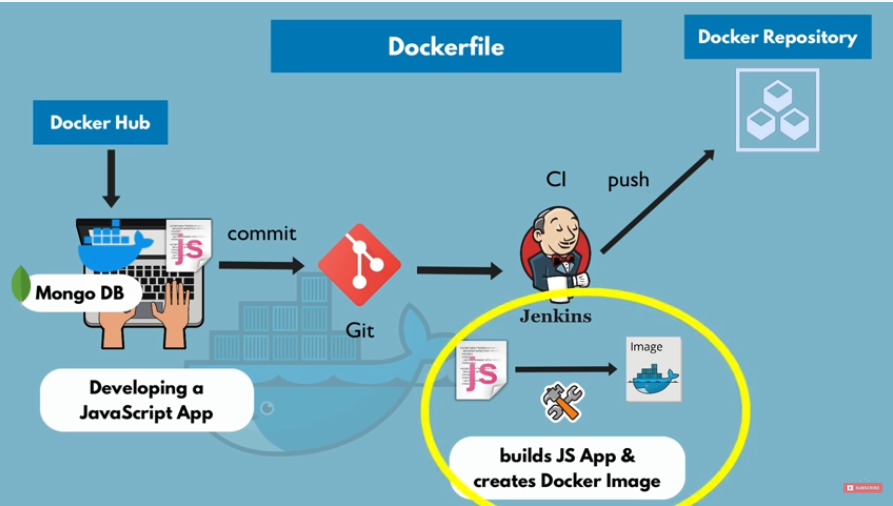
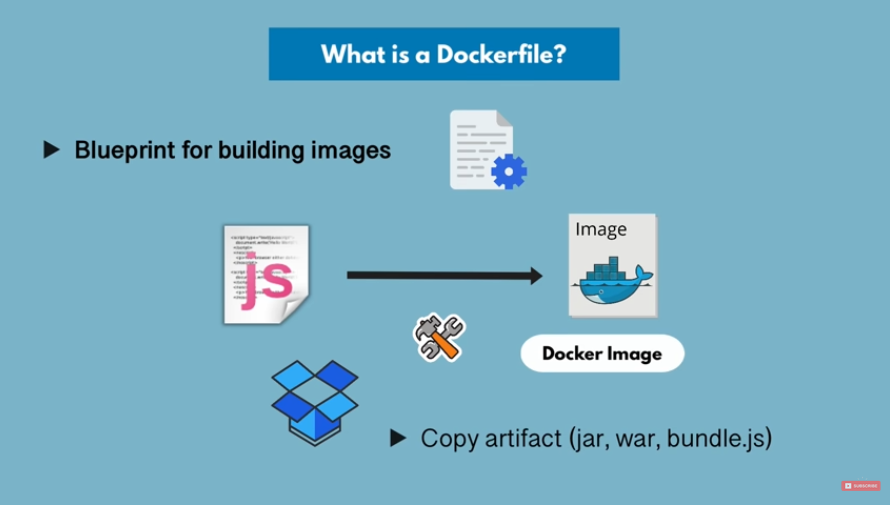
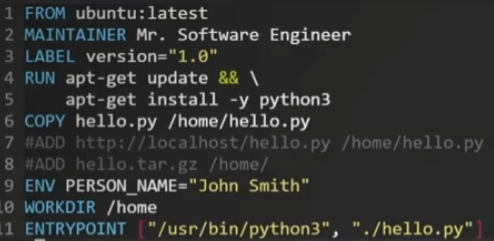
**Dockerfile** – создает образ по заранее прописанному сценарию. Нужен что бы упростить процесс распространения команд для запуска контейнера из какого либо образа. Подобные файлы сообщают Docker о том, как собирать образы, на основе которых создаются контейнеры. По сути это обычный текстовый документ **без расширения**

Jenkins автоматически собирает образ, по докерфайлу и пушит его в репозиторий. Откуда потом его можно запулить на сервер





