

|  |  |
| --- | --- |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | |
| Институт кибербезопасности и цифровых технологий | |
| Кафедра КБ-3 «Разработка программных решений и системное программирование»   |  | | --- | | **Отчёт по практике 1** | | **по дисциплине** «Разработка безопасного программного обеспечения» |     **Выполнила:**  Студентка 4 курса  Направление 02.03.02  Группа БФБО-01-20  Митрюхина Екатерина Андреевна  **Преподаватель**:  Поспелов П.М.  Москва, 2024 г. | |

1. В данной практике используется образ для Oracle VM VirtualBox, взятый с сайта <https://mirror.yandex.ru/ubuntu-releases/22.04/> под названием [ubuntu-22.04.4-live-server-amd64.iso](https://mirror.yandex.ru/ubuntu-releases/22.04/ubuntu-22.04.4-live-server-amd64.iso) .

Далее был установлен сам докер, прописывались следующие команды в консоли:

sudo apt-get update

sudo apt-get install apt-transport-https ca-certificates curl software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo apt-key fingerprint 0EBFCD88

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu \

xenial \

stable"

sudo apt-get update

sudo apt-get install docker-ce

sudo docker run hello-world

1. Вторым шагом redis находим и устанавливаем его:

sudo docker search redis

sudo docker run -d redis

\*Первая команда `docker search redis` выполняет поиск образов Docker, связанных с Redis, в публичном реестре Docker Hub. Результаты поиска будут содержать информацию о различных образах Redis, доступных для загрузки и использования.  
\*Вторая команда `docker run -d redis` запускает контейнер с образом Redis в фоновом режиме. Опция `-d` указывает на запуск контейнера в фоновом режиме, а название образа `redis` указывает на то, что используется образ Redis. После выполнения этой команды будет создан и запущен контейнер Redis.

С помощью команды sudo docker ps можно открыть список о работающих контейнерах. Пока в нашем списке только контейнер redis.

Далее сделаем проброс портов, в конечном итоге будет результат 0.0.0.0:6379:6379, используем команду sudo docker run -d --name redisHostPort -p 6379:6379 redis:latest

1. Создаем веб сайт со статичной страницей в контейнере.

Первым делом создадим каталог с помощью команды mkdir my\_project.

Далее создадим Dokerfile и web.html. Используется для этого nano редактор.

В web.html мы прописали следующее:

|  |
| --- |
| 1.PNG |

С помощью команды ls проверям, сохранился ли наш файл.

|  |
| --- |
| 2.PNG |

Теперь перейдем к созданию Dokerfile. Будем использовать для начала базовый образ NGINX версии Alpine (настроенный веб сервер в дистрибутиве linux alpine).

В Dokerfile было прописано следующее:

FROM nginx:alpine

COPY . /usr/share/nginx/html

Первая строчка определяет базовый образ. Вторая строчка копирует контент текущей папки во внутрь контейнера (наш web.html).

*\*Сборка Docker Image\**

Для сборки используется команда build. Она принимает несколько параметров. Например, параметр -t позволяет указать имя для и тег для изображения (используется часто как номер версии).

|  |
| --- |
| 3.PNG |

Для просмотра списка изображений используйте команду:

sudo docker images

|  |
| --- |
| 4.PNG |

Запускаем контейнер с пробросом 80 порта.

sudo docker run -d -p 80:80 my-static-website

|  |
| --- |
| 3.PNG |

Результат такой:

|  |
| --- |
| 1.PNG |
| 2.PNG |

Docker Compose

Теперь попробуем запустить 2 контейнера одновременно, используя docker compose.

Сама структура:

docker-compose.yml

|\_ services:

|\_ \_ **flask\_app (1 контейнер):**

|\_ \_ \_ приложение Flask

|\_ \_ \_ \_ Dockerfile

|\_ \_ \_ \_ \_ requirements.txt (библиотеки)

|\_ \_ \_ \_ \_ **templates:**

|\_ \_ \_ index.html

|\_ \_ \_ \_ result.html

|\_ \_ **db (2 контейнер):**

|\_ \_ \_ init.sql (база данных)

Начнем прописывать 1 контейнер – flask\_app.

