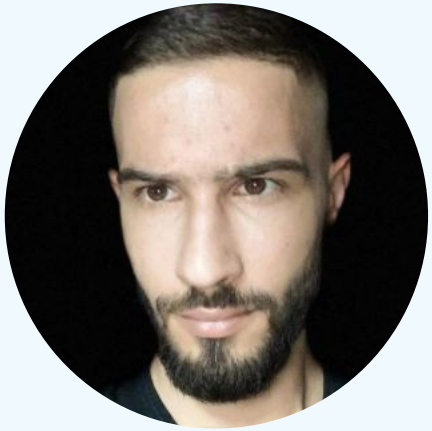
нетология

Практические навыки работы с Docker

Андрей
Щаднев



Андрей Щаднев

Senior Engineer, Tele2

Андрей Щаднев

План занятия

1. Синтаксис Docker файла
2. Сборка и публикация
3. Практическая работа
4. Итоги
5. Домашнее задание

Синтаксис Docker файла

Синтаксис Docker файла

Dockerfile - простой текстовый файл-манифест. Обычная практика называть его *Dockerfile*, но на самом деле может иметь любое имя.

Пример базового файла:

- FROM ubuntu
- RUN apt-get update
- RUN apt-get install nginx

FROM указывает на базовый образ для нашей сборки.

Каждая строка *RUN* будет выполняться Docker во время сборки.

Наши команды *RUN* не должны быть интерактивными. Во время сборки Docker не может вводить какие-либо данные.

Инструкция RUN

RUN используется:

- для выполнения команды;
- для записи изменений, внесенных в файловую систему;
- отлично подойдет для установки библиотек зависимостей, пакетов и различных файлов.

RUN не предназначен:

- для записи состояния процессов;
- для автоматического запуска демонов;
- если вы хотите, чтобы что-то запускалось автоматически при запуске контейнера, вам следует использовать CMD и/или ENTRYPOINT.

Инструкция WORKDIR

Инструкция **WORKDIR** устанавливает рабочий каталог для выполнения последующих инструкций.

Он также влияет на **CMD** и **ENTRYPOINT**, так как он устанавливает рабочий каталог, используемый при запуске контейнера и **CMD** / **ENTRYPOINT** будут искать указанный бинарный файл в нем, если не переопределено.

```
WORKDIR / src
```

Вы можете неоднократно переопределять **WORKDIR** в рамках описания манифеста, чтобы изменить рабочий каталог для дальнейших операций

Инструкции USER и COPY

Инструкция **USER** устанавливает имя пользователя или UID для использования сборке из образа и при запуске контейнера.

Этот параметр можно переопределять много раз, чтобы использовать выполнение из-под root или другого пользователя.

Инструкция **COPY** добавляет в образ файлы и данные с вашего ПК.

```
COPY . /src
```

Это добавит содержимое контекста сборки (каталог, переданный в качестве аргумента для сборки docker) в каталог /src в контейнере.

Инструкция ADD

ADD работает почти как COPY, но имеет несколько дополнительных функций.

ADD может получать файлы с удаленного хоста:
`ADD http://www.example.com/webapp.jar / opt /`

Это загрузит файл webapp.jar и поместит его в каталог / opt.

ADD автоматически распакует zip-файлы и tar-архивы:
`ADD ./assets.zip /var/www/htdocs/assets/`

Это распакует assets.zip в /var/www/htdocs/assets.

Однако **ADD** не будет автоматически распаковывать архивы с удаленного хоста.

ADD, COPY и кэш сборки

Перед созданием нового слоя Docker проверяет свой кэш сборки.

Для большинства инструкций Dockerfile Docker смотрит только на содержимое Dockerfile, чтобы выполнить поиск в кэше.

Для инструкций **ADD** и **COPY** Docker также проверяет, были ли изменены файлы, добавляемые в контейнер.

ADD всегда загружает файл, прежде чем он сможет проверить, был ли он изменен.

Инструкция ENV

Инструкция **ENV** определяет переменные окружения, которые должны быть установлены в любом контейнере, запущенном из образа.

```
ENV WEBAPP_PORT 8080
```

Это приведет к созданию переменной окружения в любых контейнерах, созданных из этого образа.

WEBAPP_PORT = 8080

Вы также можете указать переменные окружения при использовании **docker run**

```
docker run -e WEBAPP_PORT = 8000 -e WEBAPP_HOST = www.example.com
```

Инструкция EXPOSE

Инструкция **EXPOSE** сообщает Docker, какие порты должны быть открыты в этом образе.

```
EXPOSE 8080
EXPOSE 80 443
EXPOSE 53 / tcp 53 / udp
```

По умолчанию все порты закрыты.

