Практическое занятие 3: Docker

1. Установка Docker

❗ Установку Docker сейчас производить не требуется, он уже установлен на машине appserver.

Если потребуется, можно ставить по официальной документации.
На Ubuntu: https://docs.docker.com/engine/install/ubuntu/

Скрипт одного из вариантов (выполнять в терминале виртуальной машины):

```
# Переходим в режим суперпользователя
sudo -i

# Удаляем старые версии (рекомендуется)
apt-get remove docker docker-engine docker.io containerd runc

# Установка репо
apt-get update
apt-get install \
     ca-certificates \
     curl \
     gnupg \
     lsb-release -y

# Добавляем GPG-ключ
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg -
-dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

# Установка последней стабильной версии репо
echo \
  "deb [arch=$(dpkg --print-architecture) signed-
by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null

# Установка Docker Engine Community edition (в отличии от EE - enterprise edition)
 apt-get update
 apt-get install docker-ce docker-ce-cli containerd.io -y

# Закончили с установкой, проверяем статус службы
systemctl status docker
* Результат: active (running)

# Переключимся на нормального пользователя
logout

# Чтобы пользователь мог подключаться к докеру, его надо добавить в user-группу docker.
sudo nano /etc/group
* в строке docker:x:999:  в конец строки добавим имя нашего пользователя (vagrant)
* как вариант, можно использовать команду usermod:
```

```
sudo usermod -aG docker <username>
```

Проверяем:
```
id <username>
```
* Результат: в списке появилась группа docker

Чтобы подхватились изменения в группе, переоткроем сессию
```
logout
vagrant ssh
```

Посмотрим список скачанных образов
```
docker images
```
* Результат: команда выполнена, в списке вначале пусто.

2. Базовые команды

Посмотрим список скачанных образов
```
docker images
```

Хорошо бы теперь протестировать. Для этого Докер сделали образ, который просто выводит "Привет".
Запуск контейнера из образа hello-world
```
docker run hello-world
```

В документации этот шаг включен в Post-installation steps. Там есть и другие полезные вещи, например:

Configure Docker to start on boot
```
 sudo systemctl enable docker.service
 sudo systemctl enable containerd.service
```

В страницы по пост-установке есть ссылка на небольшую обучалку Getting Started, пройти будет нелишним.

Посмотреть список активных контейнеров
```
docker ps
```

Он же - вместе с созданными, но остановленными контейнерами
```
docker ps -a
```

* Результат: выводится id, имя образа, команда (скрипт), время создания, а также имя контейнера. Если вы не задали имя явно - это будет комбинация прилагательного и селебрити.

Контейнер hello-world выполнил скрипт и завершился, но чаще создаются постоянно работающие контейнеры. Для этого в нем при запуске должен запускаться некоторый цикл.

3. Создание контейнера

У образа есть имя и теги.
Теги нужны больше для версионности, по-умолчанию тег равен latest. Однако, является плохой практикой бездумно использовать последние версии образов.

Для примера скачаем образ nginx по тегу
```
docker pull nginx:mainline-alpine-perl
```

Посмотрим на список
```
docker images
```

Поднимем контейнер
```
docker run --name myweb -d -p 7090:80 nginx: mainline-alpine-perl
```
* в этой команде:
    --name имя контейнера
    -d использовать, чтоб запустить в бекграунде. hello-world просто выводил строку и завершался, а этот будет запущен постоянно и заблокирует нам сессию терминала
    -p 7090:80 # проброс внешнего порта 7090 на 80 порт контейнера
    nginx:mainline-alpine-perl имя и тег образа

По умолчанию nginx, как веб-сервер, поднимается на порту 80 (это, конечно, настраивается).

Посмотреть на сделанные настройки можно в списке запущенных процессов
```
docker ps
```

Проверка функционирования
* открываем браузер, идем на хост:7090. Если надо – откройте этот порт в файрволе.
* Результат: видим приветствие nginx

Остановить контейнер
```
docker stop <containerId или containerName>
```

Увидеть контейнер в списке остановленных
```
docker ps -a
```

опять поднимем
```
docker start <containerId | containerName>
docker ps
```

Взглянем на процессы хоста – там есть nginx. Этот наш, из контейнера.
```
ps -ef
```
* Результат: nginx там есть

4. Запуск команд внутри контейнера.

Простой вариант - использовать docker exec:
```
docker exec myweb ls /
```
* Здесь myweb – имя контейнера, ls / - команда, которая будет выполнена.

Чтобы не ограничиваться одной командой нужно подключиться к шеллу внутри контейнера.
Для этого используется комбинация ключей -i (“interactive”), -t (“TTY”)
```
docker exec -it myweb /bin/sh
```
* баш есть почти всегда, но иногда это /bin/sh

Это примерно как "запусти команду в контейнере и подключи меня к этому процессу"

мы внутри, попробуем простые команды
```
ls
ifconfig
ps
```

Возвращаем управление назад на хост
```
exit
```

5. Удалить образ.

Как удалить образ
```
docker rmi nginx:mainline-alpine-perl
```

* Так как есть запущенный контейнер, rmi вернет ошибку.

Правильно так:
Проверить какие контейнеры запущены
```
docker ps
```
Остановить те, что мешают
```
docker stop <containerId>
```
Удалить созданные и остановленные контейнеры
```
docker rm < containerId>
```
Теперь можно удалить образ
```
docker rmi <imageName>
```

Чистим за собой
Классический способ использует ключ -q, который выводит только хеши:
```
docker rm $(docker ps -a -q)
docker rmi $(docker images -q)
```

6. Docker volumes

Метод 1 (Bind)

* Найдем на DockerHub официальный образ MySQL
Explore Docker's Container Image Repository | Docker Hub
- Почему официальный? Ему можно доверять и у него есть доки
- В секции How to use this image:
  - Опция -e это Экспорт переменных, -d запуск в фоне

А что с volumes?
- Секция Caveats: where to store data
  - опция -v /my/own/datadir:/var/lib/mysql мапит локальную папку в контейнер. Это и есть volume. А если это нефициальный образ MySQL, данные могут лежать не в этой папке, а где-то еще и надо будет импользовать inspect для определения где именно.

