

Департамент образования и науки города Москвы
Государственное автономное образовательное учреждение высшего
образования города Москвы
«Московский городской педагогический университет»
Институт цифрового образования
Департамент информатики, управления и технологий

ДИСЦИПЛИНА:

Интеграция и развертывание программного обеспечения с помощью
контейнеров

Лабораторная работа 2.1

Создание Dockerfile и сборка образа

Выполнил(а): Шведова С.С., группа: АДЭУ-211

Преподаватель: Босенко Т.М.

Москва

2025

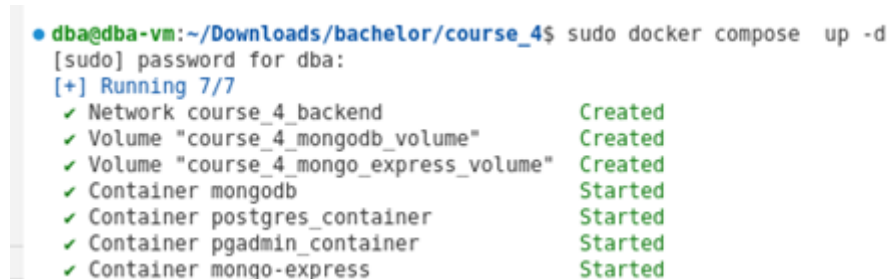
Цель работы: научиться создавать Dockerfile и собирать образы Docker для приложений.

Задачи:

1. Создать Dockerfile для указанного приложения.
2. Собрать образ Docker с использованием созданного Dockerfile.
3. Запустить контейнер из собранного образа и проверить его работоспособность.
4. Выполнить индивидуальное задание.

Вариант 13. Создайте Dockerfile для приложения на Java, которое использует библиотеку JDBC для подключения к базе данных PostgreSQL и выполнения простого запроса.

Для того, чтобы было подключение к базе данных PostgreSQL, надо его настроить. Сначала надо запустить контейнеры на основе настроек, определенных в файле docker-compose.yml (рисунок 1).



```
● dba@dba-vm:~/Downloads/bachelor/course_4$ sudo docker compose up -d
[sudo] password for dba:
[+] Running 7/7
 ✓ Network course_4_backend          Created
 ✓ Volume "course_4_mongodb_volume"  Created
 ✓ Volume "course_4_mongo_express_volume" Created
 ✓ Container mongodb                 Started
 ✓ Container postgres_container      Started
 ✓ Container pgadmin_container       Started
 ✓ Container mongo-express           Started
```

Рисунок 1. Запуск контейнеров

Затем с помощью команды `sudo docker inspect id_container` надо узнать IP address сервера PgAdmin и после этого уже перейти по нему (рисунок 2). Так как индивидуальное задание требует выполнение простого запроса, то нужно создать таблицу и заполнить ее значениями.

