Задача 2.4: Создание и запуск контейнера с веб-приложением в Docker

Подготовка

Для примера загрузим готовое web-приложение на языке Python, с использованием фреймворка flask и web-сервера Gunicorn.

https://github.com/render-examples/flask-hello-world

Создадим папку для нашего приложения и склонируем все необходимые файлы

```
mkdir webapp
```

```
git clone https://github.com/render-examples/flask-hello-world .
```

```
ruslan@ruslan-Z690-UD:~/IT-2023/task_2/task_2.4/webapp$ ll
итого 24
drwxrwxr-x 2 ruslan ruslan 4096 ноя  2 22:42 ./
drwxrwxr-x 3 ruslan ruslan 4096 ноя  2 22:41 ../
-rw-rw-r-- 1 ruslan ruslan  109 мая 29  2019 app.py
-rw-rw-r-- 1 ruslan ruslan 1074 мая 29  2019 LICENSE
-rw-rw-r-- 1 ruslan ruslan  347 мая 29  2019 README.md
-rw-rw-r-- 1 ruslan ruslan   15 мая 29  2019 requirements.txt
ruslan@ruslan-Z690-UD:~/IT-2023/task_2/task_2.4/webapp$
```

Инициируем и настраиваем Dockerfile в директории нашего проекта

1. Создадим Dockerfile и откроем его для редактирования:
   `nano Dockerfile`
2. Для Python приложения указываем базовый образ:
   `FROM python:3.9`
3. Определяем рабочую директорию внутри контейнера:
   `WORKDIR /webapp`
4. Копируем файл с необходимыми компонентами в созданную директорию:
   `COPY requirements.txt /webapp`
5. Так как приложение на языке Python, то обновляем менеджер pip и устанавливаем с его помощью необходимые компоненты:
   `RUN pip install --upgrade pip && python -m pip install -r requirements.txt`
6. Указываем Docker порт для прослушивания (tcp):
   `EXPOSE 5000`
7. Копируем наше приложение внутрь контейнера в созданную выше директорию:
   `COPY . /webapp`
8. Определяем команду с аргументами, которую нужно выполнить когда контейнер будет запущен:
   `CMD [ "gunicorn", "--bind", "0.0.0.0:5000", "app:app"]`

Запускаем проект

1. Собираем новый образ приложения с тегом "webapp":

```
sudo docker build -t webapp
```

```
uslan@ruslan-Z690-UD:~/IT-2023/task_2/task_2.4/webapp$ sudo docker build -t webapp .
sudo] пароль для ruslan:
+] Building 1.6s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 811B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/python:3.9
=> [1/5] FROM docker.io/library/python:3.9@sha256:6c5b37cdb4abbf77c4facdeaf4a7400061380df0451cedd3ee2b44cf680b6807
=> [internal] load build context
=> => transferring context: 927B
=> CACHED [2/5] WORKDIR /webapp
=> CACHED [3/5] COPY requirements.txt /webapp
=> CACHED [4/5] RUN pip install --upgrade pip && python -m pip install -r requirements.txt
=> [5/5] COPY . /webapp
=> exporting to image
=> => exporting layers
=> => writing image sha256:3f6c00a96cfd2fad74d8aee1c1f68c1a4c32bf2c9f0433fe3aff90c4e3b37eff
=> => naming to docker.io/library/webapp
uslan@ruslan-Z690-UD:~/IT-2023/task_2/task_2.4/webapp$
```

2. Проверяем - выводим список образов docker:

```
sudo docker images
```

```
ruslan@ruslan-Z690-UD:~/IT-2023/task_2/task_2.4/webapp$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
webapp       latest    3f6c00a96cfd   21 seconds ago   1.02GB
ruslan@ruslan-Z690-UD:~/IT-2023/task_2/task_2.4/webapp$
```

3. Запускаем докер контейнер в режиме "detach" (в фоновом режиме):

```
sudo docker run -d -p 5000:5000 webapp
```

4. Проверяем статус контейнера:

```
sudo docker ps
```

```
ruslan@ruslan-Z690-UD:~/IT-2023/task_2/task_2.4/webapp$ sudo docker run -d -p 5000:5000 webapp
3c2ed13ae57543623162491e3ef3666d3fe05d506c9840ce8d09715ee7f08111
ruslan@ruslan-Z690-UD:~/IT-2023/task_2/task_2.4/webapp$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                                       NAMES
3c2ed13ae575   webapp    "gunicorn --bind 0.0…"   8 seconds ago   Up 8 seconds   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   angry_haslett
ruslan@ruslan-Z690-UD:~/IT-2023/task_2/task_2.4/webapp$
```

5. Убедимся, что ваше веб-приложение доступно в браузере на порту 5000: