Архитектура вычислительных систем

Контейнеризация

Романюта Алексей Андреевич

alexey-r.98@yandex.ru

Кафедра вычислительных систем
Сибирский государственный университет телекоммуникаций и информатики

Контейнеризация

- Контейнеры позволяют поместить выполнение процесса в «песочницу». При этом изолировав его от других процессов в различных аспектах
- Для изоляции используется в основном механизм linux namespaces
- Изолируются:
  - Сеть – network namespace
  - Межпросессное взаимодействие – ipc namespace
  - Файловая система – overlayfs, mount
  - User namespace – представление id пользователей со смещением с целью избежать пересечения с хост-системой, root в контейнере != root в системе
  - Uts namespace - позволяют устанавливать имена хостов и домены, не влияя на хост систему

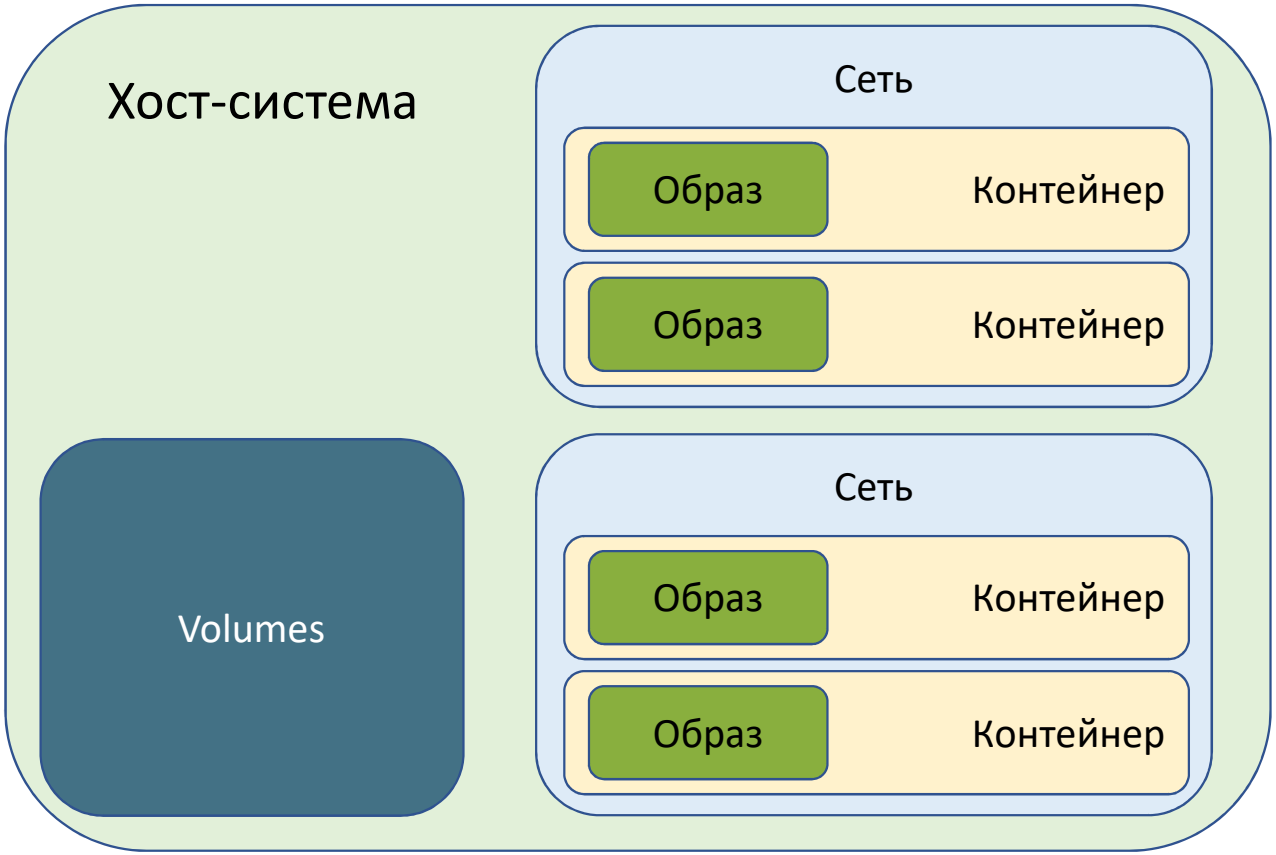
Контейнеризация

Инструменты

- Docker
- Podman
- LXC
- Containerd
- Cri-o
- runC

Docker

- Основные сущности/компоненты docker:
  - Контейнер
  - Сети (Network)
  - Volumes
  - Образы (Images)