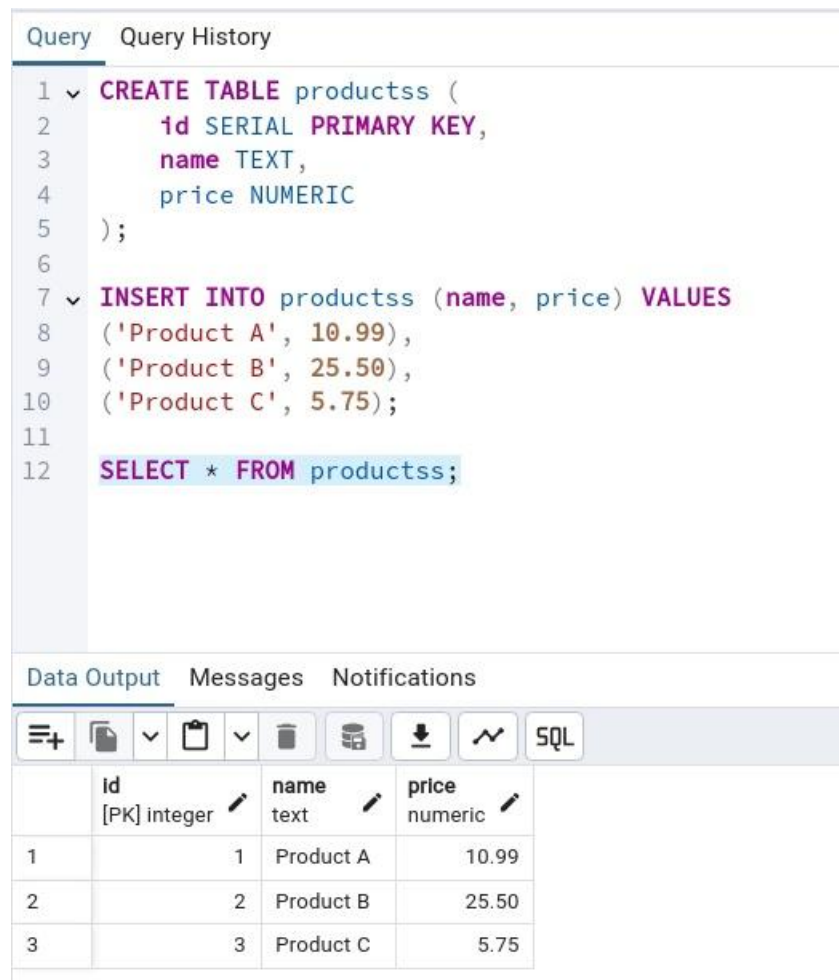


Рисунок 2. Подключение к PostgreSQL

После этого надо в папке проекта my-java-app создать докерфайл и исходный код программы, который по заданию должен быть написан на языке Java (рисунок 3).

```
dba@dba-vm:~$ cd my-java-app  
dba@dba-vm:~/my-java-app$ touch app.java  
dba@dba-vm:~/my-java-app$ nano app.java  
dba@dba-vm:~/my-java-app$ touch Dockerfile  
dba@dba-vm:~/my-java-app$ nano Dockerfile  
dba@dba-vm:~/my-java-app$
```

Рисунок 3. Создание файлов

Докерфайл выглядит следующим образом (рисунок 4).

1. **FROM openjdk:11-jdk-slim.** Указывает базовый образ, на основе которого будет строиться новый образ. Она используется облегченный образ OpenJDK версии 11 с установленным JDK (Java Development Kit). Он будет служить средой для выполнения Java-приложения.

2. **WORKDIR /app**: Эта команда устанавливает рабочую директорию внутри контейнера. Все последующие команды будут выполняться из этой директории.

3. **COPY ./app**: Эта команда копирует все файлы из текущей директории хоста (где находится Dockerfile) в директорию /app внутри контейнера.

4. **RUN javac -cp postgresql-42.7.5.jar App.java**: Эта команда выполняет компиляцию файла App.java, используя Java-компилятор javac. Параметр -cp postgresql-42.7.5.jar указывает класс-путь (classpath) для компилятора, так что он может находить классы из библиотеки Postgres, необходимой для приложения.

5. **CMD ["java", "-cp", ".:postgresql-42.7.5.jar", "App"]** определяет команду, которая будет выполнена при запуске контейнера. Она запускает Java-программу, задавая класс-путь (classpath), чтобы включить текущую директорию (.) и файл библиотеки postgresql-42.7.5.jar. App — это имя класса, который будет запущен.



```
Dockerfile
1 FROM openjdk:11-jdk-slim
2 WORKDIR /app
3 COPY . /app
4 RUN javac -cp postgresql-42.7.5.jar App.java
5
6 CMD ["java", "-cp", ".:postgresql-42.7.5.jar", "App"]
```

Рисунок 4. Dockerfile

Чтобы включить файл библиотеки postgresql-42.7.5.jar, надо его установить и переместить в рабочий каталог. На рисунке 5 показаны файлы в папке проекта, где библиотека установлена.



Рисунок 5. Папка проекта

Затем нужно создать исходный код программы, написанный на Java (рисунок 6).

Сначала происходит импорт необходимых классов, то есть код импортирует классы из пакета `java.sql`, которые необходимы для работы с базой данных, включая `Connection`, `DriverManager`, `PreparedStatement`, `ResultSet`, и `SQLException`. Затем в классе `App` определяются параметры подключения и создается вывод простого запроса. В данном случае выбирает все столбцы из таблицы `products` и, используя цикл `while`, программа проходит по каждой строке результата, и выводит значение столбца `price`.

```
J App.java
1  import java.sql.Connection;
2  import java.sql.DriverManager;
3  import java.sql.PreparedStatement;
4  import java.sql.ResultSet;
5  import java.sql.SQLException;
6  import java.sql.Statement;
7
8  public class App {
9      public static void main(String[] args) {
10         String url = "jdbc:postgresql://172.19.0.3:5432/postgres";
11         String user = "postgres";
12         String password = "changeme";
13
14         try (Connection conn = DriverManager.getConnection(url, user, password);
15             PreparedStatement preparedStatement = conn.prepareStatement("SELECT * FROM productss");
16             ResultSet rs = preparedStatement.executeQuery()) {
17
18             while (rs.next()) {
19                 System.out.println("Data: " + rs.getString("price"));
20             }
21
22         } catch (SQLException e) {
23             e.printStackTrace();
24         }
25     }
26 }
```

