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: «Развёртывание СУБД Postgres с использованием средств автоматизации развёртывания и управления приложениями»

Выполнил работу студент группы мИИВТ-241: Тогушов В.А.

подпись, дата

Принял: Короленко В.В.

подпись, дата

Воронеж 2024

Цель лабораторной 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

Индивидуальное задание:

Выполнить все пункты раздела «Учебная задача», оформить отчёт, представить результаты выполнения лабораторной работы к защите.

Для начала зайдем на официальный сайт Docker, посмотрим документацию, зайдем на свой аккаунт на Docker Hub (уже был ранее зарегистрирован). Посмотрим основные команды docker для консоли.

Теперь скачаем и установим Docker Desktop и средство для работы с СУБД DBeaver (Рисунок 1-2).

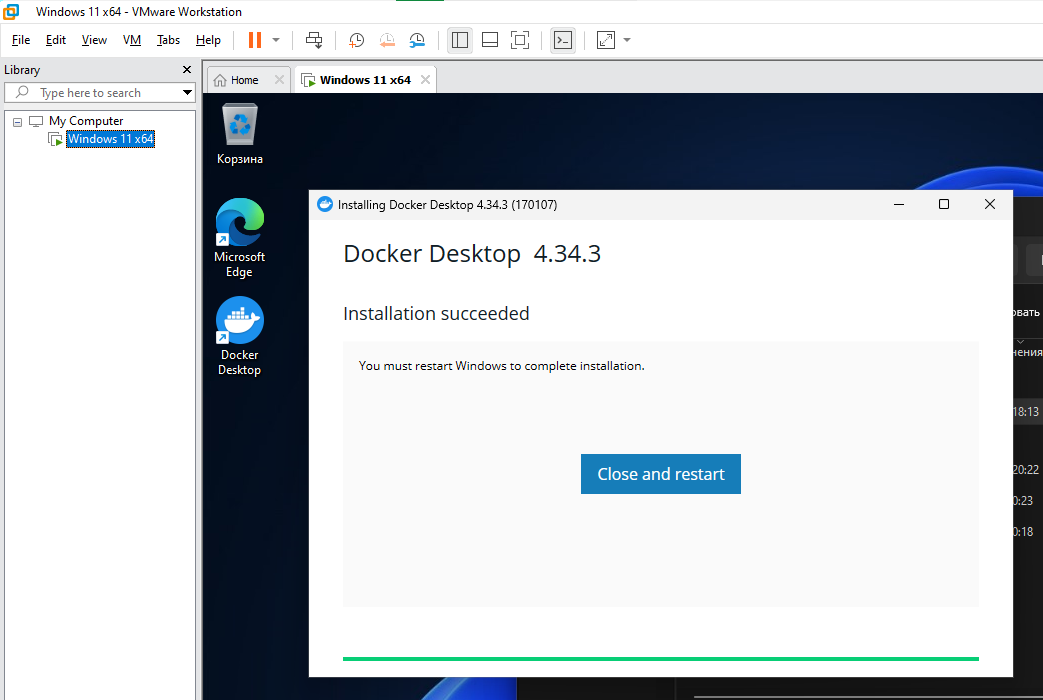


Рисунок 1 – Установка Docker Desktop

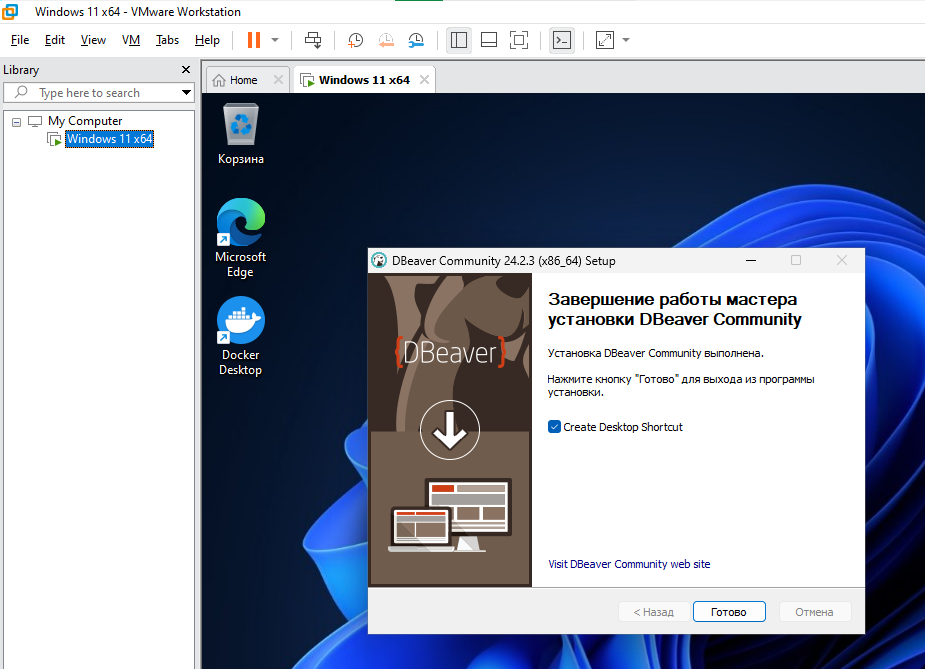


Рисунок 2 – Установка DBeaver

Далее скачаем docker-образ с СУБД postgres с помощью команды в терминале Docker Desktop: docker pull postgres (Рисунок 3).

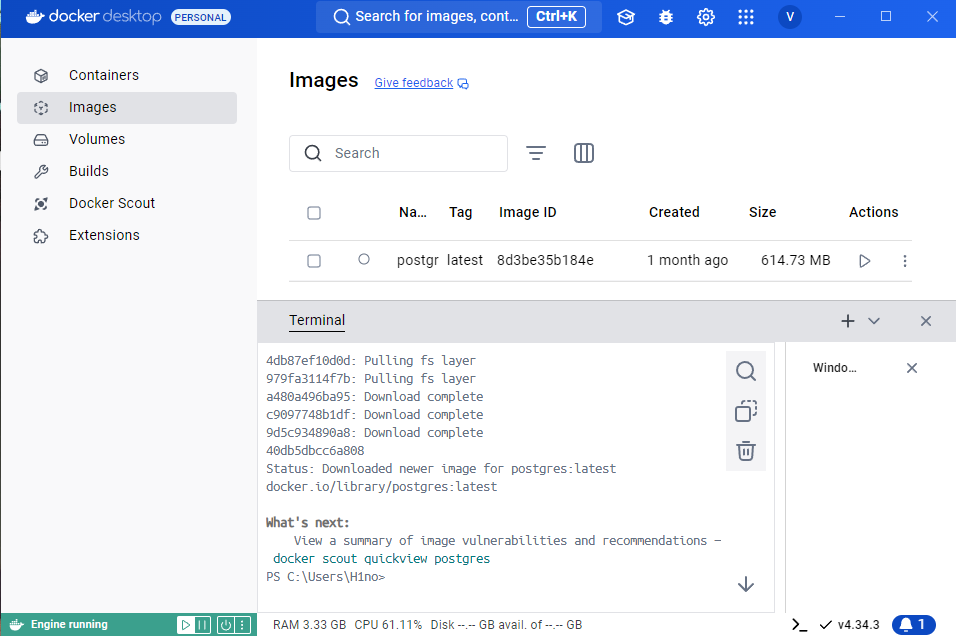


Рисунок 3 – Скачивание docker-образа

Создадим папку, например в «Документах», через терминал Docker Desktop перейдем в нее и создадим Dockerfile. В самом файле будет следующее содержание:

FROM postgres:latest (Указывает, что контейнер создаётся на основе образа PostgreSQL в его последней версии. Она загружает и запускает этот образ как базу)

