**Dockerfile**

файл с инструкциями по сборке образа.

именоватся должен с заглавной D, остальные строчный, без расширения.

если в папке Dockerfile-ов несколько можно применять следующий формат именования:

frontend.Dockerfile, backend.Dockerfile, production.Dockerfile, etc.

**https://docs.docker.com/get-started/docker-concepts/building-images/writing-a-dockerfile/**

вводный курс по Dockerfile

**https://docs.docker.com/build/**

руководство по «сборщикам» docker.

в docker есть несколько модулей отвечающих за сборку образа из Dockerfile: устаревший build; новый buildx, но на момент 27 версии docker еще не реализовавший весь функционал;

bake надстройка над build и buildx расширяющая их функционал и ещё некоторые.

Использовать нужно docker build..., т.к. под капотом эта команда уже использует buildx.

**https://docs.docker.com/reference/dockerfile/**

справочник по инструкциям в Dockerfile

**docker build -t nameCreateImage:tag .**

**docker build -t nameCreateImage -f Dockerfiles/backend.Dockerfile .**

создать образ на основе Dockerfile

-t - имя создаваемого образа и необязательный тэг (v1.0.0)

-f или --file - путь и/или имя Dockerfile с нестандартным именем

**.** - путь к контексту сборки(файлы необходимые для построения образа); точка означает

текущую директорию

**docker commit containerName nameCreateImage:tag**

создать образ из остановленного контейнера.

не является хорошей практикой, рекомендуется использовать Dockerfile, но для

эксперементальных целей подходит.

**.dockerignore**

https://docs.docker.com/build/concepts/context/#dockerignore-files

перечень файлов и директорий не учавствующие в сборке образа.

если при сборке образа используется несколько Dockerfile, можно исключать их

индивидуально и/или целый каталог с ними.

**docker push userNameOnDockerHub/imageName:tag**

https://docs.docker.com/get-started/workshop/04\_sharing\_app/

отправить образ на Docker Hub

**Dockerfile instructions:**

**# Comments**

коментировать можно только построчно

**INSTRUCTION arguments**

формат записи инструкции и её аргумента.

инструкцию docker понимает и строчными буквами, но «ТАКОВ стандарт»).

обрабатываются построчно, начиная с первай, поэтому порядок имеет значение.

**FROM image:tag**

https://docs.docker.com/reference/dockerfile/#from

базовый образ на котором будет основыватся сборка с необязательным тэгом (default tag=latest).

!Внимание, инструкция должна быть первой в файле, как исключение предшествовать могут:

дерективы парсера, переменные ARG глобальной области действия, коментарии.

**RUN, ENTRYPOINT, CMD**

все эти инструкции будут выполнять команду или набор команд, как-бы в cli контейнера, при сборке образа и запуске контейнера на основе собранного образа.

**разница между инструкциями:**

RUN при сборке образа(невозможно переопределить для контейнера);

ENTRYPOINT при запуске контейнера(сложно переопределить для контейнера);

CMD то-же при запуске контейнера, но зачастую используется для передачи флага команде из entrypoint (легко переопределяется при запуске контейнера), хотя тоже может выполнять

команду.

**пример:**

RUN apt update && apt install -y bc

устанавливаем bc при построение образа

ENTRYPOINT ["bc"]

определяем основную команду которая будет запускать bc при запуске контейнера.

команду можно переопределить при запуске контейнера так:

docker run -it --rm **--entrypoint /bin/bash** imageName

CMD ["-l"]

указываем аргумент по умолчанию «-l» с которым будет запускатся основная команда bc.

аргумент легко переопределить при запуске контейнера:

docker run -it --rm imageName **-q**

**exec и shell форма для RUN, ENTRYPOINT и CMD:**

https://docs.docker.com/reference/dockerfile/#shell-and-exec-form

INSTRUCTION ["executable","param1","param2"]

exec форма принимает аргументы в виде JSON массива [], каждый элемент в "..." кавычках

!Настоятельно рекомендуется для ENTRYPOINT и CMD из-за особенностей выполнения:

https://docs.docker.com/reference/build-checks/json-args-recommended/

INSTRUCTION command param1 param2

shell форма принимает аргументы в виде строки

RUN <<EOF

apt update && \

apt -y upgrade

EOF

shell форма с аргументами обернутыми в heredoc (не требует экранирования элементов).

