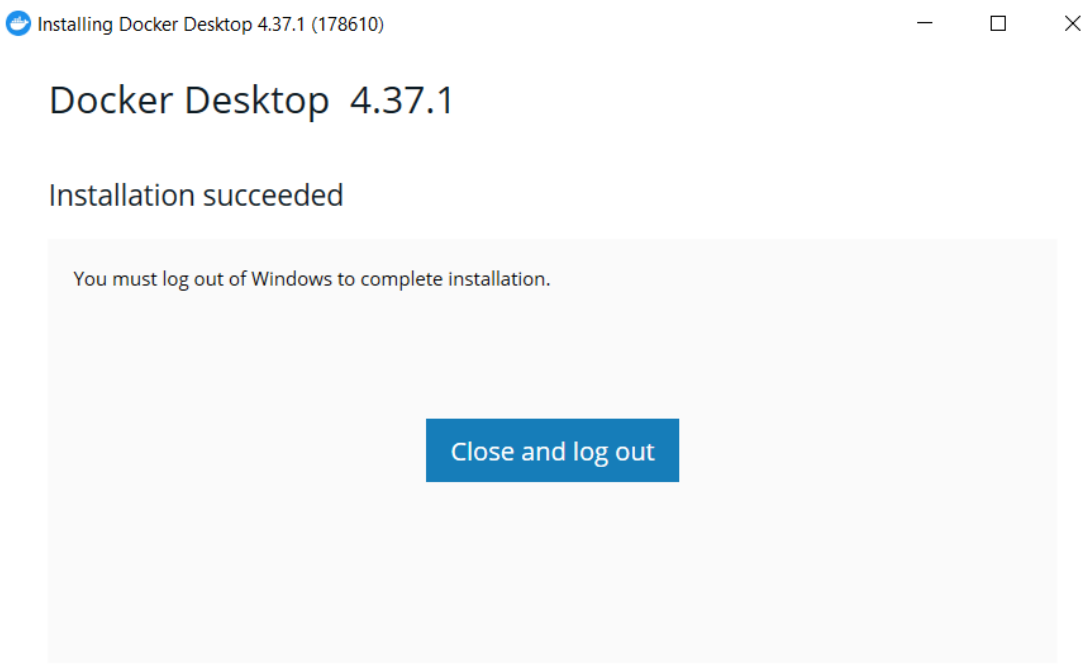
Homework_Lesson22_Docker_3

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 указаниемпути к файлу конфигурации, который вы скопировали на шаге 4.
8. Сохраните файл Dockerfile и соберите образ с помощью команды dockerbuild.
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.

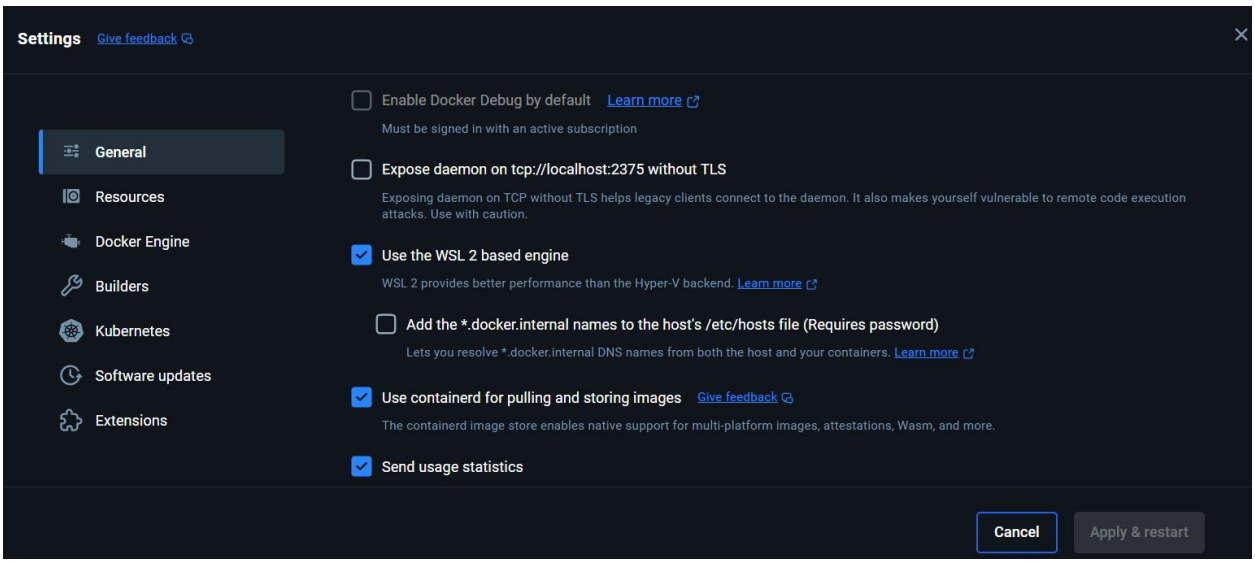
1) Скачиваем Docker Desktop и следуюм инструкциям по установке https://docs.docker.com/desktop/features/wsl/#download.
После успешной установки выдаст окно о завершении и попросит перезагрузить Windows:



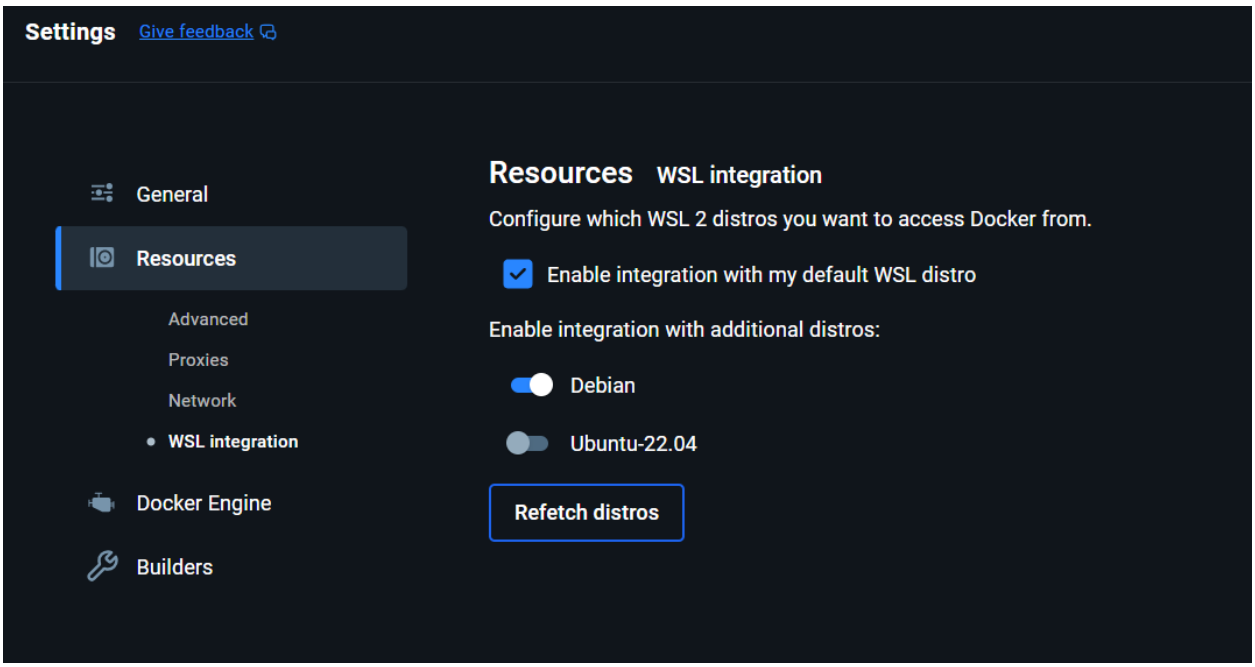
Успешная установка Docker Desktop

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 integretion:



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 переносится 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;
   }
}
```

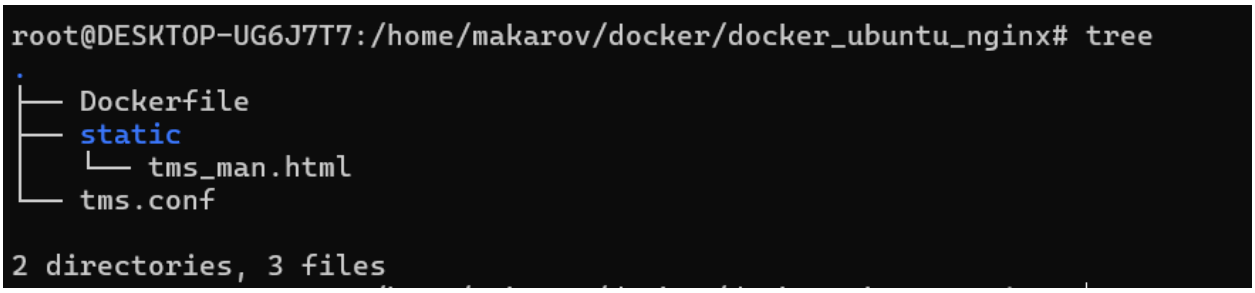
3) Создадим директорию ~/static и в неё же поместим созданную нами html-файл:

$ 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>
```

