Docker: начало работы

Docker — это популярная программа, в основе которой лежит технология контейнеризации. Docker позволяет запускать Docker-контейнеры с приложениями из заранее заготовленных шаблонов — Docker-образов (Docker images).
Контейнеризация — это технология, которая помогает запускать приложения изолированно от операционной системы. Приложение как бы упаковывается в специальную оболочку — контейнер, внутри которого находится среда, необходимая для работы.
Простыми словами контейнер — это некая изолированная песочница для запуска ваших приложений.

Установка Docker на Windows WSL2
В Windows 11 и Windows 10 (сборка 19041 и выше) для установки WSL достаточно одной команды (PowerShell с правами администратора):

```
wsl --install
```

Эта команда включит все необходимые компоненты и установит дистрибутив Linux (по умолчанию Ubuntu), вам нужно будет только перезагрузить компьютер.

Подробный процесс установки описан на сайте Microsoft https://docs.microsoft.com/ru-ru/windows/wsl/install-win10 там же указаны минимальные требования.

Если у команда *wsl --install* не сработала, необходимо проделать следующие шаги:

1) Запускаем PowerShell с правами администратора и включаем компонент «Подсистема Windows для Linux», для этого вводим команду:

```
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
```

2) Далее необходимо включить необязательный компонент «Платформа виртуальных машин», для этого в PowerShell с правами администратора выполняем команду:

```
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
```

Перезапускаем компьютер.

3) Скачиваем и устанавливаем пакет обновления ядра Linux https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi

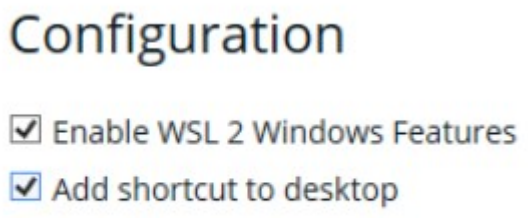
4) Выбираем WSL 2 в качестве версии по умолчанию, если этого не сделать новые дистрибутивы Linux будут установлены в WSL 1.
Вновь запускаем PowerShell с правами администратора и добавляем команду:

```
wsl --set-default-version 2
```

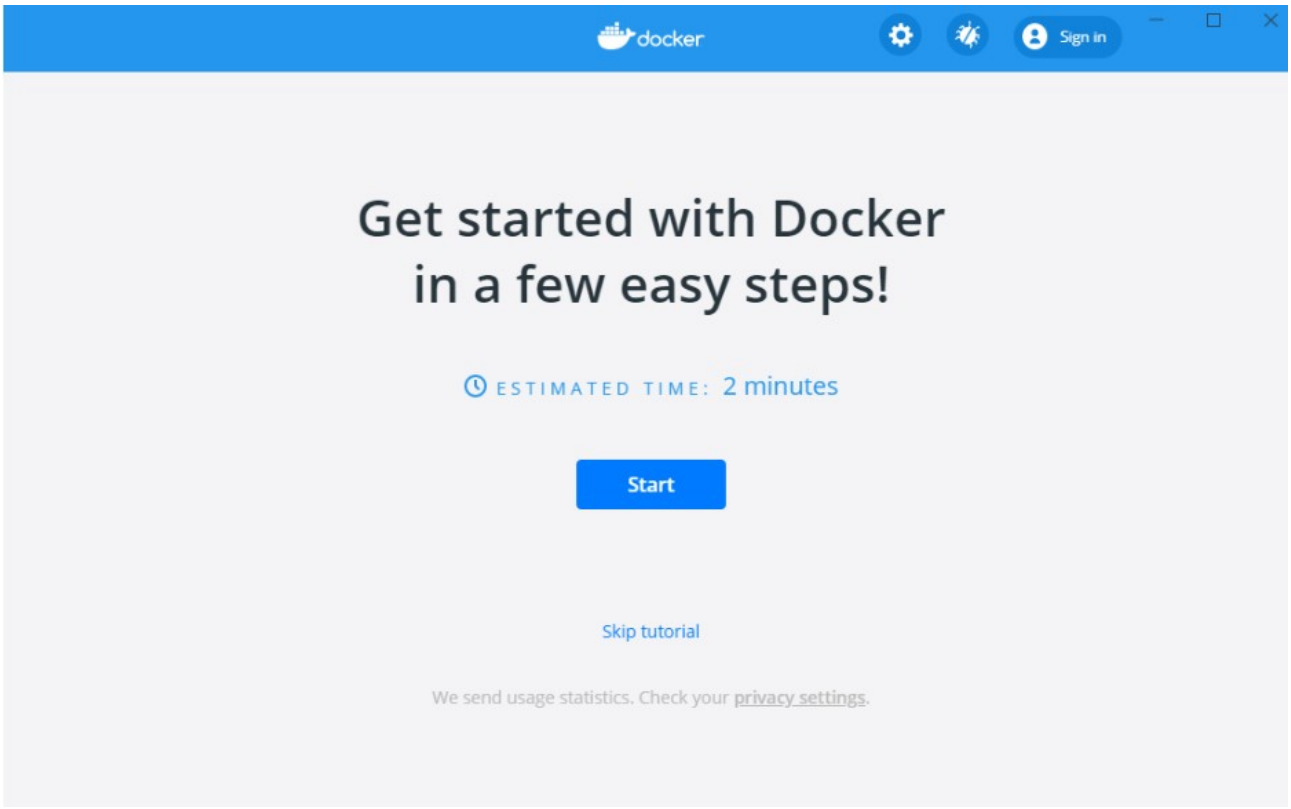
4) Выбираем в магазине Microsoft Store нужный нам дистрибутив Linux и установить его, как обычное приложение из магазина. Например, Ubuntu 18.04 https://www.microsoft.com/store/apps/9N9TNGVNDL3Q
Запускаем установленную Ubuntu, задаём логин и пароль.