**Structure of Dockerfile**



* **FROM** [image] – задает родительский образ из которого будет создаваться наш образ
  + **FROM** scratch – если создаем образ, который не зависит от других образов
* **MAINTAINER** – автор докерфайла
* **LABEL** ключ=значение – метаданные
* **ENV** ключ=значение – задает переменные окружения (опционально), которые будут доступны в последующем для конейнеров
* **ARG** ключ=значение– задает переменную, которая будет доступна только во время сборки контейнера (обращаться к ней можно с помощью $ключ). Часто используется, что бы указать версию чего либо
* **RUN** [command] – запустить команду (внутри контейнера). Может быть много команд. Нужно, что бы накатить какие нибудь сервисы apt-get install , … Желательно, если команд несколько, прописывать их под один RUN, дабы не плодить слои
* **COPY** что\_копируем куда\_копируем – копирует в контейнер файлы и папки. Так как мы формируем образ, а не контейнер, то и подключить внешние папки или тома не получится. По этому здесь мы копируем файлы и папки хоста внутрь образа

Нужно указать 2 пути

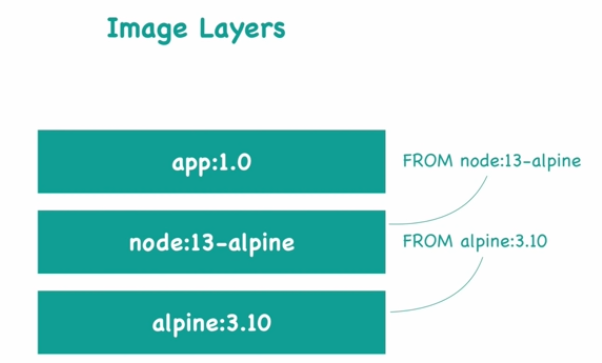
* + Путь до файлов на хост машине, относительно dockerfile (counter.py)
  + Путь куда положить эти файлы внутри контейнера (.) В нашем случае мы кладем эти файлы в рабочую директорию /app

**ADD** что\_копируем куда\_копируем – то же самое, что и COPY, но только позволяет в качестве атрибута указывать ссылку на файл. А так же он автоматически распаковывает архив (если такой был указан в качестве источника)

* **WORKDIR** /path – задает рабочую директорию для всех последующих команд
* **EXPOSE** port — указывает на необходимость открыть порт.
* **VOLUME** /path — создаёт точку монтирования для работы с постоянным хранилищем.
* **ENTRYPOINT [“/путь”]** – указывает команду которая будет выполняться при запуске контейнера

**CMD [“команда”]** – указывает команды которые будут выполняться в entrypoint

**Image layers**



Новый слой в образе создают следующие команды

* FROM
* RUN
* COPY
* ADD

При сборке образа, подразумевается, что мы запускаем команду из папки где находится dockerfile. **Dockerfile должен быть так и назван - Dockerfile**

**docker build -t python\_counter .** – собрать образ

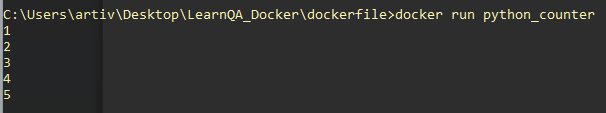
* **-t python\_counter** – это имя образа
* **.** отвечает за то где мы будем искать файлы и директории для команды COPY

Если файл находится в другом месте, то нужно указать ключ -f

**docker build -t python\_counter . -f \path\Dockerfile**

Запустим контейнер. Как видно, там не нужно вводить кучу других ключей и тд.

**docker run python\_counter**



**Собрать образ, который будет самостоятельно запускать автотесты**

К сожалению, dockerfile не умет изменить тип сетевого соединения, так как сам он описывает как образ должен создаться, а не то какие контейнеры на его основе будут запущены. По этому, если мы хотим что бы запускались автотесты, то нужно указать --network host при создании контейнера

docker build -t run\_test C:\Users\artiv\Desktop\LearnQA\_Docker\autotests -f ./Dockerfile

docker run --rm --network host run\_test