**Отчет по лабораторной работе №2**

**Лабораторная работа: Развертывание PostgreSQL с использованием Docker**

**Выполненные действия**

**1. Изучение Docker и регистрация на Docker Hub**

**- Изучены материалы на сайте https://www.docker.com**

**- Изучена документация по Docker (https://docs.docker.com)**

**- Зарегистрирован аккаунт на Docker Hub**

**2. Установка Docker Desktop**

**- Установлен Docker Desktop на Windows 10**

**- Проверена работа Docker через команду `docker --version`**

**3. Установка pgAdmin**

**- Установлен pgAdmin 4 как инструмент для работы с PostgreSQL**

**- Обоснование выбора: pgAdmin является стандартным инструментом для администрирования PostgreSQL, предоставляет графический интерфейс для управления базами данных**

**4. Скачивание образа PostgreSQL**

**- Выполнена команда: `docker pull postgres:latest`**

**5. Создание Dockerfile и init.sql**

**\*\*Dockerfile:\*\***

**```dockerfile**

**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 Dockerfile:**

**- `FROM postgres:latest` - базовый образ для создания контейнера**

**- `ENV POSTGRES\_PASSWORD=dbpass` - установка пароля для БД**

**- `ENV POSTGRES\_USER=dbuser` - установка пользователя БД**

**- `ENV POSTGRES\_DB=dbname` - создание базы данных**

**- `COPY init\_scripts/init.sql /docker-entrypoint-initdb.d/init.sql` - копирование SQL-скрипта для инициализации**

**\*\*init.sql:\*\***

**```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);**

**```**

**Описание init.sql:**

**- `CREATE TABLE IF NOT EXISTS public.index\_mass` - создание таблицы (если не существует)**

**- Определение столбцов: user\_id, weight, height**

**- `INSERT INTO public.index\_mass` - вставка тестовых данных**

**6. Сборка образа**

**- Выполнена команда: `docker build -t d\_img:latest .`**

**7. Запуск контейнера**

**- Выполнена команда: `docker run -d -p 5432:5432 --name d\_cont d\_img:latest`**

**Описание команды:**

**- `docker run` - запуск контейнера**

**- `-d` - запуск в фоновом режиме**

**- `-p 5432:5432` - проброс портов (хост:контейнер)**

**- `--name d\_cont` - имя контейнера**

**- `d\_img:latest` - используемый образ**

**8. Проверка через pgAdmin**

**- Подключение к БД с параметрами:**

**- Host: localhost**

**- Port: 5432**

**- Database: dbname**

**- Username: dbuser**

**- Password: dbpass**

**- Успешно проверено наличие таблицы index\_mass с данными**

**9. Подключение к контейнеру через psql**

**- Выполнена команда: `docker exec -it d\_cont psql -d dbname -U dbuser`**

**- Проверен список БД командой: `\l`**

**10. Создание тома (volume)**

**- Остановлен и удален существующий контейнер**

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

**Описание параметра `-v`:**

**- `-v /d\_data:/var/lib/postgresql/data` - монтирование тома**

**- `/d\_data` - директория на хосте**

**- `/var/lib/postgresql/data` - директория в контейнере**

**11. Создание файла cmd.txt**

**Содержимое файла:**

**```**

**Скачать образ postgres**

**docker pull postgres:latest**

**Собрать образ из Dockerfile**

**docker build -t d\_img:latest .**

**Запустить контейнер с томом**

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

**Подключиться к контейнеру и запустить psql**

**docker exec -it d\_cont psql -d dbname -U dbuser**

**```**

**12. Создание docker-compose.yml**

**Содержимое файла:**

**```yaml**

**version: "3.9"**

**services:**

**dbpost:**

**image: postgres:latest**

**environment:**

**POSTGRES\_DB: dbcompose**

**POSTGRES\_USER: usrcompose**

**POSTGRES\_PASSWORD: passcompose**

**volumes:**

**- ./init\_scripts/init.sql:/docker-entrypoint-initdb.d/init.sql**

**ports:**

**- 5432:5432**

**```**

**Описание docker-compose.yml:**

**- `version: "3.9"` - версия формата файла**

**- `services` - определение сервисов**

**- `dbpost` - имя сервиса**

**- `image: postgres:latest` - используемый образ**

**- `environment` - переменные окружения для настройки БД**

**- `volumes` - монтирование SQL-скрипта для инициализации**

**- `ports` - проброс портов**

### 3. Описание ключевых команд Docker

#### Команда запуска контейнера:

bash

docker run -d -p 5432:5432 --name d\_cont d\_img:latest

* docker run: Запустить новый контейнер.
* -d: Запустить контейнер в фоновом режиме (detached).
* -p 5432:5432: Проброс портов (хост:контейнер). Соединения на порт 5432 вашей машины перенаправляются в контейнер.
* --name d\_cont: Присвоить контейнеру понятное имя.
* d\_img:latest: Имя образа, на основе которого создается контейнер.