5) Установка Docker

Теперь установим Docker Desktop WSL 2 backend, идем по ссылке https://hub.docker.com/editions/community/docker-ce-desktop-windows/ Скачиваем и устанавливаем Docker Desktop for Windows.
Запускаем скачанный файл и производим обычную установку приложения Windows.
При установке убедитесь что установлена галочка на **Enable WSL 2 Windows Features**.



После установки следуйте инструкциям и перелогиньтесь в Windows, Докер будет запущен при следующем входе в Windows, иногда в первый раз может понадобится довольно длительное время.

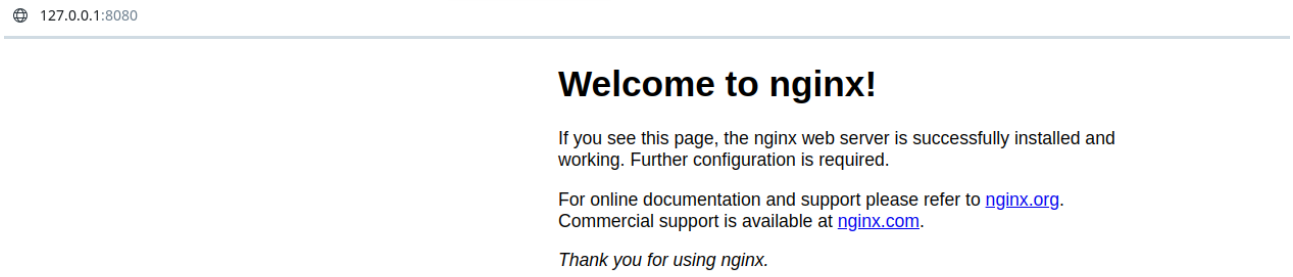


Установка Docker на Linux
Выполните установку согласно документации, выбрав версию вашего Linux-дистрибутива: https://docs.docker.com/engine/install/

6) Застим HTTP-сервер Nginx используя Docker. Для этого введём следующую команду в консоле:

```
docker run -p 8080:80 nginx:latest
```

Далее откроем браузер и введём в адресную строку: 127.0.0.1:8080. Откроется страница приветствия Nginx.



Теперь разберёмся подробнее, что происходит, когда мы вводим команду
docker run -p 8080:80 nginx:latest
Она выполняет следующее:

- Скачивает docker-образ — шаблон для создания Docker-контейнера — nginx:latest из публичного репозитория Docker Hub. Docker-образ содержит все необходимое для запуска приложения: код, среду выполнения, библиотеки, переменные окружения и файлы конфигурации. На странице Nginx в Docker Hub https://hub.docker.com/_/nginx можно найти Docker-образ nginx:latest, где latest — это тег (метка, снимок), который ссылается на самый свежий docker-образ и описывает его.
- Запускает Docker-контейнер с помощью Docker-образа
- Пробрасывает порт. Процессы в Docker-контейнерах запускаются в изоляции от ОС, то есть все порты между ОС и Docker-контейнером закрыты. Для того, чтобы мы смогли обратиться к Nginx, нужно пробросить порт, что и делает опция -p 8080:80, где 80 — это порт Nginx внутри контейнера, а 8080 — порт в локальной сети ОС.