ENV POSTGRES\_PASSWORD=dbpass (Устанавливает переменную окружения для пароля базы данных. Этот пароль будет назначен учетной записи PostgreSQL, созданной контейнером)

ENV POSTGRES\_USER=dbuser (Задаёт имя пользователя, который будет создан при инициализации базы данных. Контейнер будет использовать эту учетную запись для доступа к базе данных)

ENV POSTGRES\_DB=dbname (Указывает имя базы данных, которая будет создана контейнером при запуске. Она станет основной базой данных для пользователя)

COPY init\_scripts/init.sql /docker-entrypoint-initdb.d/init.sql (Копирует файл init.sql из папки init\_scripts в контейнер, по указанной директории. При первой инициализации базы данных будут созданы таблицы, схемы и т.д.)

Также создадим папку «init\_scripts» и уже в ней файл «init.sql» с таким содержанием:

CREATE TABLE IF NOT EXISTS public.index\_mass ( (Создаёт таблицу «index\_mass» в схеме «public», если такой таблицы ещё нет)

user\_id BIGINT, (Столбец предназначен для хранения уникальных идентификаторов пользователей)

weight BIGINT, (хранит вес пользователя)

height BIGINT (хранит роста пользователя)

);

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

(1, 75, 175),

(2, 60, 182),

(3, 93, 181); (Выполняет вставку данных в таблицу. В скобках перечислены столбцы (user\_id, weight, height), в которые будут вставлены значения).

Теперь создадим образ с помощью команды: docker build -t d\_img:latest ., перейдя в папку, в которой находится Dockerfile (Рисунок 4).

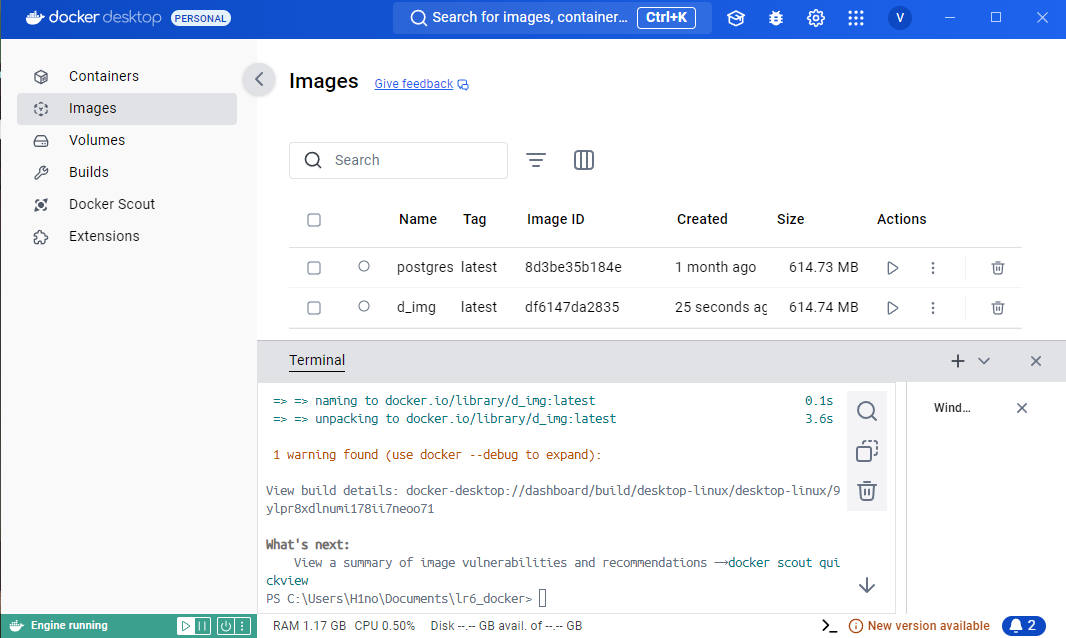


Рисунок 4 – Создание образа docker

Чтобы воспользоваться созданной СУБД, запустим 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.

docker run – используется для создания и запуска контейнера на основе заданного образа.

