Практическая работа 4

«Создание многоконтейнерного приложения с использованием Docker и docker-compose»

Цели задания:

Приобретения практического опыта и навыков в области использования контейнеров Docker, написанием Dockerfile и использованием docker-compose для оркестрации множества контейнеров.

Описание задания:

Научиться создавать многоконтейнерное приложение с помощью Docker и управлять им с помощью docker-compose. Кроме того, познакомится с настройкой и взаимодействием таких компонентов как веб-приложение, база данных PostgreSQL и управляющим приложением pgAdmin. Решение будет состоять из нескольких контейнеров, включающих в себя:

- контейнер с веб-приложением, реализованным на языке программирования Python с использованием фреймворка Flask.
- контейнер с сервером базы данных PostgreSQL, который будет использоваться веб-приложением для доступа к необходимым данным.
- контейнер с управляющим приложением pgAdmin, обеспечивающим возможность управления базой данных через веб-интерфейс.

Шаги:

Шаг 1: Установка Docker.

Если у вас еще не установлен Docker то необходимо обратиться к официальной документации размещенной по адресу https://docs.docker.com/get-docker/ чтобы выполнить инсталляцию пакета подходящей для вашей операционной системы.

Рекомендуется устанавливать и работать с Docker в ранее созданной оболочке WSL которая была создана в прошлой практической работе. Инструкцию для установки Docker Engine на Ubuntu можно взять здесь: https://docs.docker.com/engine/install/ubuntu/

Шаг 2: Написание **Dockerfile** для будущего веб-приложения.

Для написания веб приложения создайте новую ветку в вашем репозитории и соответствующий каталог с лабораторной работой. Перейдите в этот каталог и создайте новый файл с именем `app.py` со следующим содержанием:

```
from flask import Flask, render_template
import os
import psycopg2

app = Flask(__name__)

def get_db_connection():
      host = os.getenv('DATABASE_HOST', 'localhost')  # Получаем значение
      переменной окружения DATABASE_HOST
      dbname = os.getenv('POSTGRES_DB', 'mydatabase') # Получаем значение
      переменной окружения POSTGRES_DB
      user = os.getenv('POSTGRES_USER', 'myuser')     # Получаем значение
      переменной окружения POSTGRES_USER
      password = os.getenv('POSTGRES_PASSWORD', 'mypassword') # Получаем
      значение переменной окружения POSTGRES_PASSWORD

      conn = psycopg2.connect(
            dbname = dbname,
            user = user,
            password = password,
            host = host
      )
      return conn

@app.route('/')
def hello_world():
      conn = get_db_connection()
      cursor = conn.cursor()
      cursor.execute("SELECT * FROM messages")
      messages = cursor.fetchall()
      cursor.close()
      conn.close()
      return render_template('index.html', messages=messages)

if __name__ == '__main__':
      app.run(debug=True, host='0.0.0.0')
```

Это наше веб-приложение, которое будет подключаться к базе данных PostgreSQL и извлекать сообщения из таблицы "`messages`". Вы можете использовать свой вариант приложения на любом другом языке программирования, с которым вы знакомы и умеете работать.

Обратите внимание: в примере выше предполагается, что у вас есть база данных "`mydatabase`", пользователь "`myuser`" с паролем "`mypassword`". При необходимости измените их в соответствии с вашей разработкой.

Теперь, необходимо создать скрипт для инициализации нашей будущей базы данных PostgreSQL с описанием ее структуры. Он пригодится нам на следующих шагах. Для этого, создайте скрипт `init_db.sql` и добавьте в него следующие строки:

```
-- Создание таблицы с данными
CREATE TABLE messages (
    id SERIAL PRIMARY KEY,
    message TEXT
);

-- Вставка данных
INSERT INTO messages (message) VALUES
    ('Привет :)'),
    ('Вас приветствует web-приложение написанное на языке программирования
Python с использованием фреймворка Flask.'),
    ('Если вы видите данное сообщение, вам удалось выполнить сборку
многоконтейнерного приложения и ваше приложение работает корректно!');
```

Этот скрипт создает таблицу "`messages`" с одним полем "`message`" типа TEXT. Затем он вставляет три строки в эту таблицу с разными сообщениями.