Мы убедились, что Docker запущен, контейнеры создаются.

Запуск и настройка Docker-контейнеров

Выше мы запустили контейнер с веб-сервером Nginx, теперь поработаем с контейнерами более детально.

В Docker контейнер представляет собой окружение для выполнения какого-либо одного процесса. Это изолированная среда, в которой есть всё необходимое для выполнения нужного процесса, и нет ничего лишнего. Создание контейнера Docker выполняется в момент его запуска, и эти процессы запускаются с помощью команды **docker run**. Давайте сначала рассмотрим её синтаксис и опции.

Синтаксис команды docker run выглядит следующим образом:

docker run опции образ команда

Утилите обязательно надо передать образ, на основе которого будет создан контейнер. Образ может быть локальным или указывать на образ, который надо загрузить из сети. Опции позволяют настроить контейнер и параметры его запуска более детально. Сама команда позволяет переопределить программу, которая выполняется после запуска контейнера. Например, выполнив /bin/bash, вы можете подключится к самому контейнеру.

Рассмотрим основные опции утилиты, которые мы будем использовать:

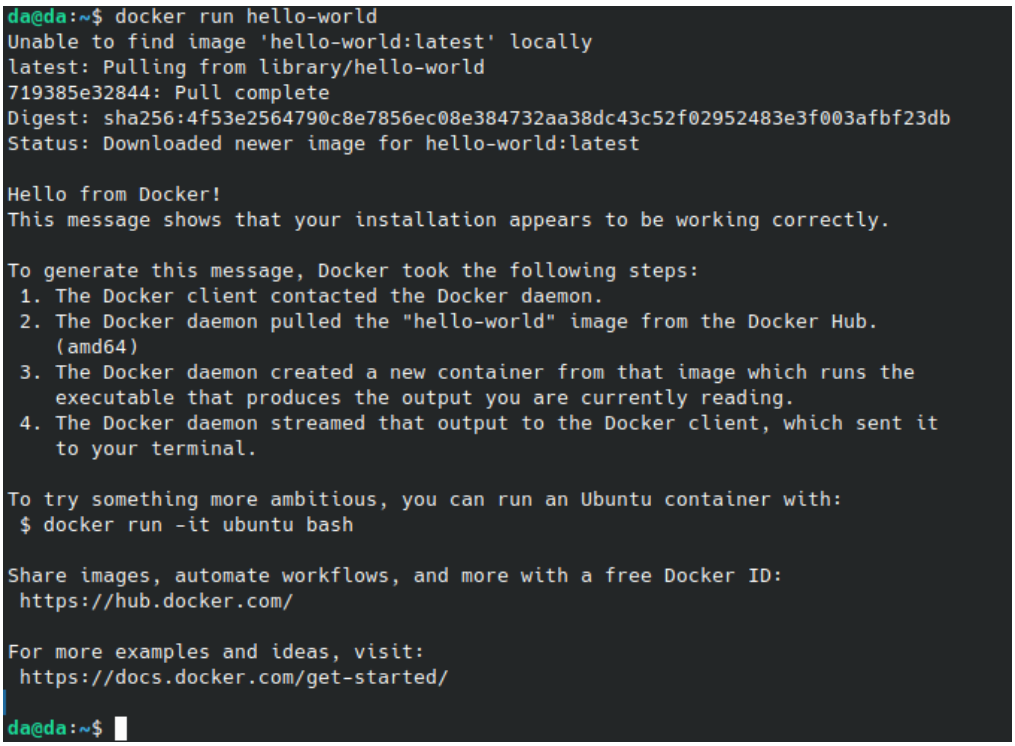
-d - запускает контейнер в фоновом режиме;
-t - прикрепляет к контейнеру псевдо-TTY-консоль;
-i - выводит в терминал STDIN поток контейнера;
--name - имя контейнера, по которому потом можно будет к нему обращаться;
--dns - устанавливает DNS-серверы для контейнера;

--network - тип сети для контейнера, может принимать такие значения: bridge (используется по умолчанию), none, host. Также можно передать идентификатор сети Docker, к которой надо подключится;
--add-host - добавляет строчку в файл /etc/hosts;
--restart - указывает, когда надо перезапускать контейнер. Возможные значения: no, on-failure, always, unless-stopped;
--rm - удаляет контейнер после завершения его работы;
-m, --memory - количество оперативной памяти, доступное Docker-контейнеру;
--memory-swap - объём памяти раздела подкачки, доступный в контейнере;
--cpus - количество ядер процессора, доступных в контейнере;
--shm-size - размер файла /dev/shm;
--device - позволяет монтировать устройства из папки /dev в контейнер;
--entrypoint - позволяет переопределить скрипт, который выполняется при запуске контейнера, перед запуском основной команды;
--expose - позволяет пробросить несколько портов из контейнера в хост-систему;
-P - пробрасывает все порты контейнера в хост-систему;
-p - переносит все порты контейнера в хост-систему без смены номера порта;
--link - позволяет настроить связь контейнеров Docker;
-e - добавляет переменную окружения в контейнер;
-v, --volume - позволяет монтировать папки хоста в контейнер;
-w - изменяет рабочую директорию контейнера.