-d – запускает контейнер в фоновом режиме (detached mode), что позволяет продолжить работу в терминале после его запуска. Без этого флага контейнер запустится в режиме отображения его логов в текущей консоли.

-p 5432:5432 – настраивает проброс порта (port forwarding) из контейнера на хост. Первый 5432 — порт на самом хосте, который будет открыт для доступа, а второй 5432 — порт внутри контейнера, на котором PostgreSQL прослушивает подключения по умолчанию.

--name d\_cont – указывает имя контейнера как d\_cont.

d\_img:latest – указывает образ, на основе которого будет запущен контейнер.

После выполнения команды контейнер запускается и работает в фоновом режиме, в терминале отобразится номер контейнера (Рисунок 5).

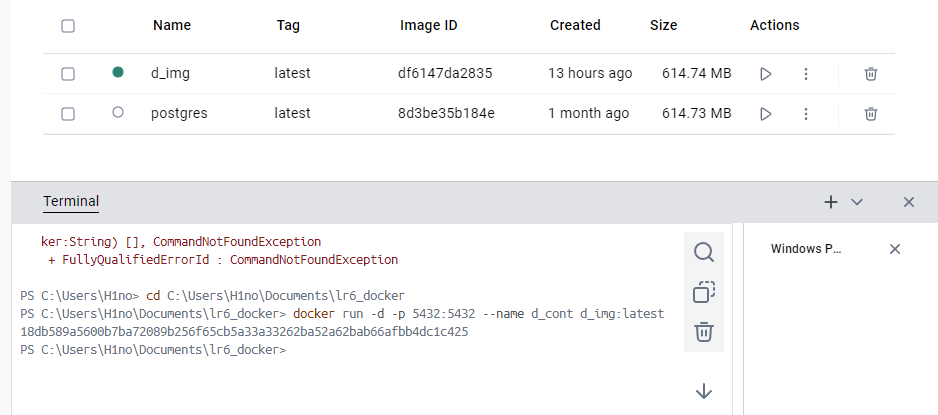


Рисунок 5 – Запуск контейнера d\_cont

Далее проверим развёрнутую СУБД в DBeaver. Выберем создать соединение и укажем тип (PostgreSQL). Подключим базу данных с параметрами, которые были указаны в Dockerfile командами ENV: POSTGRES\_PASSWORD=dbpass (пароль), POSTGRES\_USER=dbuser (имя пользователя), POSTGRES\_DB=dbname (название базы), порт – 5432 (Рисунок 6).