Для отображения данных на нашей будущей веб-страницы необходимо использовать шаблон `index.html`. Для этого нужно создать `index.html` в папке `templates` вашего проекта. Этот файл будет содержать HTML и использовать переменную "`messages`", которая передается в функции `render_template('index.html', messages=messages)`.

Ниже представлен пример, как может выглядеть файл `index.html`:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Messages</title>
</head>
<body>
    <h1>Messages</h1>
        {% for message in messages %}
            <p>{{ message[1] }}</p>
        {% endfor %}
</body>
</html>
```

В заключение нашего текущего шага, настало время создать и сам **Dockerfile**:

```
# Используем базовый образ Python
FROM python:3.9

# Устанавливаем psycopg2 для подключения к PostgreSQL
RUN pip install psycopg2-binary Flask

# Копируем все файлы в контейнер
COPY . /app
WORKDIR /app
```

```
# Определяем команду для запуска приложения
CMD ["python", "app.py"]
```

В примере выше используется официальный образ Python и устанавливаем необходимые зависимости для нашего приложения (Flask и psycopg2-binary). Затем мы копируем исходный код нашего приложения в контейнер и устанавливаем рабочую директорию. Наконец, мы определяем команду для запуска нашего приложения.

Шаг 3: Конфигурация docker-compose.

Создайте файл docker-compose.yml, в котором определите необходимые сервисы для веб-приложения, для PostgreSQL и для pgAdmin. Для каждого сервиса укажите необходимые параметры, такие как порты, переменные окружения и т.д.

Пример оформления представлен ниже:

```
version: '3.3'

services:
  web-client:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "5000:5000"
    depends_on:
      - service-db
    environment:
      - DATABASE_HOST=service-db

  service-db:
    image: postgres:latest
    environment:
      POSTGRES_DB: mydatabase
      POSTGRES_USER: myuser
      POSTGRES_PASSWORD: mypassword
    ports:
      - "5432:5432"
    volumes:
      - ./init_db.sql:/docker-entrypoint-initdb.d/init_db.sql
      - db-data:/var/lib/postgresql/data
    restart: unless-stopped

  pgadmin:
    image: dpage/pgadmin4:latest
    restart: always
    ports:
      - "8080:80"
    environment:
      PGADMIN_DEFAULT_EMAIL: admin@example.com
      PGADMIN_DEFAULT_PASSWORD: admin
    depends_on:
      - service-db
    volumes:
```

```
      - pgadmin-data:/var/lib/pgadmin

volumes:
  db-data:
  pgadmin-data:
```

Шаг 4: Создание и запуск контейнеров.

С помощью команды `docker-compose build` выполните сборку образа контейнеров.

Запустить контейнеры с помощью команды `docker-compose up`.

Шаг 5: Настройка pgAdmin.

После запуска контейнера pgAdmin зайти в веб-интерфейс pgAdmin и настройте подключение к базе данных PostgreSQL используя ранее заданные параметры для пользователя и его пароля. Убедиться, что pgAdmin успешно подключается к PostgreSQL и позволяет управлять базой данных через веб-интерфейс.

Шаг 6: Тестирование и отладка.

Откройте веб-браузер и перейдите по адресу `http://localhost:5000`, чтобы убедиться, что веб-приложение запущено и работает.

С помощью pgAdmin подключитесь к базе данных и отредактируйте текст в сообщениях, обновите страницу с веб-приложением и убедитесь, что вы получаете новые сообщения на странице.

Для подтверждения полученных результатов приложите скриншоты в итоговый отчет.

Шаг 7: Сохранение отчетов о проделанной работе.

Создайте директорию reports и разместите в ней все необходимые требуемые файлы. Зафиксируйте изменения с использованием `git add` и `git commit`. Отправьте изменения на GitLab с использованием `git push`.

Шаг 8: Создание Pull Request.

Создайте Pull Request на слияние вашей ветки в master ветку. В запросе укажите пул выполненных работ по заданию. В качестве Rewiewer укажите преподавателя.

Задание считается выполненным, если преподаватель выполнил слияние вашей ветки в основную, иначе смотрите комментарий в вашем запросе с описанием недочетов, исправляйте ошибки и снова выставляйте работу на проверку.

Вопросы и поддержка:

Если у вас возникают трудности или у вас есть вопросы, не стесняйтесь обращаться за помощью к преподавателю или на форум группы.

Удачи в выполнении задания!