1) Создание Docker-контейнера. Чтобы создать и запустить контейнер с параметрами, заданными в образе по умолчанию, просто запустите команду без параметров. Давайте воспользуемся контейнером hello-world, который как раз для этого и предназначен:

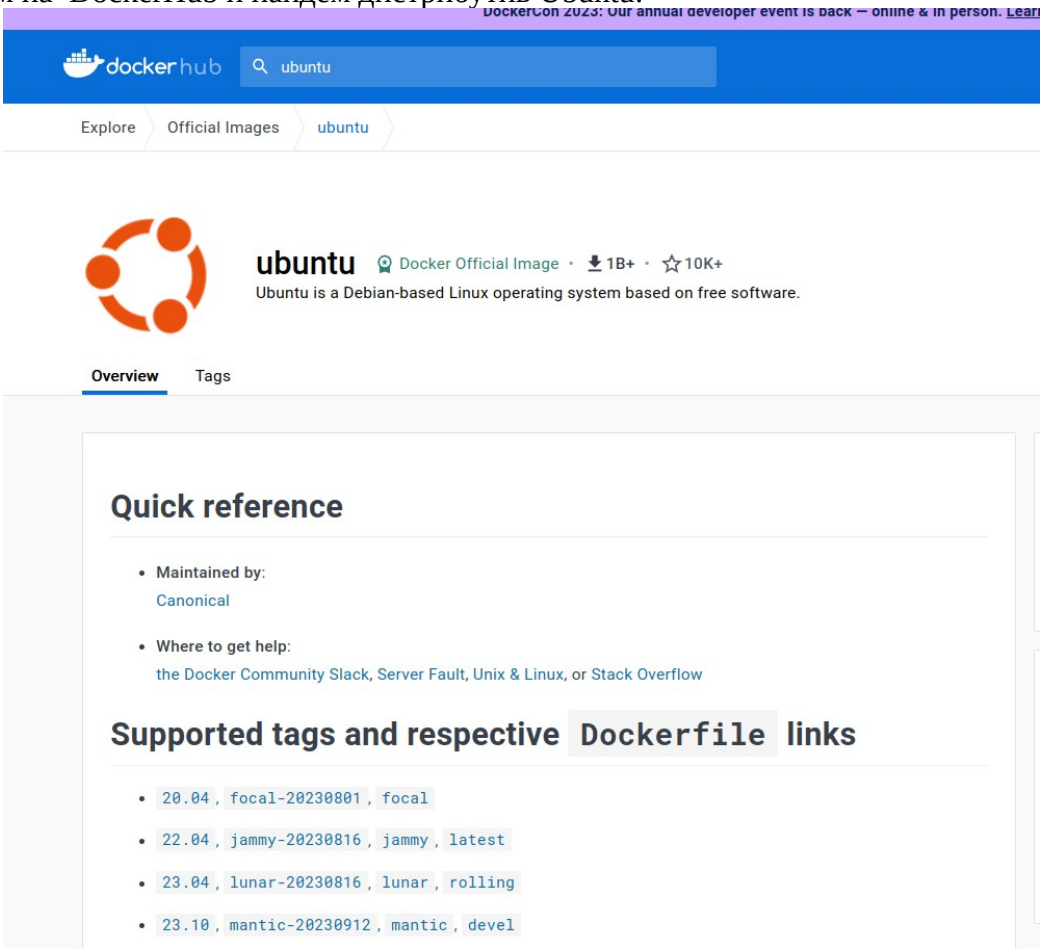
```
docker run hello-world
```

Получим результат запуска контейнера:



После запуска контейнера Docker будет выведено сообщение с приветствием, и процесс в контейнере завершится.