Первым делом создадим каталог с помощью команды mkdir flask\_app.

Создали 3 файла: app.py, Dockerfile, requirements.txt.

|  |
| --- |
|  |

В app.py прописываем следующее:

|  |
| --- |
|  |

В requirements.txt прописываем следующее:

|  |
| --- |
|  |

А в Dockerfile:

|  |
| --- |
|  |

В этом же каталоге создадим еще один под названием templates. В нем создадим файлы index.html и result.html.

|  |  |
| --- | --- |
|  |  |
| index.html | result.html |

Начнем прописывать 2 контейнер – db.

Первым делом создадим каталог с помощью команды mkdir db.

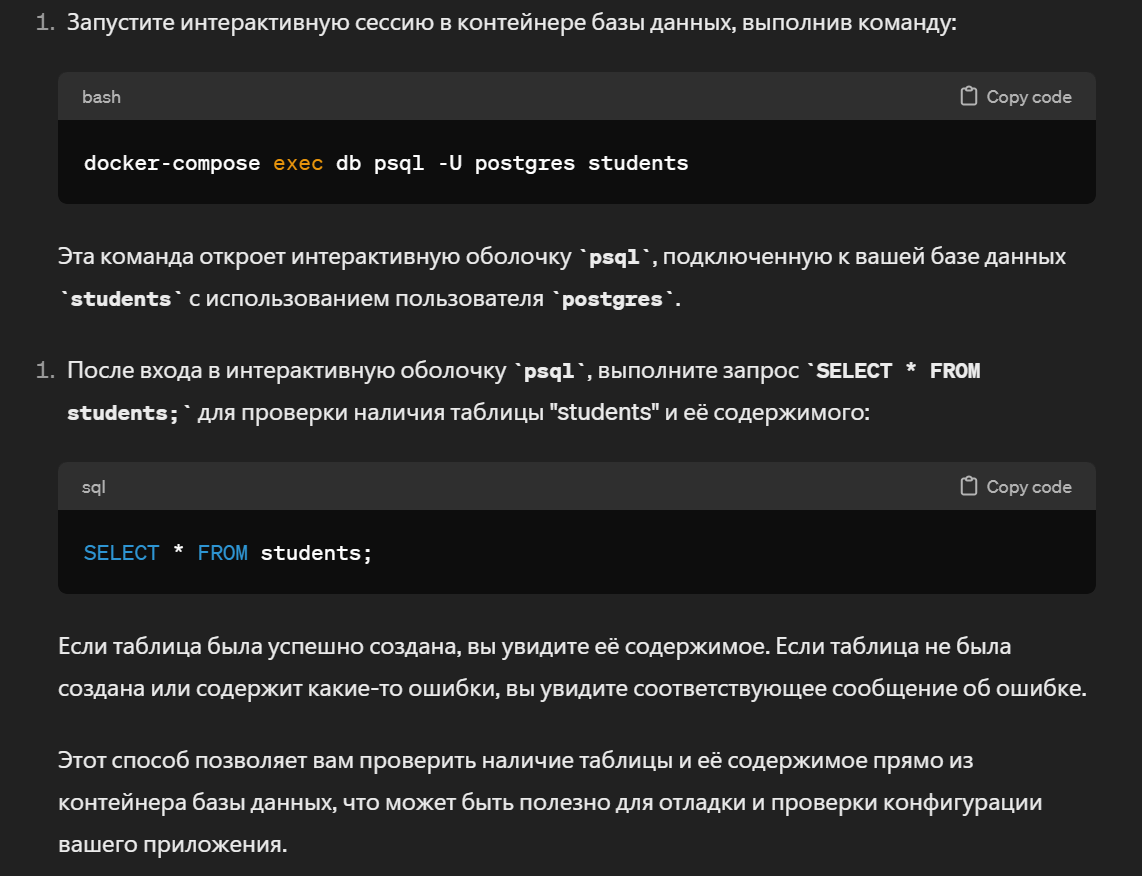
Создадим файл: init.sql.