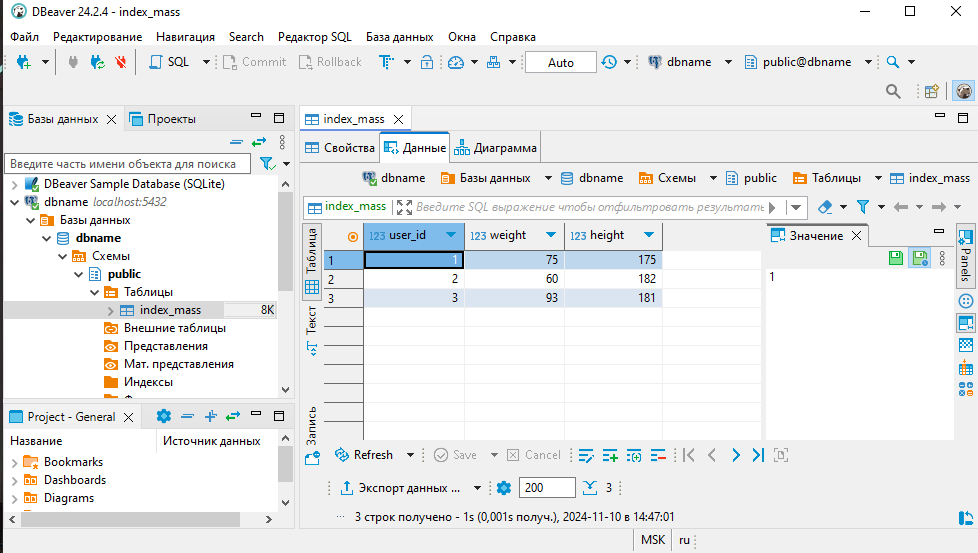


Рисунок 6 – Подключение к базе данных в DBeaver

Напишем специальную команду docker, которая позволит подключаться к работающему контейнеру, запускать интерфейс psql и вносить новые данные быстро: docker exec -it d\_cont psql -U dbuser -d dbname.

docker exec – команда для выполнения команды внутри уже запущенного контейнера.

-it – два флага, которые вместе обеспечивают интерактивный режим прямо в терминале.

d\_cont – имя контейнера, в котором будет выполняться команда psql.

psql – команда для запуска PostgreSQL CLI, которая позволяет взаимодействовать с PostgreSQL через SQL-запросы.

-U dbuser – указывает имя пользователя для подключения к базе данных.

-d dbname – указывает имя базы данных для подключения.

При выполнении команды оказываемся в консоли PostgreSQL внутри контейнера, где может выполнять команды для взаимодействия с базой данных, как если бы это была локальная установка PostgreSQL. Выведем список всех баз данных (Рисунок 7).

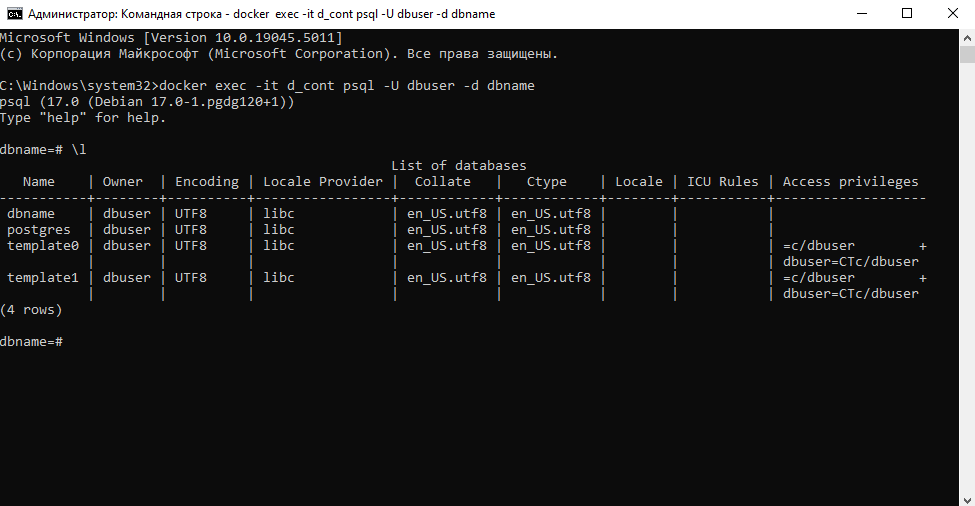


Рисунок 7 – Работа с базой данных из консоли

Выйдем из режима работы с базой данных командой: exit. Остановим контейнер командой: docker stop d\_cont.

Чтобы данные, измененные в процессе работы контейнера, были доступны после удаления контейнера (во вновь созданном контейнере) необходимо создать том (volume).

Для добавления в Dockerfile инструкции для вызова создания контейнера с volume необходимо в Dockerfile добавить строку: VOLUME /d\_data:/var/lib/postgresql/data.

При этом локально будет создан том (volume) в папке «/d\_data», в котором, в свою очередь, будут сохраняться изменения, аналогичные тем, которые происходят в контейнере в папке «/var/lib/postgresql/data».