Рисунок 6. Код программы

После этого надо собрать образ (рисунок 7).

```
dba@dba-vm:~/my-java-app$ sudo docker build -t my-java-app .
[+] Building 3.6s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 585B
=> [internal] load metadata for docker.io/library/openjdk:11-jdk-slim
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/5] FROM docker.io/library/openjdk:11-jdk-slim@sha256:868a4f2151d38ba6a09870cec584346a5edc8e9b71fde275eb2e0625273e2fd8
=> [internal] load build context
=> => transferring context: 1.1kB
=> CACHED [2/5] WORKDIR /app
=> [3/5] COPY . /app
=> [4/5] RUN javac -cp postgresql-42.7.5.jar App.java
=> [5/5] WORKDIR /app
=> exporting to image
=> => exporting layers
=> => writing image sha256:f7bb9dfe2e5c28b275fdb3ba6e47ae35afb87ace351bbbed3d7f2467487c7eee
=> => naming to docker.io/library/my-java-app
```

Рисунок 7. Сборка образа

Теперь можно запустить приложение. Как можно увидеть на рисунке 8, оно действительно вывело значения столбца price.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
dba@dba-vm:~/my-java-app$ sudo docker run --network="host" my-java-app
Data: 10.99
Data: 25.50
Data: 5.75
```

Рисунок 8. Запуск приложения

На рисунке 9 показано удаление докера.

```
dba@dba-vm:~$ sudo docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS
PORTS
7a7d68775d82   java-postres-app "java Main"             3 days ago    Exited (0) 3 days ago
a457dbff5f72   myapp         "python -m uvicorn s..." 3 months ago  Exited (255) 5 weeks ag
o 0.0.0.0:8000->8000/tcp, [::]:8000->8000/tcp myapp
fc7c96c36fe7   postgres     "docker-entrypoint.s..." 3 months ago  Exited (255) 5 weeks ag
o 0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp mydb
dba@dba-vm:~$ sudo docker rm 7a7d68775d82
7a7d68775d82
dba@dba-vm:~$ sudo docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS
PORTS
a457dbff5f72   myapp         "python -m uvicorn s..." 3 months ago  Exited (255) 5 weeks ago 0.0.
0.0:8000->8000/tcp, [::]:8000->8000/tcp myapp
fc7c96c36fe7   postgres     "docker-entrypoint.s..." 3 months ago  Exited (255) 5 weeks ago 0.0.
0.0:5432->5432/tcp, [::]:5432->5432/tcp mydb
dba@dba-vm:~$
```

Рисунок 9. Удаление докера

Выводы

1. Был создан Dockerfile для приложения на Java;
2. Собран образ Docker с использованием созданного Dockerfile;
3. Был запущен контейнер из собранного образа и была проверена его работоспособность.

Контрольные вопросы

1. Что такое Dockerfile и для чего он используется?

Dockerfile — это текстовый файл, содержащий инструкции для сборки исходного кода. Он позволяет автоматизировать процесс построения контейнеров, используя базовый образ.

2. Какие основные инструкции используются в Dockerfile?

- **FROM.** Указывает базовый образ, на основе которого нужно создать новый. Чаще всего FROM используется для образов с операционной системой и предустановленными компонентами.
- **RUN.** Указывает, какие команды необходимо выполнить внутри контейнера во время сборки образа. Так можно установить зависимости или обновить пакеты до нужной версии.
- **COPY и ADD.** Копирует файлы из локальной файловой системы в контейнер. Чаще всего копируют исходный код приложения.
- **WORKDIR.** Устанавливает рабочую директорию для последующих инструкций. Так можно последовательно работать с файлами в разных директориях.
- **CMD.** Определяет аргументы по умолчанию при запуске контейнера.
- **ENTRYPOINT.** Задает команду, которая будет выполнена при запуске контейнера.

3. Как выполняется сборка образа Docker с использованием Dockerfile?

Сборка образа Docker с использованием Dockerfile выполняется с помощью команды `docker build`.

4. Как запустить контейнер из собранного образа?

Чтобы запустить контейнер из собранного образа, нужно выполнить команду `docker run`

5. Каковы преимущества использования Dockerfile для создания образов Docker?

- Позволяет создавать одинаковое окружение для работы программы, независимо от машины, которая собирает образ.
- Оптимизирует распределение ресурсов по контейнерам.

- Управляет жизненным циклом приложения — запускает, останавливает, масштабирует и обновляет контейнеры