Docker

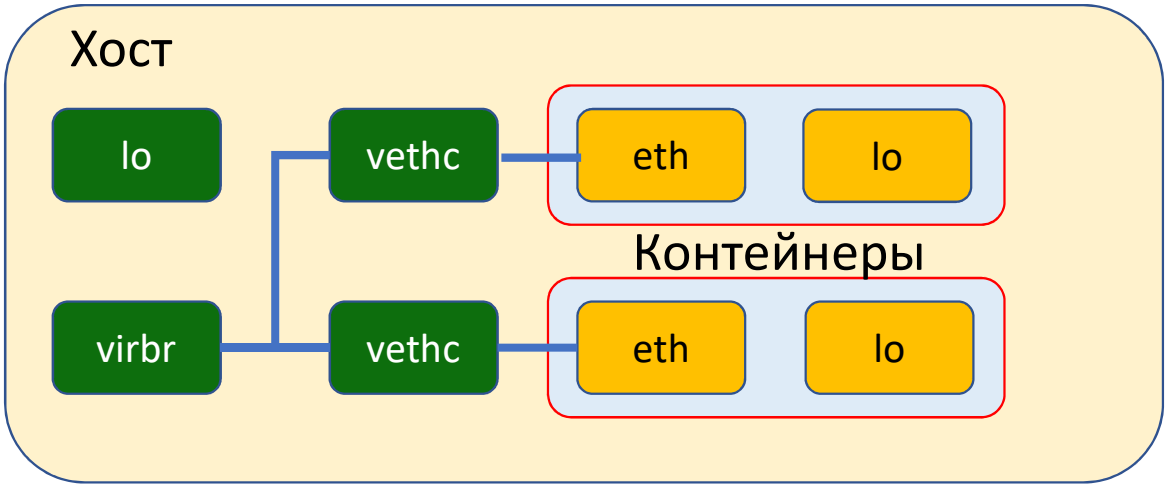
- Docker предоставляет api для создания образа и запуска контейнера в системе
- `docker create` – подготовка и создание контейнера. Принимает в качестве аргументов имя *образа* контейнера. Контейнер находится в статусе created и не запускается до специальной команды
- `docker start` – запуск контейнера. Принимает в качестве аргументов id/имя контейнера. Работает только на существующих контейнерах
- `docker run` – создание и запуск контейнера. Принимает в качестве аргументов имя *образа* контейнера. (create + run)
- `docker restart` – перезапуск контейнера. Принимает в качестве аргументов id/имя контейнера.

Docker

- `docker rm ${container_id}` – удаление контейнера. Для удаления образов, сетей, хранилищ используется аналогичная команда в соответствующем подменю. (Например – `docker image rm`)
- Для удаления запущенного контейнера, его необходимо остановить командой `docker stop ${container_id}`, где `container_id` – имя или идентификатор контейнера. Альтернативным вариантом является использование ключа *–f*
- `docker ${type} prune` – удаление неиспользуемых сущностей ${type}, где type – соответствующий тип, например - container, image, network. С помощью `docker system prune` возможно удалить все неиспользуемые сущности
- При вызове удаления хранилищ (volumes) стоит учитывать, что удалению подлежат *все неиспользуемые* объекты
- `docker system df` – позволяет примерно оценить занимаемое сущностями docker место

