**Образ / Image –** план сборки контейнера. Он рассказывает какое окружение разворачивается и какие скрипты запускаются

**Том / Volume** – выделенный жесткий диск, который подключается к контейнерам

**Контейнер / Container -** легковесный исполняемый пакет софта, включающий все необходимое для запуска приложения. Изолированн от окружения и может запускаться на любой машине. Создается контейнер из образов. Из каждого образа можно создать сколько угодно контейнеров

**docker --help** – список доступных команд

**docker image --help** – список доступных команд для образа

**docker volume --help** – список доступных команд для тома

**docker container --help** – список доступных команд для контейнера

**docker container inspect** [container\_name] – посмотреть настройки контейнера

**Контейнеры**

**Запуск и выход из контейнеров**

**docker run** hello-world – создаем контейнер из образа, который вводит текст hello world на экран. **docker run** создает контейнер hello-world это образ. Он по дефолту есть в docker. А так мы можем выбирать свои образы. Можно запустить эту команду, что бы проверить, что docker работает исправно

**docker run -i -t** --name[имя] --rm **[образ]**:[tag] [команда] - запуск контейнера Ubuntu.

* **-i** – позволяет увидеть в терминале все, что будет выводить контейнер
* **-t** – позволяет что бы все команды выводимые с терминала были восприняты как команды для контейнера
* --name [имя] – сами задаем имя контейнера
* --rm– при запуске с этим ключем, контейнер будет автоматически удален, после остановки
* **[образ]** - название образа, который хотим запустить. Например ubuntu
* [tag] – версия образа (при условии что у нас она скачана). Если скачана данная версия не будет, то скачивание происходит автоматом
* [команда] - говорить что мы должны запустить сразу после старта контейнера . Если это не указать, то запустится дефолтная команда, которая может отличаться от той, которая нам нужна. Если мы запускаем ubuntu, то команда будет bash
  + Подмена команды

1. Запускаем контейнер от другой команды
2. Создаем образ на основе контейнера
3. Теперь при создании контейнера из образа – дефолтной командой будет та, которую мы явно указывали в 1 шаге

**docker run -i -t -d** [образ] [команда] – запуск контейнера в фоне. Это нужно, если мы хотим запустить какой то процесс, в который не хотим переходить (например БД)

* **-d** – запустить контейнер в фоне

**docker run -i -t** -w [/путь][образ] [команда]

* **-w [/путь]** – запустить контейнер, из указанной директории

**docker run -p внешний\_порт:внутренний\_порт [образ]** – привязать порты (что бы можно было с хоста достучаться до контейнера)

**docker run -e** Something=Something - указать переменную окружения (например логин, пароль для подключения, имя по которому БД сможет подключиться к чему то (если такое нужно), установка плагинов, ….) Что бы посмотреть активные переменные окружения, нужно зайти в контейнер и ввести **env**

**docker container attach** [имя/id контейнера] – войти в контейнер, который запущен в фоне

**ctrl+p ; ctrl+q** – что бы выйти из контейнера, не закрыв его

**docker container stop** [имя/id контейнера] – закрыть контейнер, не заходя в него

**exit** – закрыть запущенный контейнер

**docker container start** [имя/id контейнера] – запустить закрытый контейнер (запускает его в фоне)

**Просмотр контейнеров**

**docker container ls** – позволяет посмотреть список всех запущенных контейнеров

**docker ps** - позволяет посмотреть список всех запущенных контейнеров

**docker container ls -a** – позволяет посмотреть список всех контейнеров

**docker ps -a -** позволяет посмотреть список всех контейнеров

**Удаление контейнеров**

**docker container rm** [имя/id контейнера] – удаляет контейнер. Можно удалять только остановленные контейнеры

**docker container prune** – удалить все контейнеры

**Image**

**Скачивание образов**

**docker pull** [имя] – скачивает образ. Ubuntu/Python/…

**docker pull** [имя]:[tag] – тэг это версия. Если не указать, то будет скачана последняя версия (latest)

**Просмотр образов**

**docker image ls** – список всех скачанных образов

**Удаление образов**

**docker image rm [id образа]** – удалить образ

**Создание образов.** Зачастую для создания образов используется другой образ

1. Контейнер от основного образа
2. Настраиваем нужный софт
3. Выходим из контейнера
4. Создаем образ на основе контейнера

**docker commit [id контейнера] [имя создаваемого образа]** – создает образ на основе контейнера

**Mount**

**Bind** – подключение хост машины к контейнеру. Позволяет обмениваться информацией между хостом и контейнером.

**Volume** – создание отдельного тома для подключения к любому из контейнеров. Позволяет обмениваться информацией между контейнерами. Сам том хранится в недрах docker демона.

**Tmpfs** – временное файловое хранилище

**Bind**

**docker run -it --mount** **type**=bind,**src**=C:\Users\artiv\Desktop\LearnQA\_Docker\dir\_for\_bind,**target**=/bind/ ubuntu bash

mount:

* **type=bind**/volume/tmpfs
* **src=**[путь к директории для подключения]
* **target**=[путь] – путь внутри контейнера, куда том будет подключен
* **readonly –** контейнер сможет получить данные, но не сможет записать

**Реальное использование**

Запустим контейнер с python, забиндим папку со скриптом на хост машине и запустим скрипт в контейнере, а после выполнения скрипта автоматом удалить контейнер

docker run --rm --mount type=bind,src=C:\Users\artiv\Desktop\LearnQA\_Docker\dir\_for\_bind,target=/bind/,readonly python python /bind/counter.py

Создать контейнер с новым томом, там создать скрипт, после создать новый контейнер с python, подключить созданный том и запустить скрипт в созданном томе

1. docker run -it --rm --mount type=volume,src=python\_program,target=/volume/ ubuntu bash
2. после создаем скрипт в папке volume - echo "for i in range(1, 6): print(i)" > counter.py
3. docker run --rm --mount type=volume,src=python\_program,target=/src/,readonly python python /src/counter.py

Если нам нужно протестировать выполнение скрипта на разных версиях python, то не используя docker пришлось бы удалять текущую версию python и ставить старую. Но используя docker, достаточно скачать образ более старой версии python, поле создать контейнер с этой версией и запустить скрипт

**docker pull** python:[версия]