Скачиваем образ
```
docker pull mysql:5.7
```

Создаем внутри виртуалки папку для обмена данными с контейнером. Создайте в ней что-нибудь из файлов.
```
mkdir /home/vagrant/datavol
```

Создаем и запускаем контейнер
```
docker run --name vprodb -d -e MYSQL_ROOT_PASSWORD=secretpass -p 3030:3306 -v /home/vagrant/datavol:/var/lib/mysql mysql:5.7
```

Посмотрим на контейнер
```
docker ps
```

Зайдем внутрь
```
docker exec -it vprodb /bin/bash
```

Список файлов в /var/lib/mysql контейнера соответствует /home/vagrant/datavol?
```
ls /var/lib/mysql
```
Выход
```
exit
```
Список файлов в /home/vagrant/datavol – да, соответствует
```
ls datavol
```

Теперь удалим контейнер и убедимся, что данные остались.
```
docker stop vprodb
docker rm vprodb
ls datavol
```

Такой метод больше используется для передачи кода в контейнер. Для сохранения данных, собранных в контейнере, больше подходит другой метод, volumes.

Метод 2 (Volumes)

Создать том
```
docker volume create mydbdata
```

Посмотреть на содержимое тома
```
docker volume ls
```
Создаем и запускаем новый контейнер
```
docker run --name vprodb -d -e MYSQL_ROOT_PASSWORD=secretpass -p
3030:3306 -v mydbdata:/var/lib/mysql mysql:5.7
```

Тома создаются в папке /var/lib/docker/volumes
Посмотреть на неё:
```
sudo -i
ls /var/lib/docker/volumes/
ls /var/lib/docker/volumes/mydbdata/_data/
logout
```

Посмотреть подробности о контейнере
```
docker inspect vprodb
```
* Можно увидеть:
- pid # ид процесса на хосте
- LogPath # где лежат логи, которые доступны через docker logs
- Binds # привязка томов
- IPAddress # обратите внимание, это внутренняя подсеть.

Посмотреть логи контейнера
```
docker logs vprodb
```

Чистим за собой
```
docker stop vprodb
docker rm vprodb
docker rmi mysql:5.7
```

7. Собираем образы

Поднимем статичный веб-сайт на основе шаблона с tooplate.com.

```
mkdir images
cd images/

# Скачивам шаблон
wget https://www.tooplate.com/zip-templates/2122_nano_folio.zip
sudo apt install unzip -y
unzip 2122_nano_folio.zip  # docker умеет распаковывать только tar.gz
cd 2122_nano_folio/

# Запихнем все в понятный для докера архив
tar czvf nano.tar.gz *
mv nano.tar.gz ../
cd ..
ls

# теперь у нас есть свой тарбол и можно удалить лишнее и сделать красиво
rm -rf 2122*
mkdir nano
mv nano.tar.gz nano/
cd nano/
ls      # архив теперь здесь

# Создаем настроечный файл для контейнера
# Убирайте комментарии из из кода при копировании заготовок откуда-нибудь
      # чем меньше слоев тем лучше
      # мы не можем выполнять часть команд, например, systemctl. Поэтому запускаем апач бинарником
      # ADD умеет распаковать архив, но только тарболлы

nano Dockerfile

      FROM ubuntu:latest
      LABEL "Author"="Me"
      LABEL "Project"="nano"
      RUN apt update && apt install git -y
      RUN apt install apache2 -y
      CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
      EXPOSE 80
      WORKDIR /var/www/html
      VOLUME /var/log/apache2
      ADD nano.tar.gz /var/www/html
```

Теперь надо выполнить docker build для сборки образа

```
docker build -t nanoimg .
```

* Здесь nanoimg - имя образа,
      . - путь к Dockerfile, текущая директория

В прошлых версиях при установке Apache произошла бы остановка из-за необходимости интерактивного запроса на ввод Geographic area для выставления часового пояса (tzdata). И чтобы сделать его неинтерактивным надо было добавить параметр в Dockerfile через переменную окружения (см ENV):

```
FROM ubuntu:latest
LABEL "Author"="Me"
LABEL "Project"="nano"
ENV DEBIAN_FRONTEND=noninteractive
RUN apt update && apt install git -y
RUN apt install apache2 -y
CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
EXPOSE 80
WORKDIR /var/www/html
VOLUME /var/log/apache2
ADD nano.tar.gz /var/www/html
```

Проверяем результат
```
docker images
```
* Результат: nanoimg появился в списке доступных образов

Протестируем – создадим контейнер
```
docker run -d --name nanowebsite -p 5000:80 nanoimg
docker ps
```
* Результат: контейнер активен
* в браузере идем на localhost:5020 – видим страницу из шаблона

Теперь мы можем даже выложить это на DockerHub. Для этого надо там зарегистрироваться и создать репозиторий (можно private), для примера он ниже имеет название ***myname***.

Пересоберем для удобства публикации в реестр со структурой «аккаунт/репозиторий:тег»
```
docker build -t myname/nanoimg:v2 .
docker login
docker push myname/nanoimg:v2
```

* Теперь можно почистить локальные образы.

Создаем контейнер из публичного образа
```
docker run -d --name nanowebsite -p 5000:80 myname/nanoimg:v2
docker ps
```