Docker

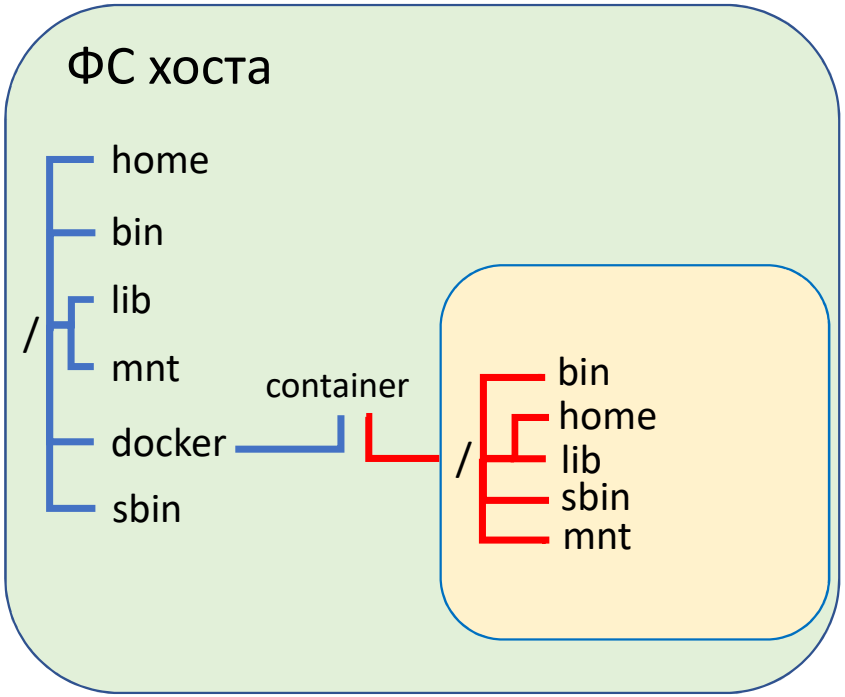
Изоляция сети и файловой системы



Каждый eth-интерфейс внутри контейнера связан с виртуальным интерфейсом хост-систем
Для контейнеров, работающих в одной виртуальной сети создается bridge-интерфейс

Network namespace

Первый подход - chroot



Docker

- Изоляция сети на уровне iptables состоит из двух этапов
  - Проверка, что контейнер отправляет трафик не самому себе/контейнерам в той же сети
    `intin != intout`
  - Проверка, что контейнер отправляет трафик не другим контейнерам. Предполагается, что контейнеры находятся в других docker-сетях
    `intout` **`not in`** `container_interfaces`

- Изоляцию сети можно отключить.
  Параметр «*--network host*» для docker или «*network_mode: host*» для docker-compose. В таком случае контейнер получает доступ ко всем интерфейсам хоста.
- Изоляцию сети не отключить, если docker запущен в режиме *userns-remap*

Docker

```
$ lsns | grep nginx
4026532248 mnt         13 1852554 root             nginx: master process nginx -g daemon off;
4026532249 uts         13 1852554 root             nginx: master process nginx -g daemon off;
4026532259 ipc         13 1852554 root             nginx: master process nginx -g daemon off;
4026532260 pid         13 1852554 root             nginx: master process nginx -g daemon off;
4026532267 mnt         13  233566 root             nginx: master process nginx -g daemon off;
4026532268 net         13 1852554 root             nginx: master process nginx -g daemon off;
4026532269 uts         13  233566 root             nginx: master process nginx -g daemon off;
4026532282 ipc         13  233566 root             nginx: master process nginx -g daemon off;
4026532283 pid         13  233566 root             nginx: master process nginx -g daemon off;
4026532295 net         13  233566 root             nginx: master process nginx -g daemon off;
4026533093 cgroup      13 1852554 root             nginx: master process nginx -g daemon off;
4026533325 cgroup      13  233566 root             nginx: master process nginx -g daemon off;
```

Docker

- Контейнеры используют *stateless* подход к работе. Все файлы и данные, созданные внутри контейнера существуют пока существует контейнер

- Сущность docker-volume, позволяет создать хранилище данных, независимое от существования контейнера

