

Homework_Lesson22_Docker_3

Цель: получить практический опыт написания Dockerfile, развертывания приложений с использованием Docker-compose/

Задание 1: Создание Dockerfile для приложения веб-сервера. Вам необходимо написать Dockerfile для создания контейнера с приложением веб-сервера на основе образа Ubuntu 20.04. Приложение должно быть запущено на порту 8080 и должно отдавать статические файлы из каталога /app/static.

Шаги, которые необходимо выполнить:

1. Создайте новый файл Dockerfile в пустой директории на вашем локальном компьютере.
2. Напишите инструкцию FROM, которая указывает базовый образ Ubuntu 20.04.
3. Установите необходимые зависимости с помощью инструкции RUN. Установите пакеты nginx и curl, а также создайте каталог /app/static.
4. Скопируйте файл конфигурации nginx из вашего локального каталога внутрь контейнера с помощью инструкции COPY.
5. Скопируйте статические файлы из каталога /app/static на вашем локальном компьютере внутрь контейнера с помощью инструкции COPY.
6. Используйте инструкцию EXPOSE для открытия порта 8080.
7. Используйте инструкцию CMD для запуска команды nginx с указанием пути к файлу конфигурации, который вы скопировали на шаге 4.
8. Сохраните файл Dockerfile и соберите образ с помощью команды docker build.
9. Запустите контейнер из образа с помощью команды docker run и проверьте, что веб-сервер отдает статические файлы из каталога /app/static на порту 8080.

Задание 2 – развертывание приложения с помощью Docker-compose

Шаги, которые необходимо выполнить:

1. Создайте новый файл docker-compose.yml в пустой директории на вашем локальном компьютере.
2. Напишите инструкцию version в версии 3.
3. Определите сервис для базы данных PostgreSQL. Назовите его "db". Используйте образ postgres:latest, задайте переменные окружения POSTGRES_USER, POSTGRES_PASSWORD и POSTGRES_DB для установки пользовательского имени, пароля и имени базы данных соответственно.
4. Определите сервис для веб-сервера на основе образа NGINX. Назовите его "web". Используйте образ nginx:latest. Определите порт, на котором должен работать сервер, с помощью инструкции ports. Задайте путь к файлам конфигурации NGINX внутри контейнера, используя инструкцию volumes.
- 5.* Определите ссылку на сервис базы данных в сервисе веб-сервера. Используйте инструкцию links.
6. Сохраните файл docker-compose.yml и запустите приложение с помощью команды docker-compose up.
7. Проверьте, что приложение работает, перейдя в браузере на localhost:80.

Выполнение первого задания:

1. Установка Docker Desktop и интеграция на WSL.

1) Скачиваем Docker Desktop и следуем инструкциям по установке <https://docs.docker.com/desktop/features/wsl/#download>.

После успешной установки выдаст окно о завершении и попросит перезагрузить Windows:

Docker Desktop 4.37.1

Installation succeeded

You must log out of Windows to complete installation.

