависимостей, копирования файлов и настройки среды исполнения.

docker-compose управляет несколькими сервисами (контейнерами), определяя их взаимосвязи, сети, объемы и другие ресурсы. Файл docker-compose.yml позволяет запускать и останавливать весь стек приложений сразу.

8. Какая команда позволяет отправлять различные задания в запущенный докер-контейнер?

Команда docker exec позволяет выполнять команды внутри работающего контейнера. Пример использования:

docker exec -it <container\_name> <command>

Например, для входа в оболочку контейнера можно использовать:

docker exec -it <container\_name> bash

9. С помощью какого инструмента можно сохранить важные данные после аварийного отключения контейнера?

Для сохранения важных данных после аварийного отключения контейнера используются тома (volumes). Данные, сохранённые в томах, остаются доступны даже после остановки или удаления контейнера. Тома можно монтировать в контейнеры с помощью параметров командной строки или через файл docker-compose.yml.

Пример использования тома в docker run:

docker run -v /path/to/data:/data <image\_name>

В этом случае данные будут сохраняться в папку /path/to/data на хост-машине, даже если контейнер будет удален.