|  |
| --- |
|  |

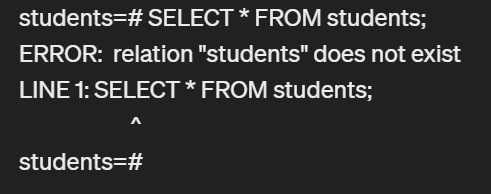
Пропишем в файле init.sql следующее:

|  |
| --- |
|  |

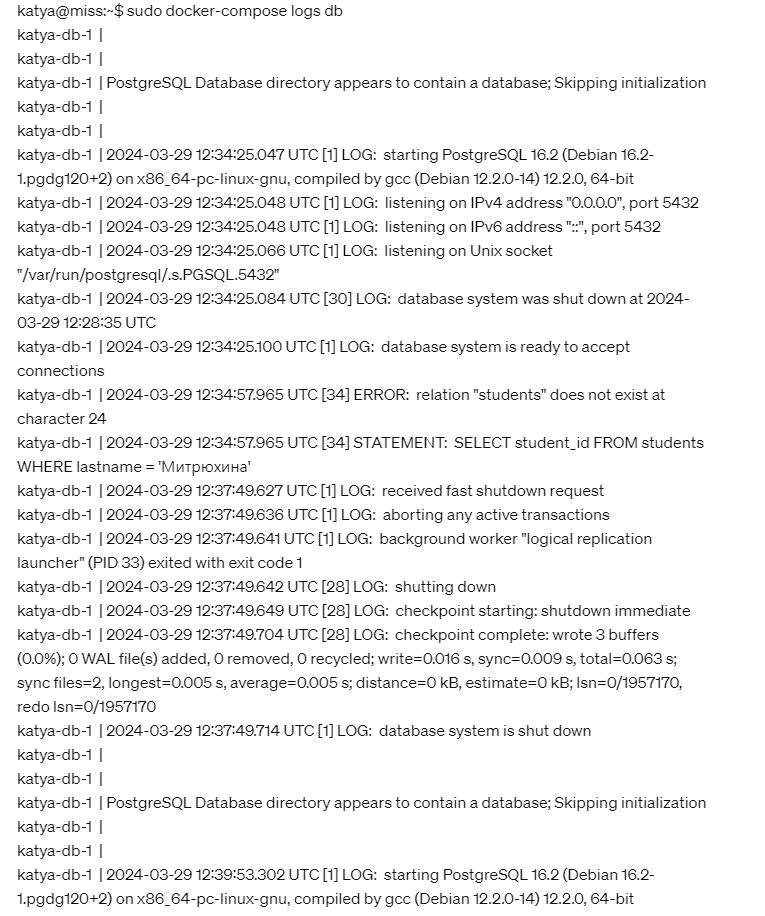
Проверяем, существует ли данная таблица в докере:

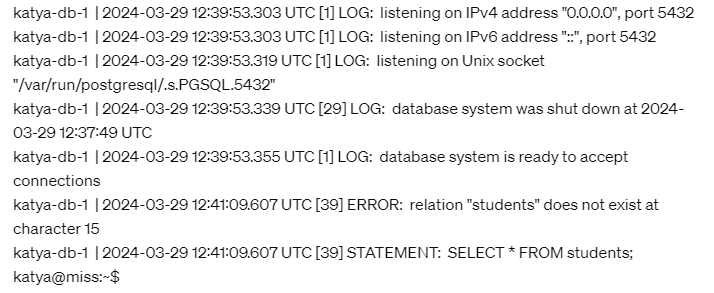


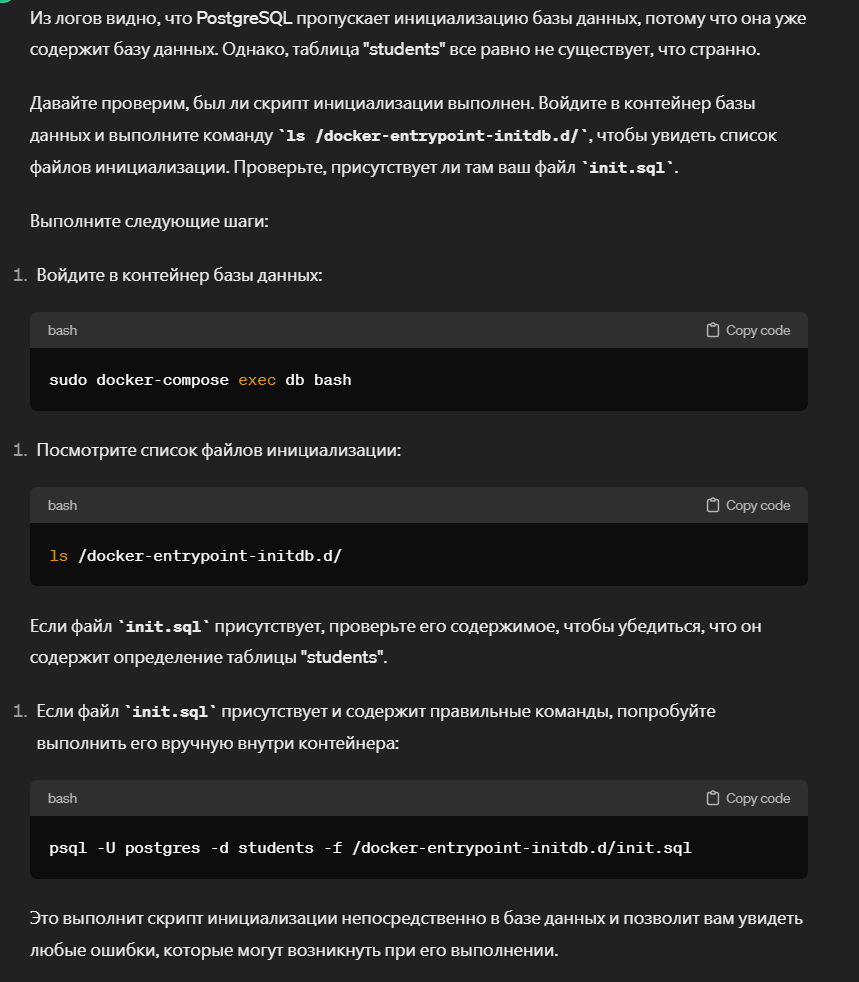
Далее оказалось, что таблица не создана.



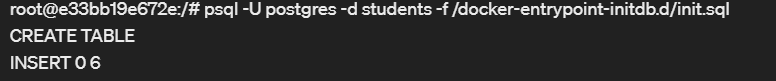
Далее проверяем логи инициализации базы данных с помощью команды: sudo docker-compose logs db.







Результат:



Скрипт инициализации базы данных успешно выполнен. Таблица "students" была создана, и данные были успешно добавлены.

Теперь пропишем файл docker-compose.yml – конфигурацию для всех 3-х контейнеров: html\_page, flask\_app, db.

|  |
| --- |
|  |

Перед запуском запустим каждый контейнер с помощью команды sudo docker-compose up -d (название контейнера).

Полезная команда, если меняли что-то в контейнерах или сам docker compose: sudo docker-compose up - -build.

Теперь, когда все прописано, запустим все 2 контейнера с помощью команды:

sudo docker-compose up

|  |
| --- |
|  |

В результате вводим фамилию, и выводятся фамилия и номер студенческого билета.

|  |
| --- |
|  |
|  |