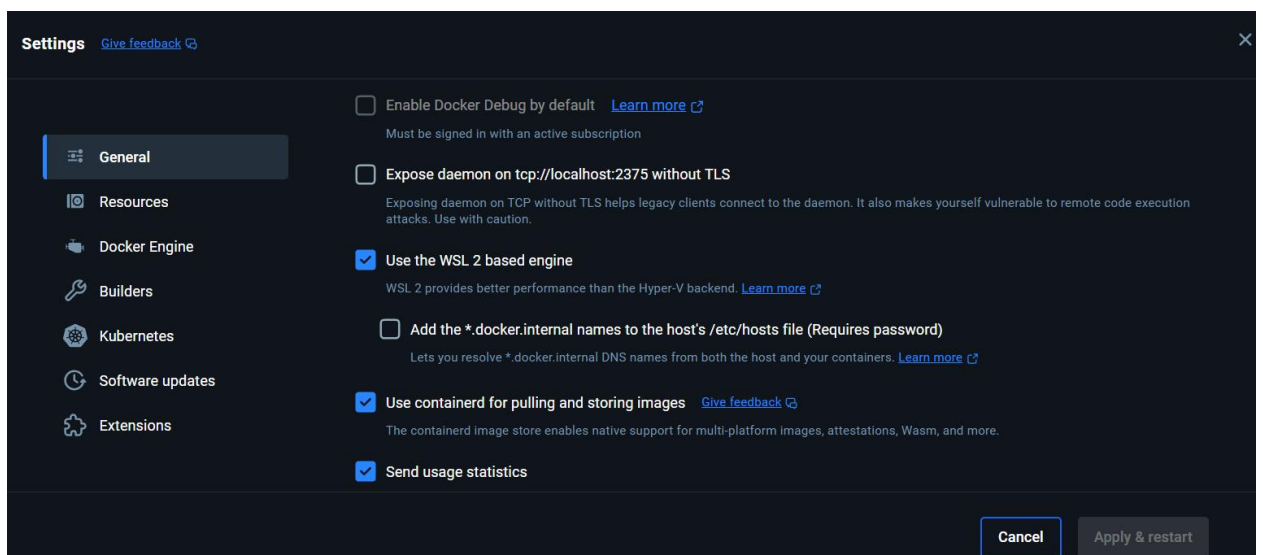
Close and log out

Успешная установка Docker Desktop

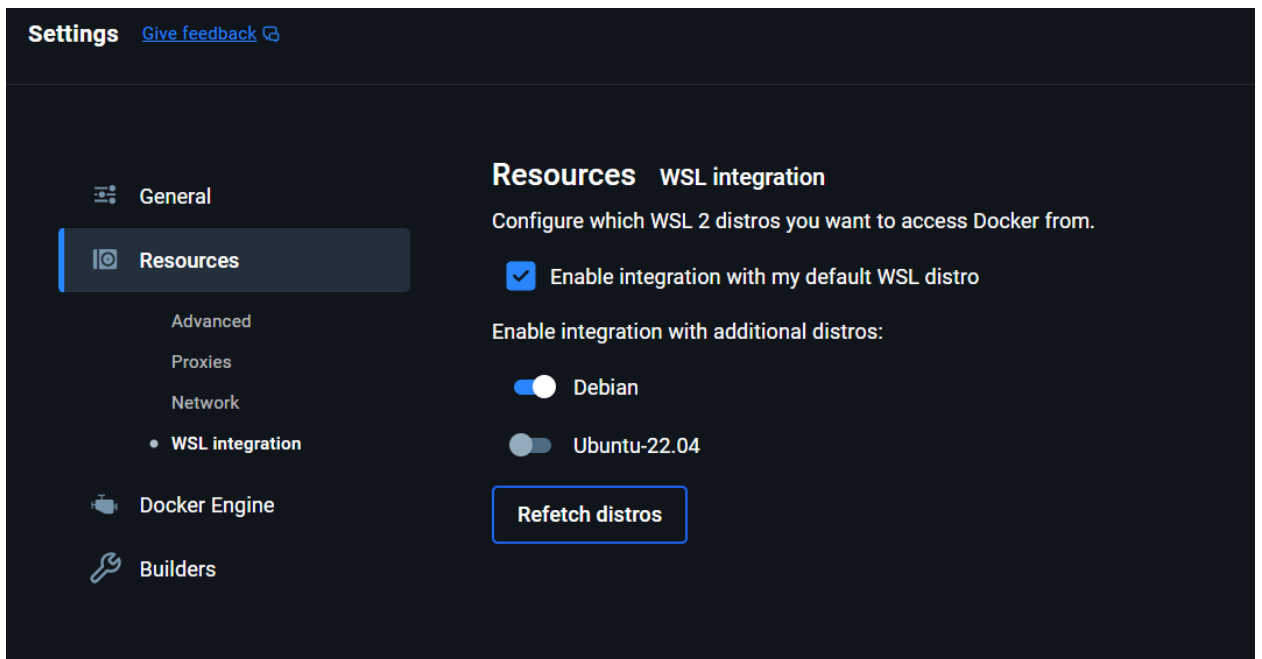
2) После перезагрузки Windows, нужно запустить Docker Desktop и перейти в настройки:



3) Поставить флажок “Use the WSL2 based engine”:



4) Выбрать из установленных дистрибутивов WSL2, которые необходимо включить интеграцию Docker, перейдя в раздел Resources -> WSL integration:



5) После, нужно убедиться, что Docker установлен. Для этого открываем в терминале дистрибутив WSL и вводим следующие команды:

```
$ docker --version
```

6) Проверим правильность работы установки, выполнив встроенный образ Docker:

```
$ docker run hello-world
```

```
makarov@DESKTOP-UG6J7T7:~$ docker --version
Docker version 27.5.1, build 9f9e405
makarov@DESKTOP-UG6J7T7:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
e6590344b1a5: Download complete
Digest: sha256:d715f14f9eca81473d9112df50457893aa4d099adeb4729f679006bf5ea12407
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
makarov@DESKTOP-UG6J7T7:~$ |
```

2. Переходим к заданию:

```
1) $ mkdir /docker/docker_ubuntu_nginx  \\ Создаем директорию, где будем создавать докер файл и создавать необходимые файлы для копирования их в контейнер
$ touch Dockerfile
$ nano Dockerfile
```