Теперь удалим созданный контейнер. Создадим другой контейнер с помощью команды: docker run -d -p 5432:5432 --name d\_cont d\_img:latest -v /d\_data:/var/lib/postgresql/data.

docker run – команда для создания и запуска контейнера на основе указанного образа.

-d – запускает контейнер в фоновом режиме, что позволяет продолжить использовать терминал после его запуска.

-p 5432:5432 – настраивает проброс порта, связывая порт 5432 контейнера с портом 5432 на хосте.

--name d\_cont – устанавливает имя контейнера как d\_cont.

d\_img:latest – указывает образ для создания контейнера.

-v /d\_data:/var/lib/postgresql/data – используется для постоянного хранения данных, связывая директорию на хосте с директорией внутри контейнера:

/d\_data – это путь на хост-машине, где будут храниться данные PostgreSQL.

/var/lib/postgresql/data – путь в контейнере, где PostgreSQL по умолчанию сохраняет все данные базы данных.

В контейнере создаётся база данных, которая сохраняет все свои данные в папке /d\_data на хосте. Это позволяет сохранять данные между перезапусками контейнера.

В результате база данных PostgreSQL работает внутри контейнера с доступом к данным, хранящимся вне контейнера, что обеспечивает их постоянное хранение (Рисунок 8).

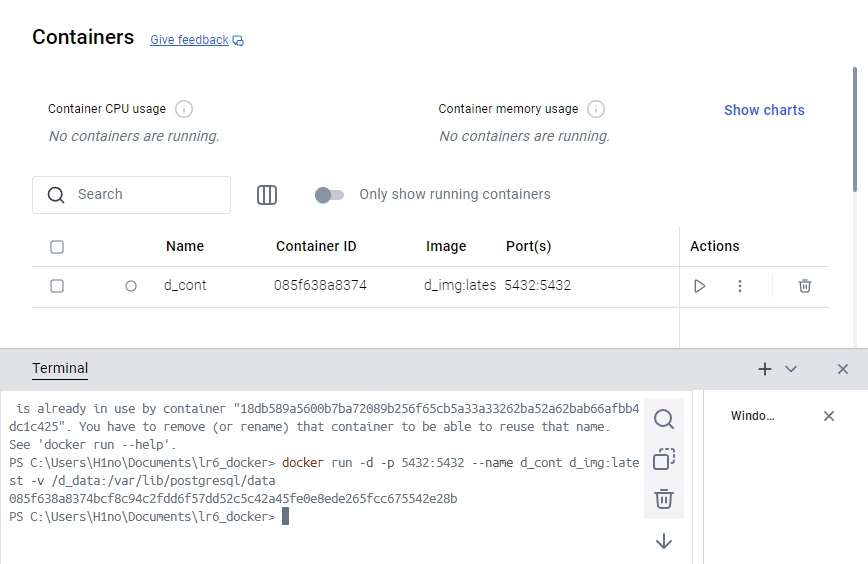


Рисунок 8 – Создание нового контейнера

Создадим текстовый файл «cmd.txt», в котором опишем все команды, которые необходимо использовать для развертывания базы данных Postgres с помощью Dockerfile (создание образа, запуск контейнера, создание тома).

Также в этом файле укажем 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 (Рисунок 9).

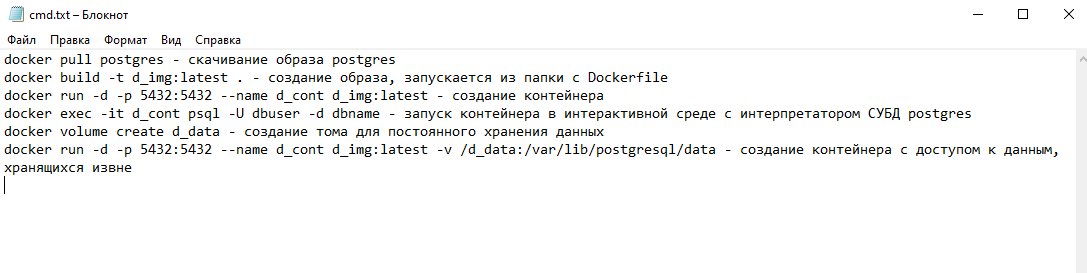


Рисунок 9 – Файл «cmd.txt» с docker командами для создания базы данных

Теперь создадим контейнер с использованием файла docker-compose.yml с таким содержанием:

version: "3.9" (Это версия файла конфигурации Docker Compose. Указывает версию синтаксиса, которая будет использоваться для интерпретации файла.)

