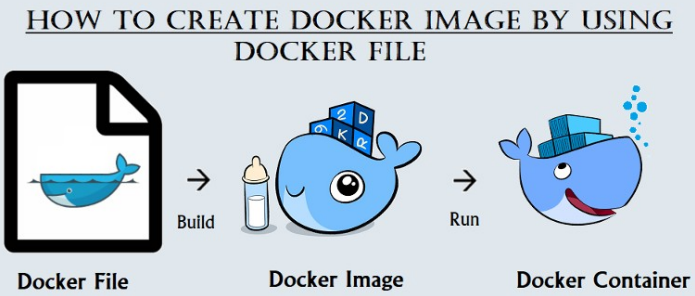
Docker
Создание образов

Файл Dockerfile

Файл Dockerfile содержит набор инструкций, следуя которым Docker будет собирать образ контейнера. Каждая инструкции начинается с новой строки с заглавными буквами. После инструкций идут их аргументы. Инструкции, при сборке образа, обрабатываются сверху вниз. Пример инстукции:

- FROM ubuntu:20.04
- COPY . /app



Список основных инструкций Dockerfile

FROM — задает базовый образ

LABEL — описывает метаданные. Например, сведения об авторе образа.

ENV — создает переменную окружения.

RUN — выполняет команду и создаёт слой образа. Используется для установки в контейнер пакетов.

COPY — копирует файлы и директории в контейнер.

ADD — копирует файлы и директории в контейнер, а также может распаковывать локальные .tar файл.

CMD — описывает команду с аргументами, которую нужно выполнить когда контейнер будет запущен.

ARG — задает переменные для передачи Docker во время сборки образа.

WORKDIR — задает работую директорию для следующей инструкции **CMD** и **ENTRYPOINT**.

ENTRYPOINT — предоставляет команду с аргументами для вызова во время выполнения контейнера.

EXPOSE — указывает на необходимость открыть порт.

VOLUME — создаёт точку монтирования для работы с постоянным хранилищем.

Инструкция FROM

- Файл Dockerfile должен начинаться с инструкции **FROM**, или с инструкции **ARG**, за которой идёт инструкция **FROM**.
- Пример простого Dockerfile:

```
FROM ubuntu:20.04
```

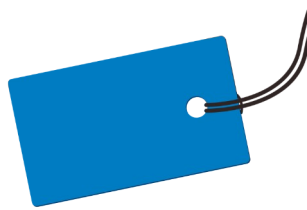
- Ключевое слово **FROM** сообщает Docker о том, чтобы при сборке образа использовался базовый образ, который соответствует предоставленному имени и тегу. Базовый образ ещё называют родительским образом.

Пример Dockerfile

- Взглянем на пример Dockerfile, который собирает маленький образ. В нём имеются механизмы, определяющие команды, вызываемые во время выполнения контейнера.

```
FROM python:3.7.2-alpine3.8
LABEL maintainer="jeffmshale@gmail.com"
ENV ADMIN="jeff"
RUN apk update && apk upgrade && apk add bash
COPY . ./app
ADD https://raw.githubusercontent.com/discdiver/pachy-vid/master/sample_vids/vid1.mp4 \
/my_app_directory
RUN ["mkdir", "/a_directory"]
CMD ["python", "./my_script.py"]
```

Инструкция LABEL



- Инструкция **LABEL** (метка) позволяет добавлять в образ метаданные. В случае с рассматриваемым сейчас файлом, она включает в себя контактные сведения создателя образа. Объявление меток не замедляет процесс сборки образа и не увеличивает его размер. Они лишь содержат в себе полезную информацию об образе Docker, поэтому их рекомендуется включать в файл.

```
FROM python:3.7.2-alpine3.8
LABEL maintainer="jeffmshale@gmail.com"
```

Инструкция ENV

- Инструкция **ENV** позволяет задавать постоянные переменные среды, которые будут доступны в контейнере во время его выполнения. В примере после создания контейнера можно пользоваться переменной ADMIN.

```
FROM python:3.7.2-alpine3.8
LABEL maintainer="jeffmshale@gmail.com"
ENV ADMIN="jeff"
```

- Инструкция **ENV** хорошо подходит для задания констант. Если вы используете некое значение в Dockerfile несколько раз, скажем, при описании команд, выполняющихся в контейнере, и подозреваете, что, возможно, вам когда-нибудь придётся сменить его на другое, его имеет смысл записать в подобную константу.