Для поиска уже готовых образов для контейнеров можно использовать веб-сайт DockerHub: https://hub.docker.com. Здесь есть образы для большинства дистрибутивов и системных сервисов, таких, как Nginx, Apache, PHP-FPM и многих других.
Перейдём на DockerHub и найдём дистрибутив Ubuntu:
Вверху



отображается название контейнера, а чуть ниже - доступные версии. При создании контейнера версия записывается через двоеточие. Например, давайте создадим и запустим контейнер с Ubuntu 22.04. Чтобы к контейнеру было легко получить доступ потом, зададим ему имя с помощью опции —name:

docker run --name MyUbuntu2204 ubuntu:22.04

```
da@da:~$ docker run --name MyUbuntu2204 ubuntu:22.04
Unable to find image 'ubuntu:22.04' locally
22.04: Pulling from library/ubuntu
445a6a12be2b: Pull complete
Digest: sha256:aabed3296a3d45cede1dc866a24476c4d7e093aa806263c27ddaadbdce3c1054
Status: Downloaded newer image for ubuntu:22.04
da@da:~$
```

2) Подключение к контейнеру.
Образ Ubuntu 18.04, на основе которого мы создали контейнер выше, не содержит команды, которая бы постоянно выполнялась, поэтому если вы попытаетесь подключится к нему с помощью команды docker exec, то получите ошибку: You cannot attach to a stopped container, start it first:

docker attach MyUbuntu2204

```
da@da:~$ docker attach MyUbuntu2204
You cannot attach to a stopped container, start it first
da@da:~$
```

Так происходит потому, что окружение не может работать без основного процесса, для которого и создан контейнер. Пока работает процесс, будет работать и контейнер. Как только процесс завершён, контейнер завершается, и созданный нами ранее контейнер полностью бесполезен. Вы не сможете к нему подключится, потому что он будет падать сразу же после старта, а к упавшему контейнеру подключится нельзя. Его можно удалить:

docker rm MyUbuntu2204

А нам надо создать контейнер с командой, которая будет выполняться. Для этого просто передайте команду после имени образа, например /bin/bash. Чтобы контейнер был доступен интерактивно, создавайте его с опциями -i и -t:

docker run -it --name MyUbuntu2204 ubuntu:22.04 /bin/bash

```
da@da:~$ docker run -it --name MyUbuntu2204 ubuntu:22.04 /bin/bash
root@85d4b4d5e022:/#
```

Теперь вы в контейнере и можете выполнять действия прямо в изолированном окружении.

3) Переменные окружения.
Очень часто для изменения настроек контейнера используются переменные окружения. Вы задаёте какую-нибудь переменную окружения, а затем её значение используется вашей программой в самом контейнере для выполнения различных действий. Для задания переменных окружения используется опция **-e**.
Запуск контейнера Docker:

docker run -it -e "MyVar=TTIT" --name MyUbuntu2304 ubuntu:23.04 /bin/bash

```
da@da:~$ docker run -it -e "MyVar=TTIT" --name MyUbuntu2304 ubuntu:23.04 /bin/bash
Unable to find image 'ubuntu:23.04' locally
23.04: Pulling from library/ubuntu
10fb01f4f619: Pull complete
Digest: sha256:f1090cfa89ab321a6d670e79652f61593502591f2fc7452fb0b7c6da575729c4
Status: Downloaded newer image for ubuntu:23.04
root@647b2a981d56:/#
```

Находясь в окружении контейнера, проверим значение переменной «MyVar»:

echo $MyVar

```
root@647b2a981d56:/# echo $MyVar
TTIT
root@647b2a981d56:/#
```

4) Монтирование папок и хранилищ.

Когда вам нужно, чтобы контейнер мог работать с исходными кодами программы или с часто изменяемыми данными, рекомендуется сделать специальную настройку - будет неудобно каждый раз копировать информацию в контейнер. Куда проще примонтировать каталог хоста в контейнер, и все изменения исходников будут сразу же доступны в контейнере. Это очень удобно. Для монтирования каталога следует передать её опции **-v**. Например, примонтируем каталог ~/app_docker в контейнер:

docker run -it -v "/home/da/app_docker/:/mnt" --name MyUbuntu2304_1 ubuntu:23.04 /bin/bash

```
da@da:~$ docker run -it -v "/home/da/app_docker/:/mnt" --name MyUbuntu2304_1 ubuntu:23.04 /bin/bash
root@8854d008a8ff:/# ls /mnt/
01  02  03
root@8854d008a8ff:/#
```

Все файлы, которые мы будем создавать в каталоге «app_docker» будут доступны в файловой системе контейнера в каталоге «mnt»
Хранилища позволяют монтировать в контейнер виртуальный диск, который не удаляется при удалении контейнера.

5) Удаление контейнеров и образов.
Удалим все контейнеры, которые мы уже создали.
Просмотрим все имеющиеся контейнеры:

docker ps —all

```
da@da:~$ docker ps --all
CONTAINER ID   IMAGE          COMMAND       CREATED          STATUS                          PORTS     NAMES
8854d008a8ff   ubuntu:23.04   "/bin/bash"   19 minutes ago   Exited (129) 10 minutes ago               MyUbuntu2
304_1
647b2a981d56   ubuntu:23.04   "/bin/bash"   33 minutes ago   Exited (129) 19 minutes ago               MyUbuntu2
304
da@da:~$
```

Команда выводит все имеющиеся контейнеры.
Для удаления всех контейнеров используем две команды:

docker stop $(docker ps -a -q)

docker container rm $(docker ps -a -q)

```
da@da:~$ docker stop $(docker ps -a -q)
8854d008a8ff
647b2a981d56
da@da:~$ docker container  rm $(docker ps -a -q)
8854d008a8ff
647b2a981d56
da@da:~$ docker ps —all
"docker ps" accepts no arguments.
See 'docker ps --help'.

Usage:  docker ps [OPTIONS]

List containers
da@da:~$
```

Удалим все неиспользуемые образы:

docker image prune

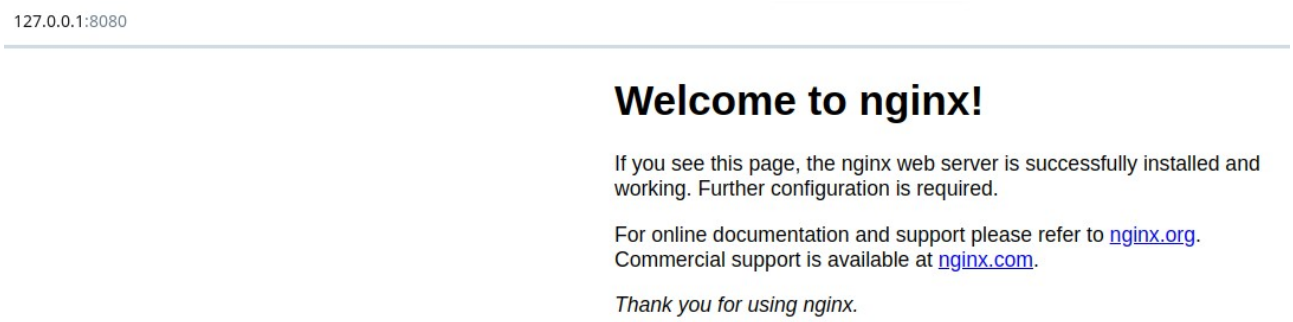
```
da@da:~$ docker image prune
WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N] y
Deleted Images:
deleted: sha256:d671b12d1b01dce5bc88f45b3b9ef2c055d8da54816190cf97730f0c818a31f9
deleted: sha256:554b20182646fd2b765e79f422557aa8bc9ed066063a557740a6381074f5101e

Total reclaimed space: 0B
da@da:~$
```

6) Порты контейнера.
Если вам нужно получить доступ к какому-либо сервису контейнера по порту, например к веб-интерфейсу, этот порт надо пробросить в хост-систему. Для этого используется опция -p. Давайте установим Nginx и пробросим его порт в хост-систему:

docker run --name MyWebServer -p 8080:80 -d nginx

```
da@da:~$ docker run --name MyWebServer -p 8080:80 -d nginx
7b095e4dabc6afdab3ea2e5954c12f03b849425b12136605d4ee5304a2861cc1
da@da:~$
```

После запуска контейнера, нам доступен веб-сервер:



Удалите контейнер с Nginx.

7) Связывание контейнеров.
Связывание контейнеров позволяет настроить взаимодействие между ними. Связанный контейнер будет доступен по сети по его имени. Соответствующая строчка будет автоматически добавлена в файл /etc/hosts контейнера. Для связывания используется опция **--link**. Чтобы закрепить знания, полученные выше, давайте создадим контейнер с базой данных MySQL, а затем свяжем его с PhpMyAdmin.
Сначала создаём контейнер MySQL с постоянным хранилищем в /var/lib/mysql. В переменных окружения ему надо передать пароль суперпользователя. Какие переменные окружения ожидает получить контейнер - эту информацию обычно можно найти на странице контейнера на DockerHub. Используйте опцию **-d**, чтобы контейнер запустился в фоновом режиме:

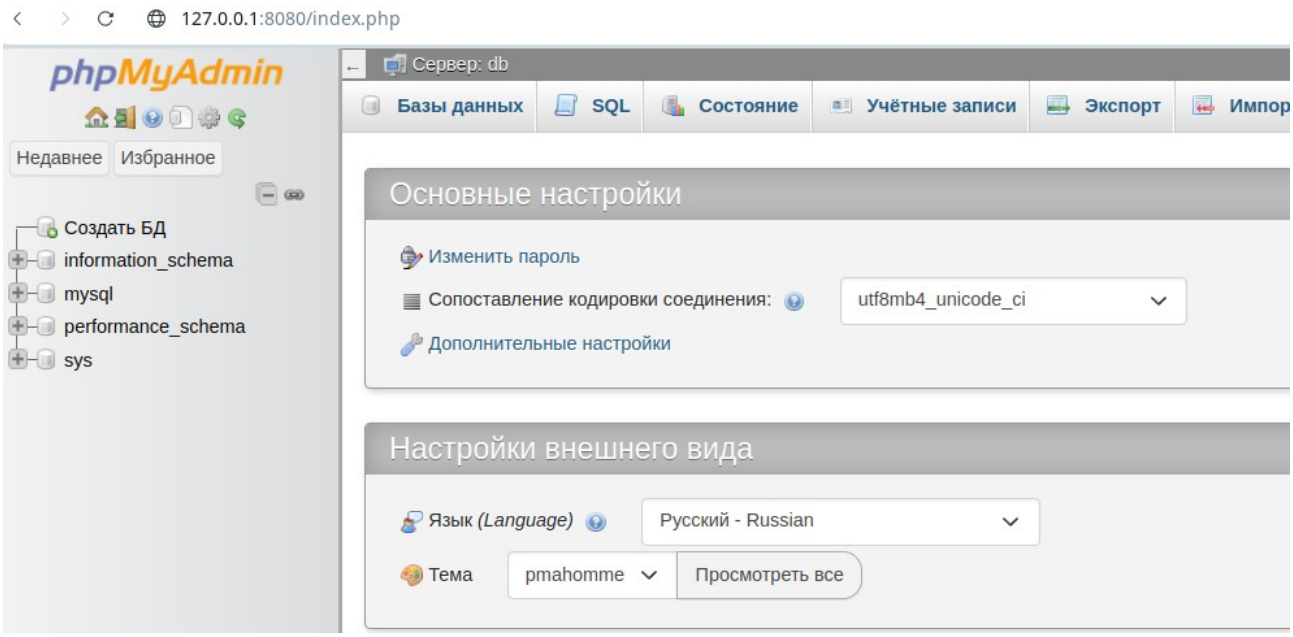
docker run -v mysql_volume:/var/lib/mysql --name MySQL -e MYSQL_ROOT_PASSWORD=passwd -d mysql:8.0

```
da@da:~$ docker run -v mysql_volume:/var/lib/mysql --name MySQL -e MYSQL_ROOT_PASSWORD=passwd -d mysql:8.0
Unable to find image 'mysql:8.0' locally
8.0: Pulling from library/mysql
5262579e8e45: Pull complete
741b767e25b7: Pull complete
06e0c37837cf: Pull complete
c6f5d3670db7: Pull complete
d5c567b29c3e: Pull complete
323a74fdf36b: Pull complete
130e11b8eb71: Pull complete
e92f1f2dd77c: Pull complete
43c0f03962c9: Pull complete
6194c2f9ce13: Pull complete
a235a73ec4d4: Pull complete
Digest: sha256:a7a96a9dbf6f310703c4e0c61086b23c5835c33a05544cdc952a7cd0b8feb675
Status: Downloaded newer image for mysql:8.0
e92def49b71defa5dfb5200b99affd7b2974a23a437dbad2ae6368b35f1e88cc
da@da:~$
```

Затем создаём контейнер с PhpMyAdmin для доступа к этой базе данных и связываем с ним контейнер MySQL под именем db:

docker run --name PhpMyAdmin -d --link MySQL:db -p 8080:80 phpmyadmin/phpmyadmin

Также здесь мы пробрасываем порт 80 в хост-систему, чтобы получить доступ к веб-интерфейсу. Теперь можно проверять в браузере.