Объявить порт с помощью **EXPOSE** недостаточно, чтобы сделать его общедоступным.

Dockerfile не контролирует, на каком порту сервис будет доступен.

CMD и ENTRYPOINT

Команды **CMD** и **ENTRYPOINT** лучше всего работают вместе.

```
ENTRYPOINT ["nginx"]
CMD ["-g",”daemon off;" ]
```

ENTRYPOINT определяет команду, которую нужно запустить, а **CMD** указывает ее параметры. Затем в командной строке мы можем при необходимости переопределить параметры.

```
docker run -d <dockerhubUsername> / image -binaryName
```

Это переопределит параметры **CMD** с помощью других флагов

Что, если мы хотим определить параметры по умолчанию для нашего контейнера? Для этого можно использовать **ENTRYPOINT** и **CMD** вместе.

ENTRYPOINT определит базовую команду для нашего контейнера. **CMD** определит параметры по умолчанию для этой команды. Обе инструкции используют синтаксис JSON.

Дополнительные инструкции в Dockerfile

ONBUILD - позволяет хранить инструкции, которые будут выполняться, когда этот образ используется в качестве базового для другого образа.

LABEL - добавляет к образу метаданные.

ARG - определяет переменные окружения на время сборки (необязательные или обязательные).

STOPSIGNAL - устанавливает сигнал для остановки докера (по умолчанию TERM).

HEALTHCHECK - определяет команду, оценивающую состояние контейнера.

SHELL - устанавливает программу по умолчанию, которая будет использоваться для интерпретации строкового синтаксиса RUN, CMD и т.д.

Пример корректного манифеста Dockerfile

Добавление зависимостей в качестве отдельного шага означает, что Docker может более эффективно кэшировать и устанавливать их только при изменении параметров.

```
FROM python
COPY ./requirements /tmp/requirements
RUN pip install -qr /tmp/requirements
COPY . /src/
WORKDIR /src
EXPOSE 5000
CMD ["python", "app.py"]
```

Свойства корректного манифеста Dockerfile

- Малое количество слоев.
- Используется кэш сборки, для прироста скорости сборки.
- Используется юнит тестирование в процессе сборки.

Сборка и публикация

TAG, BUILD и PUSH

Собрать образ из манифеста:
`docker build -t yourapp .`

Где “.” - это билд контекст и данном случае текущая директория

Указать другое местонахождение Dockerfile:
`docker build -t yourapp –f /path/to/dockerfile .`

Добавить тег для образа:
`docker tag yourapp yourrepo/yourapp`

Публикация образа в DockerHub / artifactory / локальном репозитории:
`docker push yourrepo/yourapp`

Теперь любой желающий может запускать `yourrepo/yourapp` где угодно из DockerHub или из внутреннего корпоративного репозитория.

Практическая работа

Описание задач

Воссоздаем функциональность и запускаем dockerfile с использованием другого базового образа.

Пример dockerfile:
https://hub.docker.com/r/mpepping/ponysay/dockerfile

Базовый статический веб-сайт в рабочем состоянии.

Задача 1

```
FROM ubuntu:latest

RUN apt-get update && apt-get install -y software-properties-common &&
add-apt-repository ppa:vincent-c/ponysay && apt-get update
RUN apt-get install -y ponysay

ENTRYPOINT ["/usr/bin/ponysay"]
CMD ["Hey, guys"]
```

Задача 1

```
FROM ubuntu:latest

RUN adduser pony
RUN apt-get update && apt-get install -y software-properties-common &&
add-apt-repository ppa:vincent-c/ponysay && apt-get update
RUN apt-get install -y ponysay

COPY ./pony.sh /app/pony.sh

USER pony
ENTRYPOINT ["bash"]
CMD ["/app/pony.sh"]
```

Задача 1 - скрипт для ENTRYPOINT

```
#!/bin/bash
user=`whoami`
if [ $user == "root" ]
then
        ponysay "Damn.. looks like I'm $user"
else
        ponysay "Hey, I'm $user"
fi
```

Задача 2

Вариант 1

```
FROM nginx:alpine

COPY ./index.html /usr/share/nginx/html/index.html

COPY ./favicon.ico /usr/share/nginx/html/favicon.ico
```

Вариант 2

```
FROM nginx:alpine

COPY ./staticsite /usr/share/nginx/html/
```

Итоги

Итоги

- Мы ознакомились с синтаксисом Dockerfile и принципами корректного использования основных инструкций.
- Научились собирать и публиковать образ.
- Реализовали манифесты Dockerfile и собрали соответствующие образы по практическим задачам.

Домашнее задание

Давайте посмотрим ваше домашнее задание.

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

нетология

Задавайте вопросы и пишите отзыв о лекции!

Андрей Щаднев

Андрей Щаднев