Инструкция RUN

- Инструкция **RUN** позволяет создать слой во время сборки образа. После её выполнения в образ добавляется новый слой, его состояние фиксируется. Инструкция **RUN** часто используется для установки в образы дополнительных пакетов.

```
FROM python:3.7.2-alpine3.8
LABEL maintainer="jeffmshale@gmail.com"
ENV ADMIN="jeff"
RUN apk update && apk upgrade && apk add bash
```

- В примере, инструкция RUN apk update && apk upgrade сообщает Docker о том, что системе нужно обновить пакеты из базового образа. Вслед за этими двумя командами идёт команда && apk add bash, указывающая на то, что в образ нужно установить bash.

Инструкция RUN

- То, что в командах выглядит как apk — это сокращение от Alpine Linux package manager (менеджер пакетов Alpine Linux). Если вы используете базовый образ какой-то другой ОС семейства Linux, тогда вам, например, при использовании Ubuntu, для установки пакетов может понадобиться команда вида RUN apt-get.
- Инструкция **RUN** и схожие с ней инструкции — такие, как **CMD** и **ENTRYPOINT**, могут быть использованы либо в exec-форме, либо в shell-форме.
- Exec-форма использует синтаксис, напоминающий описание JSON-массива. Например, так: RUN ["my_executable", "my_first_param1", "my_second_param2"].
- В примере мы использовали shell-форму инструкции **RUN** в таком виде: RUN apk update && apk upgrade && apk add bash.sh.

Инструкция COPY

- Инструкция **COPY** копирует в контейнер файлы и папки.

```
FROM python:3.7.2-alpine3.8
LABEL maintainer="jeffmshale@gmail.com"
ENV ADMIN="jeff"
RUN apk update && apk upgrade && apk add bash
COPY . ./app
```

- В примере инструкция **COPY** сообщает Docker о том, что нужно взять файлы и папки из локального контекста сборки и добавить их в текущую рабочую директорию образа. Если целевая директория не существует, эта инструкция её создаст.

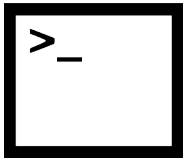
Инструкция ADD

- Инструкция **ADD** позволяет решать те же задачи, что и **COPY**, но с ней связана ещё пара вариантов использования. Так, с помощью этой инструкции можно добавлять в контейнер файлы, загруженные из удалённых источников, а также распаковывать локальные .tar-файлы.

```
...
RUN apk update && apk upgrade && apk add bash
COPY . ./app
ADD https://raw.githubusercontent.com/discdiver/pachy-vid/master/sample_vids/vid1.mp4 \
/my_app_directory
```

- В этом примере инструкция **ADD** была использована для копирования файла, доступного по URL, в директорию контейнера my_app_directory. Надо отметить, однако, что документация Docker не рекомендует использование подобных файлов, полученных по URL, так как удалить их нельзя, и так как они увеличивают размер образа.

Инструкция CMD



- Инструкция **CMD** предоставляет Docker команду, которую нужно выполнить при запуске контейнера. Результаты выполнения этой команды не добавляются в образ во время его сборки. В нашем примере с помощью этой команды запускается скрипт my_script.py во время выполнения контейнера..

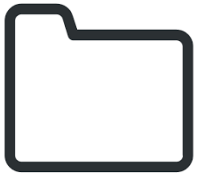
```
...
ADD https://raw.githubusercontent.com/discdiver/pachy-vid/master/sample_vids/vid1.mp4 \
/my_app_directory
RUN ["mkdir", "/a_directory"]
CMD ["python", "./my_script.py"]
```

- В одном файле Dockerfile может присутствовать лишь одна инструкция **CMD**. Если в файле есть несколько таких инструкций, система проигнорирует все кроме последней.

Пример 2 Dockerfile

```
FROM python:3.7.2-alpine3.8
LABEL maintainer="jeffmshale@gmail.com"
# Устанавливаем зависимости
RUN apk add --update git
# Задаём текущую рабочую директорию
WORKDIR /usr/src/my_app_directory
# Копируем код из локального контекста в рабочую директорию образа
COPY . .
# Задаём значение по умолчанию для переменной
ARG my_var=my_default_value
# Настраиваем команду, которая должна быть запущена в контейнере во время его выполнения
ENTRYPOINT ["python", "./app/my_script.py", "my_var"]
# Открываем порты
EXPOSE 8000
# Создаём том для хранения данных
VOLUME /my_volume
```

Инструкция WORKDIR



- Инструкция **WORKDIR** позволяет изменить рабочую директорию контейнера. С этой директорией работают инструкции **COPY**, **ADD**, **RUN**, **CMD** и **ENTRYPOINT**, идущие за **WORKDIR**. Вот некоторые особенности, касающиеся этой инструкции:
  - Лучше устанавливать с помощью **WORKDIR** абсолютные пути к папкам, а не перемещаться по файловой системе с помощью команд **cd** в Dockerfile.
  - Инструкция **WORKDIR** автоматически создаёт директорию в том случае, если она не существует.
  - Можно использовать несколько инструкций **WORKDIR**. Если таким инструкциям предоставляются относительные пути, то каждая из них меняет текущую рабочую директорию.