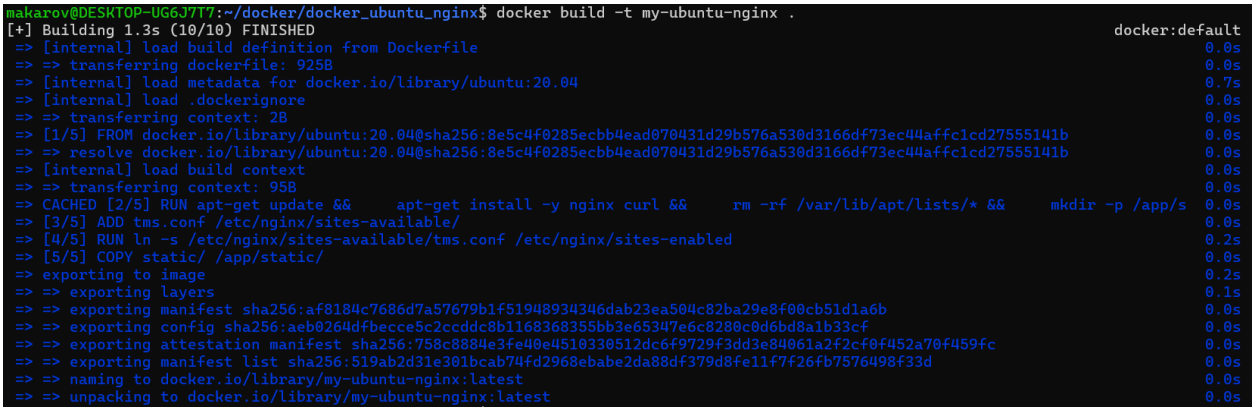
Итого у нас получается вот такое содержимое директории:



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

4) Запускаем сборку нашего образа (image):

$ docker build -t my-ubuntu-nginx .



$ 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
```

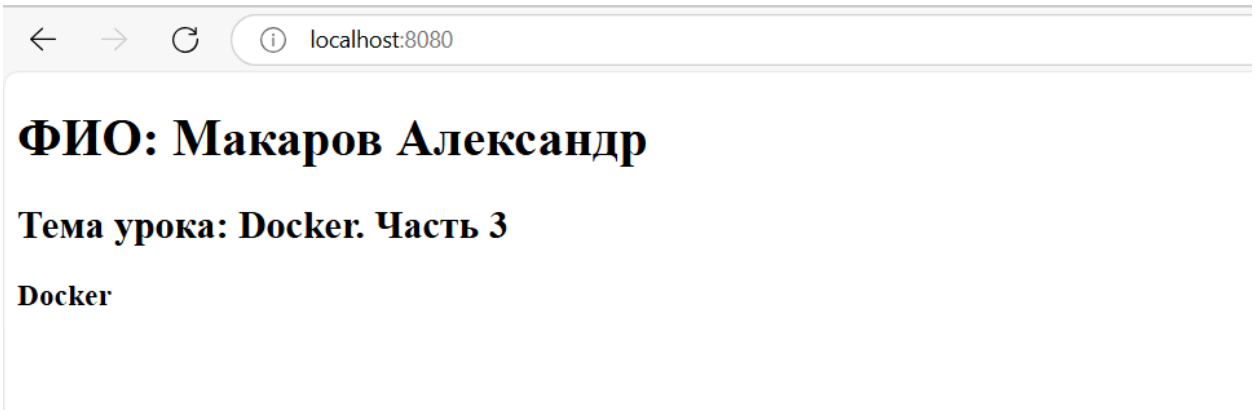
Проверка работы нашего сайта с контейнера. Заходим в сам контейнер:

$ 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 вывзвали curl

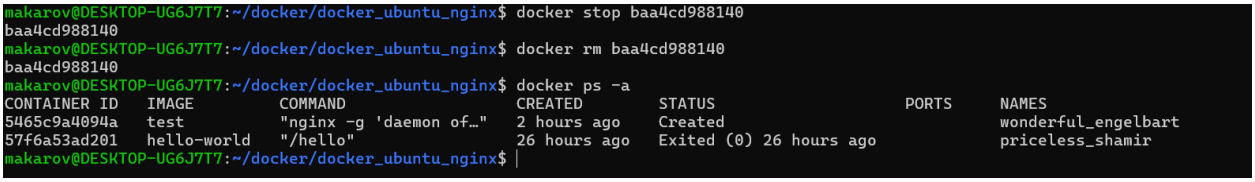
Также наш сайт доступен извне:



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



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

Выполнение второго задания:

1) Создаем пустую директорию и в ней новый файл docker-compose.yml.

$ mkdir /mkdir /docker/docker_compose_postgres
$ touch docker-compose.yml

Вносим изменения в файл docker-compose.yml