**RUN command1 && command2 && ...**

https://docs.docker.com/reference/dockerfile/#run

выполнить команду как-бы в контейнере строящегося образа; новые файлы созданные при выполнении инструкции RUN сохраняются в новом слое создаваемого образа; каждая последующая инструкция RUN создает новый слой в создаваемом образе; каждая

последующая инструкция RUN имеет доступ ко всем предыдущими слоям

**RUN apt update && apt -y upgrade**

пример инструкции RUN с apt.

!Внимание, сборщик образа не должен сталкиватся с интерактивными вопросами, например команда upgrade может спросить «действительно ли вы хотите обновить пакет» поэтому ей

передан флаг «-y», иначе возникнет ошибка и образ не соберется.

**RUN --mount**

https://docs.docker.com/reference/dockerfile/#run---mount

смонтировать разные типы хранилищ НА ВРЕМЯ ВЫПОЛНЕНИЯ ЭТОЙ ИНСТРУКЦИИ.

поддерживаются типы хранилищ описаны в storage.docx + несколько дополнительных типов

в документации по ссылке.

**RUN --mount=type=bind,source=dir/index.php,destination=/app/index.php**

https://docs.docker.com/reference/dockerfile/#run---mounttypebind

!Внимание, флаг --mount в инструкции RUN работает НЕ ТАК как для контейнера! Внимательно читай далее!

Смонтировать файл с localhost в как-бы контейнер создаваемого образа на ВРЕМЯ ВЫПОЛНЕНИЯ ТЕКУЩЕЙ ИНСТРУКЦИИ RUN, т.е. подразумевается что в текущей инструкции RUN, помиио монтирования, будут выполнены все необходимые действия с файлом. При выполнении всех последующих (или предыдущих) инструкции Dockerfile файл index.php будет уже размонтирован и недоступен. Так-же файл с localhost не будет

смонтирован ни в образ ни в контейнер.

**LABLE**

**LABEL "com.example.vendor"="ACME Incorporated"**

**LABEL com.example.label-with-value="foo"**

**LABEL version="1.0"**

**LABEL description="This text illustrates \**

**that label-values can span multiple lines."**

**LABEL multi.label1="value1" multi.label2="value2" other="value3"**

добавление метаданных к образу.

рекомендуется использовать вместо устаревшего MAINTAINER, наследуются от базовых или родительских образов.

**docker image inspect imageName**

посмотреть данные образа включая метаданные

**docker image inspect --format='{{json .Config.Labels}}' imageName**

посмотреть только метаданные

**EXPOSE 443**

**EXPOSE 80/tcp**

**EXPOSE 80/udp**

декларация порта и протокола (по умолчанию tcp) который должен быть опубликован, является хорошим тоном для Dockerfile.

непосредственно expose порта происходит при создании контейнера и может отличатся от инструкции в Dockerfile:

docker run -p 90:90 imageName

**ENV** MY\_NAME="John Doe"

https://docs.docker.com/reference/dockerfile/#env

переменная, доступная на этапе сборки и в готовом образе.

если нужна переменная дуступная только на этапе сборки образа смотри ARG.

**ARG user1**

**ARG user1=someuser**

https://docs.docker.com/reference/dockerfile/#arg

переменная доступная только на этапе сборки; может иметь значение по умолчанию, но по документации значение должно передаваться при сборке образа флагом:

docker build **--build-arg** user1=what\_user .

!Внимание, переменная при сборке доступна только со строки где она объявленна.

**Предопределенные ARG**

https://docs.docker.com/reference/dockerfile/#predefined-args

Dockerfile содержит несколько объявленных пустых переменных которыми можно воспользоватся при построении образа:

docker build --build-arg **HTTPS\_PROXY**=https://my-proxy.example.com .

эти переменные и их значения не сохраняются в логе построения образа и могут быть

использованны для секретной информации

**Automatic platform ARGs in the global scope**

https://docs.docker.com/reference/dockerfile/#automatic-platform-args-in-the-global-scope

Dockerfile содержит несколько предопределенных переменных , например TARGETPLATFORM=linux/amd64, но для их использования нужно их объявить, сотри

документацию.

**ADD source destination**

**ADD ["source", "destination"]**

https://docs.docker.com/reference/dockerfile/#add

добавить(скопировать) файл или директорию в образ.

инструкция так-же умеет копировать с URL, в процессе копирования распаковывать архив и многое другое.