Инструкция ARG

- Инструкция **ARG** позволяет задать переменную, значение которой можно передать из командной строки в образ во время его сборки. Значение для переменной по умолчанию можно представить в Dockerfile. Например: ARG my_var=my_default_value.

```
...
# Задаём текущую рабочую директорию
WORKDIR /usr/src/my_app_directory
# Копируем код из локального контекста в рабочую директорию образа
COPY . .
# Задаём значение по умолчанию для переменной
ARG my_var=my_default_value
```

- В отличие от **ENV**-переменных, **ARG**-переменные недоступны во время выполнения контейнера. Однако **ARG**-переменные можно использовать для задания значений по умолчанию для **ENV**-переменных из командной строки в процессе сборки образа. А **ENV**-переменные уже будут доступны в контейнере во время его выполнения.

Инструкция ENTRYPOINT

- Инструкция **ENTRYPOINT** позволяет задавать команду с аргументами, которая должна выполняться при запуске контейнера. Она похожа на команду **CMD**, но параметры, задаваемые в **ENTRYPOINT**, не перезаписываются в том случае, если контейнер запускают с параметрами командной строки.

```
...
ARG my_var=my_default_value
# Настраиваем команду, которая должна быть запущена в контейнере во время его выполнения
ENTRYPOINT ["python", "./app/my_script.py", "my_var"]
```

- Вместо этого аргументы командной строки, передаваемые в конструкции вида docker run my_image_name, добавляются к аргументам, задаваемым инструкцией **ENTRYPOINT**. Например, после выполнения команды вида docker run my_image bash аргумент bash добавится в конец списка аргументов, заданных с помощью **ENTRYPOINT**. Готовя Dockerfile, не забудьте об инструкции CMD или ENTRYPOINT.

Инструкция ENTRYPOINT

- В документации к Docker есть несколько рекомендаций, какую инструкцию стоит выбрать **CMD** или **ENTRYPOINT**:
  - Если при каждом запуске контейнера нужно выполнять одну и ту же команду — используйте **ENTRYPOINT**.
  - Если контейнер будет использоваться в роли приложения — используйте **ENTRYPOINT**.
  - Если вы знаете, что при запуске контейнера вам понадобится передавать ему аргументы, которые могут перезаписывать аргументы, указанные в Dockerfile, используйте **CMD**.
- В нашем примере использование инструкции ENTRYPOINT ["python", "my_script.py", "my_var"] приводит к тому, что контейнер, при запуске, запускает Python-скрипт my_script.py с аргументом my_var. Значение, представленное my_var, потом можно использовать в скрипте с помощью argparse. Обратите внимание на то, что в Dockerfile переменной my_var назначено значение по умолчанию с помощью ARG. В результате, если при запуске контейнера ему не передали соответствующее значение, будет применено значение по умолчанию.

Инструкция EXPOSE

- Инструкция **EXPOSE** указывает какие порты планируется открыть для того, чтобы через них можно было связаться с работающим контейнером. Эта инструкция не открывает порты. Она играет роль документации к образу, средством общения того, кто собирает образ, и того, кто запускает контейнер.
- Для того чтобы открыть порт (или порты) и настроить перенаправление портов, нужно выполнить команду docker run с ключом -p. Если использовать ключ в виде -P (с заглавной буквой P), то открыты будут все порты, указанные в инструкции **EXPOSE**.

Инструкция VOLUME

- Инструкция **VOLUME** позволяет указать место, которое контейнер будет использовать для постоянного хранения файлов и для работы с такими файлами.
- Этим список таких инструкций не исчерпывается. В частности, мы не рассмотрели такие инструкции, как USER, ONBUILD, STOPSIGNAL, SHELL и HEALTHCHECK.
- Файлы Dockerfile — это ключевой компонент экосистемы Docker, работать с которым необходимо научиться всем.