8) Сеть для контейнеров.
Контейнеры можно не связывать. Если надо объединить три и больше контейнеров между собой, то куда удобнее сразу связать их в одну общую сеть, чем создавать множество подключений для каждого из этих контейнеров. Все объекты в одной сети будут иметь доступ к друг другу по их имени. Сначала необходимо создать сеть:

docker network create -d bridge docker_network

Посмотреть список созданных сетей можно командой:

docker network list

```
da@da:~$ docker network list
NETWORK ID     NAME             DRIVER    SCOPE
5ecfdb8b887c   bridge           bridge    local
b7c7af3fc19f   docker_network   bridge    local
263b29de1055   host             host      local
de18a8afb25e   none             null      local
da@da:~$
```

Теперь можно её использовать. Объединим с помощью сети наш MySQL- и PhpMyAdmin-сервера. Для этого надо их остановить и удалить:

docker stop MySQL

docker stop PhpMyAdmin

docker rm MySQL

docker rm PhpMyAdmin

Затем создаём:

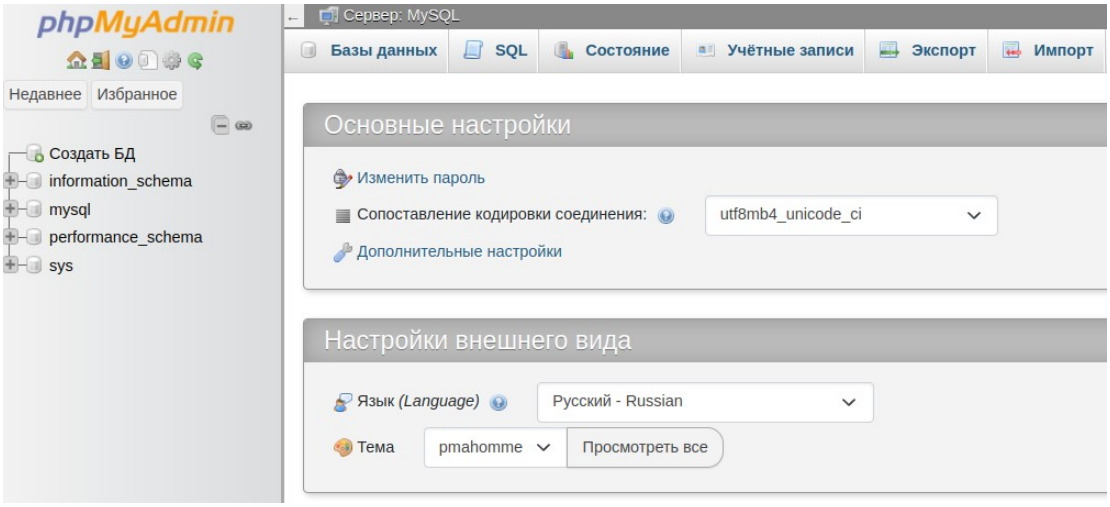
docker run -v mysql_volume:/var/lib/mysql --network docker_network --name MySQL -e MYSQL_ROOT_PASSWORD=passwd -d mysql:8.0

А для PhpMyAdmin теперь надо передать хост, на котором расположена база данных в переменной окружения **PMA_HOST**:

docker run --name PhpMyAdmin -d --network docker_network -e PMA_HOST=MySQL -p 8080:80 phpmyadmin/phpmyadmin

```
da@da:~$ docker run -v mysql_volume:/var/lib/mysql --network docker_network --name MySQL -e MYSQL_ROOT_PASSWOR
D=passwd -d mysql:8.0
921eae4d4b9d1d25c0ad876d162f8d55cadb28cabe09abdc6d9402162e7769b8
da@da:~$ docker run --name PhpMyAdmin -d --network docker_network -e PMA_HOST=MySQL -p 8080:80 phpmyadmin/phpm
yadmin
ef242a53c437e0f34ad5f543524f906b394d0facd91c52f4d3cf4d23ef685994
da@da:~$
```

Проверяем, что всё работает:



Практическое задание

1. Проделать все шаги от установки Docker, до работы с сетью контейнеров.
2. Составить отчёт о проделанной работе.
3. Дать определения основным понятиям используемым в Docker (см. Тезисы лекции по Docker на курсе: https://moodle.tomtit-tomsk.ru/course/view.php?id=991).
4. Защитить отчёт.

Литература

1. Установить Docker на Windows 10 WSL2. https://tretyakov.net/post/ustanovit-docker-na-windows-10-wsl2/
2. Документация Docker. https://docs.docker.com
3. Установка Linux в Windows с помощью WSL. https://learn.microsoft.com/ru-ru/windows/wsl/install
4. Гайд по Docker: что это такое, зачем его использовать и как с ним работать. https://ru.hexlet.io/blog/posts/gid-docker#chto-takoe-docker
5. Запуск контейнера Docker. https://losst.pro/zapusk-kontejnera-docker
6. Использование Docker для чайников. https://losst.pro/ispolzovanie-docker-dlya-chajnikov