services: (Определяет начало раздела сервисов)

 dbpost: (Указывает имя для сервиса. Позволяет легко идентифицировать контейнер, особенно если их несколько.)

  image: postgres:latest (Задает образ контейнера, который будет использоваться)

  environment: (Указывает необходимые переменные окружения для настройки базы данных PostgreSQL)

   POSTGRES\_DB: dbcompose (Создаст базу данных с именем dbcompose)

   POSTGRES\_USER: usrcompose (Устанавливает имя пользователя для подключения к базе данных)

   POSTGRES\_PASSWORD: passcompose (Устанавливает пароль для пользователя, заданного в POSTGRES\_USER)

  volumes: (Настраивает постоянное хранилище данных или файлов между хостом и контейнером)

   - ./init\_scripts/init.sql:/docker-entrypoint-initdb.d/init.sql (Монтирует локальный файл init.sql в контейнер по указанному пути)

  ports: (Указывает правила перенаправления портов между хостом и контейнером)

   - 5432:5432 (Порт 5432 на хост-машине связывается с портом 5432 контейнера, где работает PostgreSQL)

Используем команду: docker-compose up для запуска Docker Compose:

Загружает образ postgres:latest для создания контейнера.

Инициализирует базу данных dbcompose с пользователем usrcompose и паролем passcompose.

Применяет SQL-скрипт из файла init.sql, если он существует, для инициализации схемы базы данных.

Привязывает порт 5432 контейнера к порту 5432 на хосте, обеспечивая доступ к базе данных извне.

Получаем настроенную и работающую базу данных PostgreSQL в контейнере, с которой можно взаимодействовать через порт 5432 на локальной машине (Рисунок 10).