#### Команда подключения к PSQL:

bash

docker exec -it d\_cont psql -d dbname -U dbuser

* docker exec: Выполнить команду внутри запущенного контейнера.
* -it: Интерактивный режим с псевдо-TTY.
* d\_cont: Имя контейнера, в котором выполняется команда.
* psql: Запускаемая команда (клиент PostgreSQL).
* -d dbname -U dbuser: Аргументы для psql: подключиться к БД dbname под пользователем dbuser.

#### Команда запуска с томом (Volume):

bash

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

* -v pg\_data:/var/lib/postgresql/data: Смонтировать том с именем pg\_data в директорию внутри контейнера /var/lib/postgresql/data, где PostgreSQL хранит все данные. Это обеспечивает их сохранность после удаления контейнера.

### 4. Инструкция по подключению в pgAdmin

1. Запустите контейнер с PostgreSQL (d\_cont).
2. Откройте pgAdmin и добавьте новый сервер (Register > Server).
3. Во вкладке **Connection** укажите:
   * **Host name/address**: localhost
   * **Port**: 5432
   * **Maintenance database**: dbname (или dbcompose для контейнера из docker-compose)
   * **Username**: dbuser (или usrcompose)
   * **Password**: dbpass (или passcompose)
4. Сохраните настройки. Теперь вы можете просматривать таблицы, выполнять запросы и управлять БД через графический интерфейс pgAdmin.

### 5. Процесс работы docker-compose.yml

При выполнении команды docker-compose up -d в директории с файлом docker-compose.yml происходит следующее:

1. **Парсинг файла**: Docker Compose читает файл и создает в памяти план развертывания.
2. **Пулл образа**: Если образа postgres:latest нет локально, он скачивается из Docker Hub.
3. **Создание тома**: Для монтирования файла init.sql неявно создается том.
4. **Создание контейнера**: Запускается контейнер с именем [директория]\_dbpost\_1.
5. **Настройка окружения**: В контейнер передаются переменные POSTGRES\_\*.
6. **Монтирование скрипта**: Файл ./init\_scripts/init.sql монтируется в контейнер.
7. **Проброс порта**: Порт контейнера связывается с портом на хосте.
8. **Инициализация БД**: При первом запуске PostgreSQL выполняет скрипт init.sql, создавая таблицу и данные.

**Результат**: Развернута полностью настроенная СУБД PostgreSQL, готовая к использованию по адресу localhost:5432.

### Контрольные вопросы и ответы

#### 1. Что такое Docker?

**Docker** — это платформа с открытым исходным кодом для автоматизации развертывания приложений в контейнерах. Контейнеры позволяют упаковать приложение со всеми его зависимостями и библиотеками в стандартизированную единицу программного обеспечения, что обеспечивает быструю и надежную работу приложения в разных средах.

#### 2. Зачем нужен Docker?

Docker нужен для:

* Изоляции приложений и их зависимостей
* Обеспечения консистентности среды выполнения на разных этапах (разработка, тестирование, production)
* Быстрого развертывания и масштабирования приложений
* Эффективного использования ресурсов системы
* Упрощения процесса CI/CD (непрерывной интеграции и доставки)

#### 3. Что такое docker-образ?

**Docker-образ** — это шаблон только для чтения, используемый для создания контейнеров. Образ включает в себя все необходимое для запуска приложения: код, среду выполнения, библиотеки, переменные окружения и конфигурационные файлы. Образы создаются из Dockerfile и могут храниться в реестрах (например, Docker Hub).

#### 4. Что такое docker-контейнер?

**Docker-контейнер** — это запущенный экземпляр docker-образа. Контейнер включает в себя приложение и все его зависимости, но разделяет ядро операционной системы с другими контейнерами. Контейнеры изолированы друг от друга и от хостовой системы, что обеспечивает безопасность и предсказуемость работы приложений.

#### 5. Что такое volume и зачем он нужен?

**Volume** — это механизм для сохранения данных, генерируемых и используемых контейнерами Docker. Volumes нужны для:

* Сохранения данных после удаления контейнера
* Обмена данными между контейнерами
* Резервного копирования и миграции данных
* Повышения производительности (особенно в Docker Desktop для Windows/Mac)

#### 6. Что такое docker-compose?

**Docker Compose** — это инструмент для определения и запуска многоконтейнерных приложений Docker. С помощью YAML-файла (docker-compose.yml) можно описать все сервисы, сети и тома, необходимые для работы приложения, и запустить их одной командой.

#### 7. В чем разница между dockerfile и docker-compose?

**Dockerfile** используется для создания пользовательского docker-образа, в то время как **docker-compose** используется для орchestration (оркестровки) нескольких контейнеров:

* Dockerfile описывает, как построить образ
* Docker-compose описывает, как запустить многоконтейнерное приложение

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

Команда docker exec позволяет выполнять команды внутри запущенного контейнера. Например:

bash

docker exec -it имя\_контейнера команда

Флаг -it обеспечивает интерактивный режим работы с TTY.

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

Для сохранения важных данных после аварийного отключения контейнера используются **тома (volumes)**. Тома обеспечивают постоянное хранение данных независимо от жизненного цикла контейнеров. Данные в томах сохраняются даже после удаления всех контейнеров, которые их использовали.

**Репозиторий Git**

**Все материалы работы размещены в публичном репозитории GitHub:**

**https://github.com/GraveGit/lab6**