Создаем инструкции для Dockerfile. Добавляем следующие строки в Dockerfile:

```
# Указываем базовый образ по заданию
FROM ubuntu:20.04

# Обновляем пакетный менеджер и устанавливаем необходимые зависимости
RUN apt-get update && \
    apt-get install -y nginx curl && \
    rm -rf /var/lib/apt/lists/* && \
    mkdir -p /app/static

# Копируем файл конфигурации nginx в контейнер
ADD tms.conf /etc/nginx/sites-available/

# Создание симлинка нашего сайта tms.by
RUN ln -s /etc/nginx/sites-available/tms.conf /etc/nginx/sites-enabled

# Копируем файлы из локального каталога /app/static
COPY static/ /app/static/

# Открываем порт 8080
EXPOSE 8080

# Команда для запуска nginx с указанным файлом конфигурации
CMD ["nginx", "-g", "daemon off;"]
```

2) Сохраняем Dockerfile. После, в этой же директории, где у нас создан Dockerfile, создаем все необходимые файлы конфигурации и прочие файлы, которые будут переноситься в контейнер. Создаем файл конфигурации NGINX tms.conf:

```
$ touch tms.conf
$ nano tms.conf

#tms.by
server {
    listen 8080;
    server_name tms_man.by www.tms_man.by;

    root /app/static;
    index tms_man tms_man.html;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

3) Создадим директорию ~/static и в неё же поместим созданную нами html-файл:

```
$ mkdir /static
$ cd ./static
$ touch tms_man.html
$ nano tms_man.html
```

Содержимое файла tms_man.html:

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>NGINX</title>
</head>
<body>
  <h1>ФИО: Макаров Александр</h1>
  <h2>Тема урока: Docker. Часть 3</h2>
  <h3>Docker</h3>
</body>
</html>
```

Итого у нас получается вот такое содержимое директории:

```
root@DESKTOP-UG6J7T7: /home/makarov/docker/docker_ubuntu_nginx# tree
.
├── Dockerfile
├── static
│   └── tms_man.html
└── tms.conf

2 directories, 3 files
```

Когда все необходимые файлы подготовлены переходим к сборке нашего образа (IMAGE).

4) Запускаем сборку нашего образа (image):

```
$ docker build -t my-ubuntu-nginx .
```

```
makarov@DESKTOP-UG6J7T7:~/docker/docker_ubuntu_nginx$ docker build -t my-ubuntu-nginx .
[+] Building 1.3s (10/10) FINISHED
=> [internal] load build definition from Dockerfile                                docker:default 0.0s
=> => transferring dockerfile: 925B                                              0.0s
=> [internal] load metadata for docker.io/library/ubuntu:20.04                  0.7s
=> [internal] load .dockerignore                                                 0.0s
=> => transferring context: 2B                                                    0.0s
=> [1/5] FROM docker.io/library/ubuntu:20.04@sha256:8e5c4f0285ecbb4ead070431d29b576a530d3166df73ec44affc1cd27555141b 0.0s
=> => resolve docker.io/library/ubuntu:20.04@sha256:8e5c4f0285ecbb4ead070431d29b576a530d3166df73ec44affc1cd27555141b 0.0s
=> [internal] load build context                                                 0.0s
=> => transferring context: 95B                                                  0.0s
=> CACHED [2/5] RUN apt-get update && apt-get install -y nginx curl && rm -rf /var/lib/apt/lists/* && mkdir -p /app/s 0.0s
=> [3/5] ADD tms.conf /etc/nginx/sites-available/                               0.0s
=> [4/5] RUN ln -s /etc/nginx/sites-available/tms.conf /etc/nginx/sites-enabled 0.2s
=> [5/5] COPY static/ /app/static/                                              0.0s
=> => exporting to image                                                         0.2s
=> => exporting layers                                                            0.1s
=> => exporting manifest sha256:af8184c7686d7a57679b1f51948934346dab23ea504c82ba29e8f08cb51d1a6b          0.0s
=> => exporting config sha256:aeb0264dfbecce5c2ccddc8b1168368355bb3e65347e6c8280c0d6bd8a1b33cf          0.0s
=> => exporting attestation manifest sha256:758c8884e3fe40e4510330512dc6f9729f3dd3e84061a2f2cf0f452a70f459fc 0.0s
=> => exporting manifest list sha256:519ab2d31e301bcab74fd2968ebabe2da88df379d8fe11f7f26fb7576498f33d       0.0s
=> => naming to docker.io/library/my-ubuntu-nginx:latest                       0.0s
=> => unpacking to docker.io/library/my-ubuntu-nginx:latest                    0.0s
```

\$ docker images

\\ команда позволяет вывести все наши образы

```
makarov@DESKTOP-UG6J7T7:~/docker/docker_ubuntu_nginx$ docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
my-ubuntu-nginx     latest          519ab2d31e30   49 minutes ago 217MB
test                latest          3bc6b94c566f   2 hours ago    217MB
my-jenkins          latest          9026b9d4a16a   23 hours ago   1.1GB
docker              dind            3ab005a2e487   12 days ago    517MB
hello-world         latest          d715f14f9eca   13 days ago    20.4kB
```

