Dockerfile

DockerFile - набор инструкций по установке и настройке контейнера. Также используется для создания преднастроенных образов. Обычно dockerfile расположен в директории, из которой вызывается команда.
Имя файла: Dockerfile (без расширения).
Контекст создания - набор локальных файлов/каталогов, к которым можно обращаться через copy/add.
/ не стоит использовать как контекст, полностью включается в архив
комментарии
При сборке образа попытки использования кэша идут до первого промаха. Затем кэш не проверяется.
.dockerignore - файл исключений из контекста
При сборке компилируемых приложений лучше использовать multi-stage build.

Основная команда	Параметры	Описание
docker build		создание образа Создание образа из docker файла . - контекст создания (текущая папка) `docker build -t test/cowsay-dockerfile .` После этого образ появляется в списке образов
	-t	имя слоя
	-f	расположение слоя
	--squash	сжимает все слои в один
docker history	<image>	инструкции создания образа
docker buildx		для компиляции под разные платформы

Принципы

- Нужно создавать эфемерные контейнеры без данных внутри. Переменные настраиваются потом.
- Контекст сборки - директория размещения Dockerfile.
- Есть .dockerignore аналогичен .gitignore
- Не устанавливать ненужные пакеты
- Использовать кэш сборки

Основные правила создания контейнера

- Не хранить данные / учетные данные внутри контейнера
- Не дробить поставку приложения
- Не создавать большие образы
- Не делать однослойные контейнеры
- Не создавать образы из запущенных контейнеров
- Не надо использовать только тег latest
- Не выполнять в контейнере более одного процесса
- Не запускать от имени root
- Не надо использовать явную связь с IP

Формат exec и формат командной оболочки

Формат exec

JSON-массив
первый элемент массива: имя выполняемого файла
остальные - параметры, передаваемые при запуске.

Формат командной оболочки

строка произвольной формы, передаваемая для интерпретации в /bin/sh -c.
Используйте формат exec, чтобы избежать случайного искажения строк командной оболочки, или в тех случаях, когда образ не содержит /bin/sh.
Команды файла:

Основная команда	Параметры	Описание
FROM	<name>	название базового образа
MAINTAINER	<name>	имя поддерживающего пользователя
USER	<name>	!Всегда определять! Задает пользователя (по имени или по идентификатору UID) для использования во всех последующих инструкциях RUN, CMD, ENTRYPOINT.
WORKDIR		рабочий каталог для последующих RUN, CMD, ENTRYPOINT, ADD, COPY. можно использовать несколько раз можно относительные пути, итоговый путь относительно предыдущего WORKDIR.
RUN	<commands>	команды при инициализации образа, обычно установка пакетов каждая создает новый слой
CMD {''}		команда с аргументами, выполняемая после запуска контейнера. Аргументы могут быть переопределены при запуске контейнера. В файле может присутствовать лишь одна инструкция CMD.

Основная команда	Параметры	Описание
ENTRYPOINT	<program name>	Выполняемый файл, который будет вызываться для обработки аргументов, переданных в команду docker run `ENTRYPOINT ["/usr/games/cowsay", "-k"]`
COPY	<> <>	копирование файла из ФС ОС в ФС образа COPY . /src копирует все из текущей папки в папку /src
VOLUME	<>	Том в ФС
ARG		Определяет переменные среды, доступные внутри образа при сборке. Неизменяемые.

Основная команда	Параметры	Описание

ENV		Определяет переменные среды внутри образа, но могут изменяться. На эти переменные можно ссылаться в последующих инструкциях.*
		`ENV MY_VERSION 1.3`
		`RUN apt-get install -y mypackage=$MY_VERSION`
		
		Переменные среды со значением по умолчанию
		Файл first.sh
		`echo $NODE_ENV`
		
		Dockerfile
		`FROM alpine:latest`
		`ARG NODE_ENV=production2`
		`ENV NODE_ENV=${NODE_ENV}`
		`RUN mkdir /var/www`
		`COPY first.sh /var/www/first.sh`
		`CMD ["sh", "/var/www/first.sh"]`
		
		Собираем образ
		`docker build -t bobrobot:1.0 .`
		
		docker-compose.yml
		`services:`
		`  bobrobot:`
		`  image: bobrobot:1.0`
		`  environment:`
		`  NODE_ENV: "first2"`
		
		В итоге при указании NODE_ENV выводится first2, иначе production2

Основная команда	Параметры	Описание
EXPOSE		Сообщает механизму Docker, что в контейнере будет процесс, прослушивающий порт(ы) не оказывает воздействия на сетевую среду нужно для аргумента -p в docker run
ONBUILD		инструкция, выполняемая когда образ будет использоваться как основной уровень для другого образа. Полезным при обработке данных, добавляемых в образ-потомок (например, инструкция копирования дополнительного кода из заданного каталога и запуска скрипта сборки, обрабатывающего скопированные данные).