```yaml
version: '3'

services:
  db:
    image: postgres:latest
    environment:
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: admin
      POSTGRES_DB: tms

  web:
    image: nginx:latest
    ports:
      - "8080:8080"
    volumes:
      - ./tms.conf:/etc/nginx/conf.d/tms.conf
      - ./tms_man.html:/usr/share/nginx/html/tms_man.html
    links:
      - db
```

1. В первой строке мы определяем инструкцию version: 3 по заданию.
1.1 Дальше в нем мы определяем два сервиса, где postgres под названием сервиса db и nginx:latest под названием web.
1.2 Определяем сервис для базы данных PostgreSQL:
1.2.1 В первом сервисе по заданию используем образ postgres:latest, это пакет с которого будет собираться/устанавливаться СУБД PostgreSQL. Также задаем создание пользователя admin с паролем admin + создание БД под названием tms.
1.3. Определяем сервис для веб-сервиса на основе образа NGINX:
1.3.1 Используем образ NGINX, задаем порты 8080:8080;
1.3.2 Задаем путь к конфигурационным файлам NGINX внутри контейнера, с помощью инструкции “volumes:” .
1.3.3 Определяем ссылку на сервис базы-данных в сервисе NGINX.
1.4 Сохраняем файл docker-compose.yml.

2) Добавляем конфигурационные файлы NGINX, сам файл конфигурации сайта + нашу html-страницу.

Файл конфигурации NGINX нашего сайта:

```
#tms.by

server {
    listen 8080;
#   server_name localhost;

    root /usr/share/nginx/html;
    index tms_man.html;

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

Страница нашего сайта:

```
<!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-Compose</h3>
</body>
</html>
```

После запускаем docker-compose:

$ docker-compose up --build \\ запуск начинаем с директории, где лежит сам файл docker-compose

```
makarov@DESKTOP-UG6J7T7:~/docker/docker_compose_postgres$ docker-compose up --build
WARN[0000] /home/makarov/docker/docker_compose_postgres/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove i
t to avoid potential confusion
[+] Running 2/2
 ✔Container docker_compose_postgres-db-1   Created                                                                                          0.0s
 ✔Container docker_compose_postgres-web-1  Created                                                                                          0.0s
Attaching to db-1, web-1
db-1   |
db-1   | PostgreSQL Database directory appears to contain a database; Skipping initialization
db-1   |
db-1   | 2025-02-07 11:27:53.220 UTC [1] LOG:  starting PostgreSQL 17.2 (Debian 17.2-1.pgdg120+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 12
.2.0-14) 12.2.0, 64-bit
db-1   | 2025-02-07 11:27:53.253 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
db-1   | 2025-02-07 11:27:53.253 UTC [1] LOG:  listening on IPv6 address "::", port 5432
db-1   | 2025-02-07 11:27:53.261 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
db-1   | 2025-02-07 11:27:53.268 UTC [29] LOG:  database system was shut down at 2025-02-07 11:22:45 UTC
db-1   | 2025-02-07 11:27:53.310 UTC [1] LOG:  database system is ready to accept connections
web-1  | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
web-1  | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
web-1  | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
web-1  | 10-listen-on-ipv6-by-default.sh: info: IPv6 listen already enabled
web-1  | /docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
web-1  | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
web-1  | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
web-1  | /docker-entrypoint.sh: Configuration complete; ready for start up
web-1  | 2025/02/07 11:27:53 [notice] 1#1: using the "epoll" event method
web-1  | 2025/02/07 11:27:53 [notice] 1#1: nginx/1.27.4
web-1  | 2025/02/07 11:27:53 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
web-1  | 2025/02/07 11:27:53 [notice] 1#1: OS: Linux 5.15.167.4-microsoft-standard-WSL2
web-1  | 2025/02/07 11:27:53 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
web-1  | 2025/02/07 11:27:53 [notice] 1#1: start worker processes
web-1  | 2025/02/07 11:27:53 [notice] 1#1: start worker process 23
web-1  | 2025/02/07 11:27:53 [notice] 1#1: start worker process 24
web-1  | 2025/02/07 11:27:53 [notice] 1#1: start worker process 25
```

$ docker-compose ps -a \\ выводит активные запущенные контейнеры
$ docker-compose exec -it <id-контейнера> bash \\ позволяет приконнектиться к запущенному контейнеру используя bash

Для настройки приходилось заходить на контейнер, для определения директорий и проверки переноса всех наших конфигурационных файлов.

$ docker inspect <id-контейнера> \\ выводит информацию о контейнере

Наш рабочий сайт с использованием docker-compose.yml:

ФИО: Макаров Александр

Тема урока: Docker. Часть 3

Docker-Compose

$ 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