5) Запускаем первый наш контейнер:

\$ docker run -d -p 8080:8080 my-ubuntu-nginx \\запуск контейнера

```
makarov@DESKTOP-UG6J7T7:~/docker/docker_ubuntu_nginx$ docker run -d -p 8080:8080 my-ubuntu-nginx
baa4cd988140ccb7e5b9e0ed31742d2b6d47b6a8b92301197f672343a8714020
```

Проверка работы нашего сайта с контейнера. Заходим в сам контейнер:

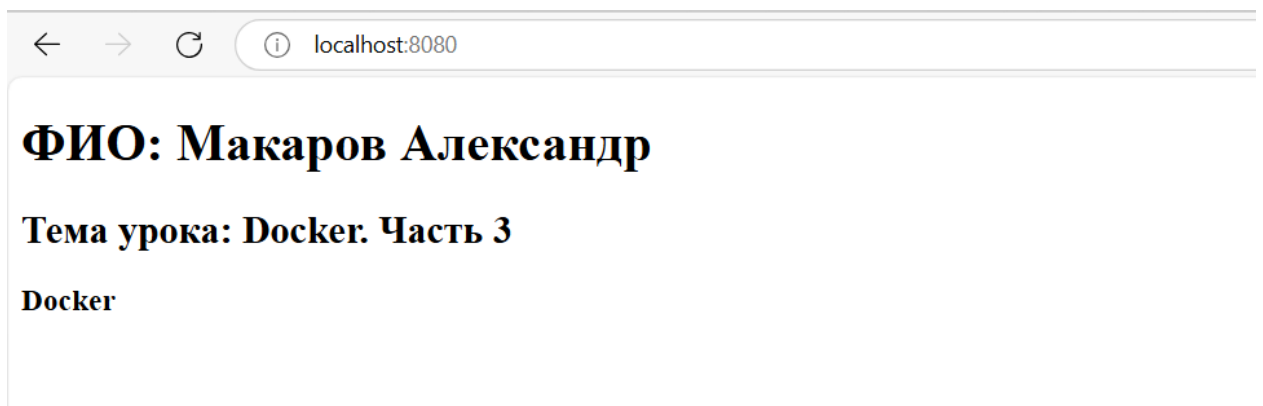
\$ docker exec -it baa bash

/# curl localhost:8080

```
makarov@DESKTOP-UG6J7T7:~/docker/docker_ubuntu_nginx$ docker exec -it baa bash
root@baa4cd988140:/# curl localhost:8080
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>NGINX</title>
</head>
<body>
  <h1>ФИО: Макаров Александр</h1>
  <h2>Тема урока: Docker. Часть 3</h2>
  <h3>Docker</h3>
</body>
</html>
root@baa4cd988140:/# |
```

Мы подключились к нашему контейнеру и вызвали curl

Также наш сайт доступен извне:



После всех манипуляций, можно удалить наш контейнер:

\$ docker stop baa4cd988140

\\ останавливаем наш контейнер

\$ docker rm baa4cd988140

\\ после остановки -> удаляем его

\$ docker ps -a \\\ позволяет нам просматривать запущенные контейнеры

```
makarov@DESKTOP-UG6J7T7:~/docker/docker_ubuntu_nginx$ docker stop baa4cd988140
baa4cd988140
makarov@DESKTOP-UG6J7T7:~/docker/docker_ubuntu_nginx$ docker rm baa4cd988140
baa4cd988140
makarov@DESKTOP-UG6J7T7:~/docker/docker_ubuntu_nginx$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
5465c9a4094a   test          "nginx -g 'daemon of..." 2 hours ago    Created              wonderful_engelbart
57f6a53ad201   hello-world   "/hello"                26 hours ago   Exited (0)    26 hours ago   priceless_shamir
makarov@DESKTOP-UG6J7T7:~/docker/docker_ubuntu_nginx$ |
```

Здесь видим, что в списке нашего контейнера baa4cd988140 нету, он удален

Выполнение второго задания:

```
$ sudo apt install curl softwareproperties-common ca-certificates apt-transport-https -y \\  
установка 4 необходимых пакетов для Docker  
$ sudo curl -f -s -S -L https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add - \\  
импортирование GPG ключей для верификации подписей ПО  
$ add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian $(lsb_release  
-cs) stable" \\  
добавление репозитория Docker для Debian  
$ sudo apt-get update -y \\  
обновление индексов пакетов  
  
$ sudo apt-get install docker-ce docker-ce-cli -y \\  
установка Docker
```