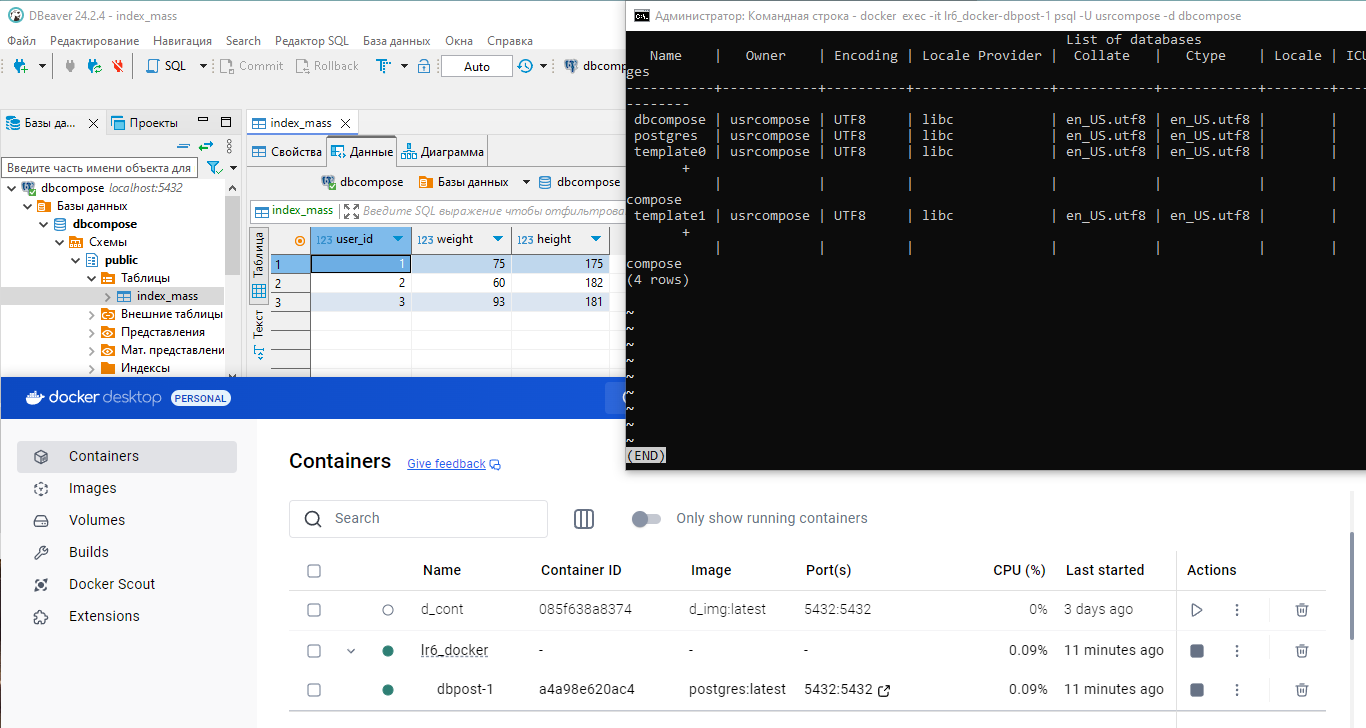


Рисунок 10 – Созданный контейнер с помощью «docker-compose.yml»

Контрольные вопросы:

1) Что такое Docker?

Docker — это программная платформа для разработки, доставки и запуска контейнерных приложений.

2) Зачем нужен Docker?

Docker позволяет отделить приложения от инфраструктуры, а контейнеры не зависят от базовой инфраструктуры, их можно легко перемещать между облачной и локальной инфраструктурами.

3) Что такое docker-образ?

Docker-образ – это шаблон для создания Docker-контейнеров. Представляет собой исполняемый пакет, содержащий всё необходимое для запуска приложения: код, среду выполнения, библиотеки, переменные окружения и файлы конфигурации.

4) Что такое docker-контейнер?

Docker-контейнер — это стандартизированный, изолированный и портативный пакет программного обеспечения, который включает в себя всё необходимое для запуска приложения, включая код, среду выполнения, системные инструменты, библиотеки и настройки.

Контейнеры позволяют упаковать приложение и все его зависимости в единый объект, который может быть запущен на любой системе, поддерживающей Docker, без изменения среды выполнения.

5) Что такое volume и зачем он нужен?

Docker volumes — это каталог или именованное место хранения вне файловой системы контейнера, которое доступно одному или нескольким контейнерам.

6) Что такое docker-compose?

Docker-compose — это надстройка над Docker, приложение, написанное на Python, которое позволяет запускать множество контейнеров одновременно и маршрутизировать потоки данных между ними.

7) В чем разница между dockerfile и docker-compose?

Dockerfile используется для определения среды и конфигурации для одного контейнера Docker. Это текстовый файл, который содержит серию инструкций для создания образа Docker.

Docker Compose используется для определения и запуска многоконтейнерных приложений Docker. Это инструмент, который позволяет определить набор связанных контейнеров Docker, которые работают вместе для запуска приложения.

Таким образом, Dockerfile используется для создания образа Docker, а Docker Compose — для запуска множества связанных контейнеров как одного приложения.

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

Команда docker exec позволяет отправлять различные задания в запущенный Docker-контейнер.

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

Для сохранения важных данных после аварийного отключения контейнера можно использовать инструмент Docker Volumes. Он предоставляет постоянное хранилище для контейнеров, отключая привязку данных к жизненному циклу контейнера. Таким образом, сделанные в контейнерах записи остаются доступными после уничтожения содержавшего их контейнера и могут повторно использоваться в других.