!Внимание, если требуется просто скопировать файлы в образ, то рекомендуется использовать аналогичную команду COPY, так как ADD имеет больший функционал и его действия могут иметь неожиданный результат.

**ADD index.php /app/**

добавить index.php из текущей директории (рядом с Dockerfile) в контейнер в папку app

**ADD index.php /app/index2.php**

добавить и переименовать файл

**ADD myarchive.tar.gz /usr/src/myapp**

распаковать архив в директорию образа

**ADD https://example.com/myfile.txt /etc/myapp/config**

загрузить файл с URL в папку контейнера

**ADD git@github.com:user/repo.git /usr/src/things/**

загрузить файл с github в папку контейнера

**ADD file1.txt file2.txt /usr/src/things/**

в source можно указывать несколько источников, в таком случае последний аргумент всегда должен указывать на destination директорию заканчивающихся слешом

**ADD --keep-git-dir=true https://github.com/moby/buildkit.git#v0.10.1 /buildkit**

не удалять .git директории при копировании (по умолчанию удаляются).

полезно при разработке приложения в контейнере.

**COPY source destination**

**COPY ["source", "destination"]**

https://docs.docker.com/reference/dockerfile/#copy

скопировать файл или директорию в образ.

инструкция похожа на ADD, хотя имеет меньший функционал, но является предпочтительной, так как её результат более прагнозируемый.

**COPY index.php /app/**

скопировать index.php из текущей директории (рядом с Dockerfile) в контейнер в папку app

**COPY index.php /app/index2.php**

скопировать и переименовать файл

**COPY file1.txt file2.txt /usr/src/things/**

в source можно указывать несколько источников, в таком случае последний аргумент всегда должен указывать на destination директорию заканчивающихся слешом

**COPY --from=nginx:latest /etc/nginx/nginx.conf /nginx.conf**

копировать файл из образа

**COPY --chown=myuser:mygroup myfile.txt /app/**

копировать файл myfile.txt в директорию /app/ внутри контейнера и устанавливаем владельца myuser и группу mygroup

**COPY --chmod=644 config.json /app/**

копировать файл config.json в директорию /app/ и устанавливаем для него права доступа 644

(чтение и запись для владельца, только чтение для группы и остальных)

**VOLUME /directory**

**VOLUME ["/directory"]**

https://docs.docker.com/reference/dockerfile/#volume

декларация точки монтирования хранилища, является хорошим тоном для Dockerfile.

создает точку монтирования в образе, но не определяет тип монтирования или источник.

тип монтирования и источник должен быть задан при создании контейнера или docker сам создаст анонимный том и смонтирут его в папку контейнера.

Примечание: директорию для монтирования в образе можно создать явно, например «RUN

mkdir /directory» если, например, в директорию образа нужно скопировать файлы изначально.

**USER userName**

**USER userName: groupName**

https://docs.docker.com/reference/dockerfile/#user

устанавливает пользователя от имени которого будут выполнятся последующие инструкции при сборке образа и в контейнере.

!Внимание, инструкция USER выполтяет только переключение пользователя, т.е. пользователь должен быть уже создан; в Dockerfile только инструкции следующие после USER будут выполнятся от указанного имени пользователя, по умолчанию все инструкции

выполняются от имени root.

RUN groupadd -r groupName && useradd --no-log-init -r -g groupName userName

создать группу groupName и пользователя userName в этой группе без пароля

RUN groupadd -r -g 1001 groupName && useradd --no-log-init -r -u 1001 -g groupName userName

явное указание user ID и group ID

USER userName

переключится на пользователя userName

**WORKDIR /path/to/workdir**

https://docs.docker.com/reference/dockerfile/#workdir

задает рабочий каталог для любых инструкций RUN, CMD, ENTRYPOINT, COPY и ADD, которые следуют за ней в Dockerfile.

инструкция может быть использована многократно, если в ходе построения образа необходимо менять текущую директорию; поддерживает абсолютный и относительный путь, относительно текущего каталога; по умолчанию это корневая директория /, но эта

инструкция наследуется от родительского образа.

**STOPSIGNAL**

https://docs.docker.com/reference/dockerfile/#stopsignal

устанавливает сигнал системного вызова, который будет отправлен **контейнеру** для выхода.

**HEALTHCHECK**

https://docs.docker.com/reference/dockerfile/#healthcheck

изменения стандартных настроек docker для остановки контейнера при сбое в нём