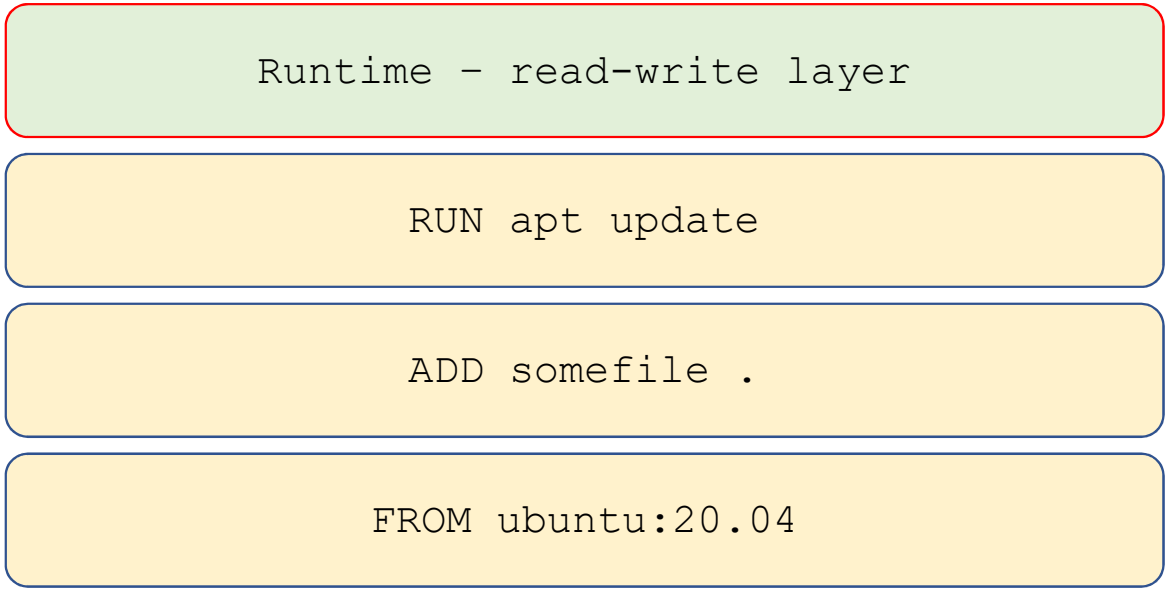
- Помимо docker-volume в контейнер можно монтировать сетевой ресурс или файлы хост-машины. Такой ресурс является сущностью volume, которая не управляется демоном docker. На такой ресурс не распространяются команды *prune*

Docker

- При сборке итоговый образ состоит из нескольких «слоев»
- Каждый слой – одна инструкция в Dockerfile
- Не все команды создают слои, только те, что модифицируют файлы

Dockerfile:

```
FROM ubuntu:20.04
ADD somefile .
RUN apt update
```



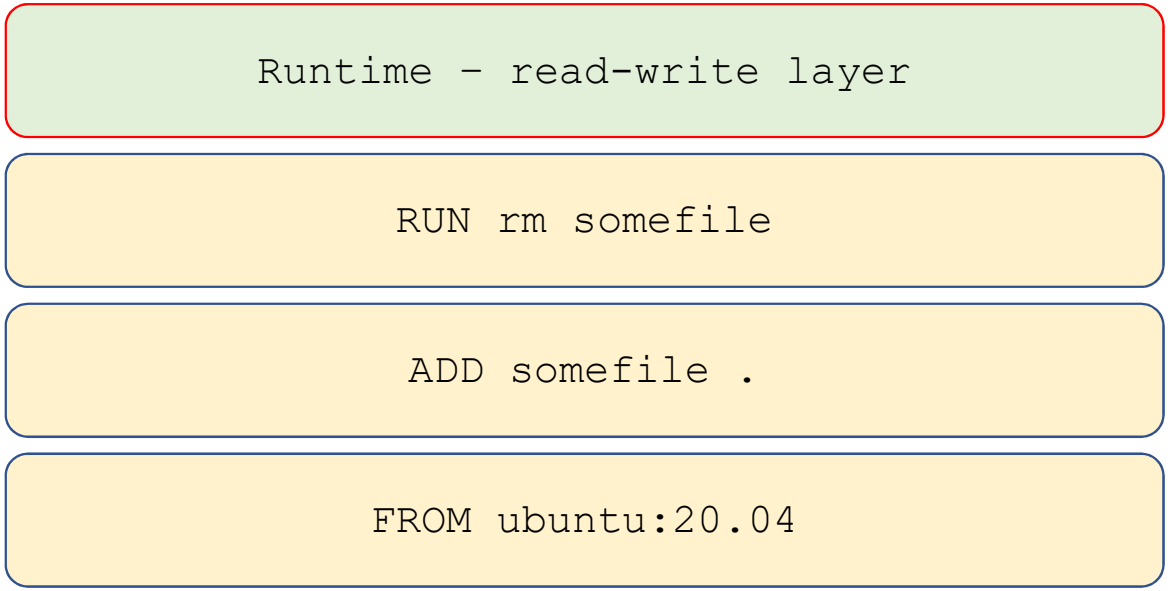
Для работы каждого слоя используется Overlayfs

Docker

- Файлы, добавленные в одном слое, полностью удалить можно только в том же слое
- Если удалить в последующих – останется как история изменений