* Для различных образов могут использоваться доп. переменные, иногда обязательные. Детали нужно уточнять для конкретного образа.
При сборке команды исполняются однократно. Затем при запуске образа они не повторяют исполнение. Поэтому для

```
FROM alpine:latest
ARG NODE_ENV=production2
ENV NODE_ENV=${NODE_ENV}
RUN mkdir /var/www
RUN echo $NODE_ENV > /var/www/first.txt
CMD ["cat", "/var/www/first.txt"]
```

вывод всегда будет production2.

Многошаговая сборка (Multi-stage building)

Создание одного образа с участием нескольких временных

Необходима если нужно при помощи временного образа скомпилировать исходники и результат сохранить в финальный образ. Временный образ удаляется.

```
FROM golang:1.20-alpine AS base
WORKDIR /src
COPY go.mod go.sum .
RUN go mod download
COPY . .

FROM base AS build-client
RUN go build -o /bin/client ./cmd/client
```

```
FROM base AS build-server
RUN go build -o /bin/server ./cmd/server

FROM scratch AS prod
COPY --from=build-client /bin/client /bin/
COPY --from=build-server /bin/server /bin/
ENTRYPOINT [ "/bin/server" ]
```

Сборка образа:

```
docker build -t multi:stage .
```

Создание нескольких образов

Также можно создать несколько образов при помощи одного Dockerfile. Отличие в последней стадии:

```
...
FROM scratch AS prod-client
COPY --from=build-client /bin/client /bin/
ENTRYPOINT [ "/bin/client" ]

FROM scratch AS prod-server
COPY --from=build-server /bin/server /bin/
ENTRYPOINT [ "/bin/server" ]
```

Сборка образов:

```
docker build -t multi:client --target prod-client -f Dockerfile-final .
docker build -t multi:server --target prod-server -f Dockerfile-final .
```

Мультиплатформенная сборка

Возможно, но пока не интересно

Использование условий для создания разных образов

Условия можно использовать только в конструкции RUN и CMD

```
RUN if [ "$TARGET" = "test" ]; then \
      cp -r /files/src/tests .; \
    fi
```

```
# Разные команды для production и тестов
CMD if [ "$TARGET" = "production" ]; then \
      exec gunicorn app.main:app --bind 0.0.0.0:8000; \
    else \
      exec pytest -v; \
    fi
```

Затем собирается образ с указанной переменной окружения

```
docker build --build-arg TARGET=production -t myapp:prod .
```

Полный пример:

```
FROM alpine:latest
ARG BUILD_ENV
RUN mkdir /var/www
WORKDIR /var/www
COPY test.txt .
COPY prod.txt .
RUN if [ "$BUILD_ENV" = "test" ]; then \
       cat test.txt > first.txt; \
    else \
       cat prod.txt > first.txt; \
    fi
CMD ["cat", "/var/www/first.txt"]
```

Использование multistage сборки для ограничения копирования

if не работает в команде COPY. Но иногда бывает нужен образ для тестирования без содержимого директории, а в prod с содержимым. Пример:

```
# Общая стадия для подготовки файлов
FROM alpine:latest as base
COPY common_files /app/common_files

# Стадия для prod (копирует папку)
FROM base as prod
COPY prod_folder /app/prod_folder

# Стадия для test (не копирует папку)
```

```
FROM base as test
RUN echo "Running in test mode, no prod_folder copied"

# Выбираем финальную стадию через ARG
ARG TARGET_ENV=prod
FROM ${TARGET_ENV} as final

# Остальные инструкции...
```

Сжатие итогового образа (многоуровневого) в один уровень

Не рекомендуется, но если надо: --squash флаг при build

Разное

При apt-get install использовать no-install-recommends - сильно снижает объем.

Revision #15
Created 30 June 2024 04:48:55 by Admin
Updated 30 May 2026 16:29:10 by Admin