Dockerfile:

```
FROM ubuntu:20.04
ADD somefile .
RUN rm somefile
```

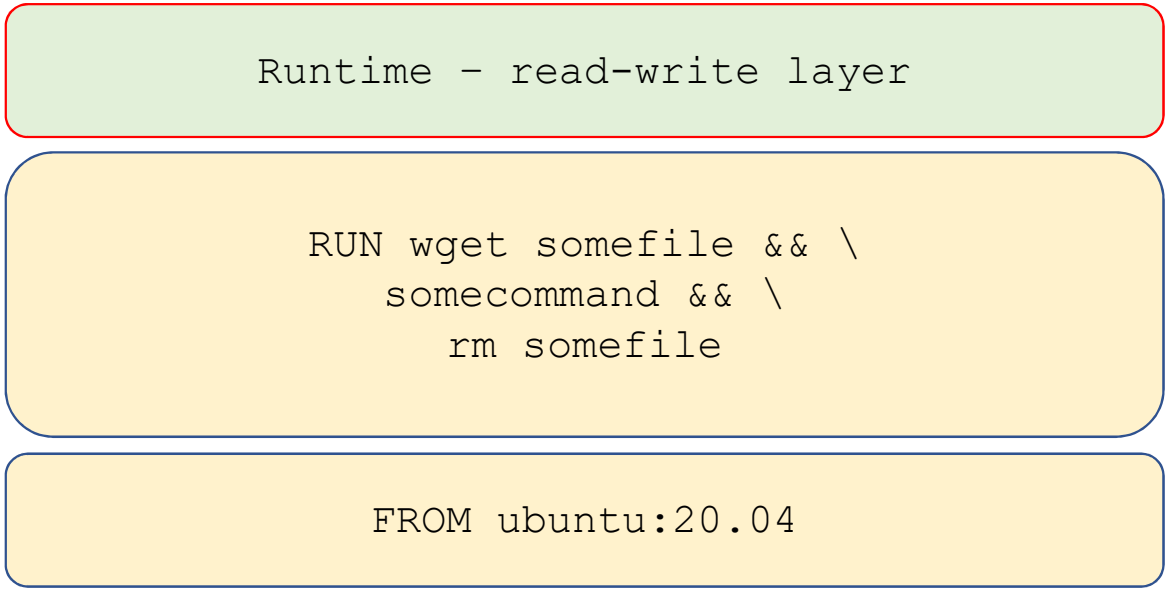


Docker

- Кэш пакетного менеджера так же можно удалить в процессе создания слоя образа (`apt update && apt install && apt clean`)

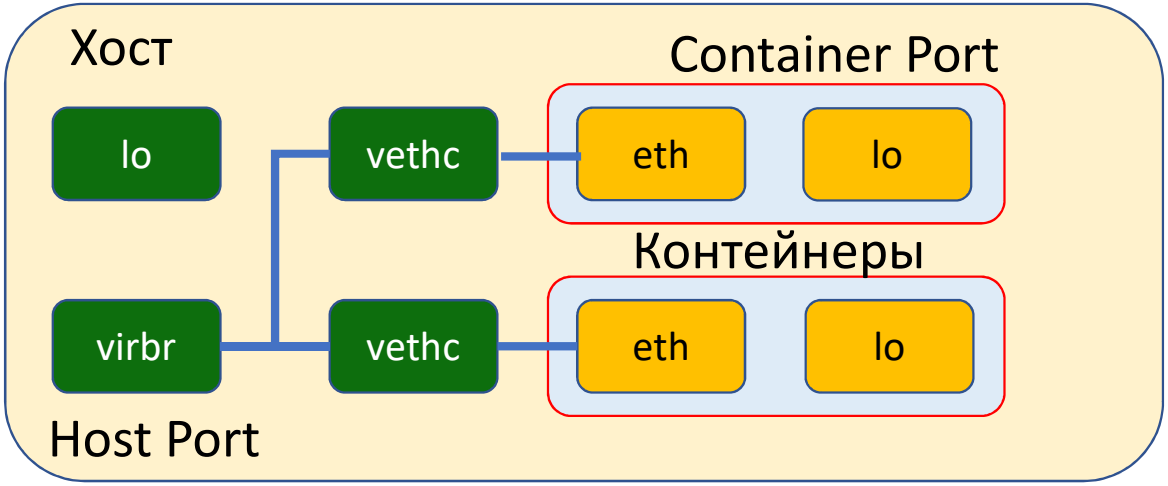
Dockerfile:

```
FROM ubuntu:20.04
RUN wget somefile && \
somecommand && \
rm somefile
```



Docker

- При запуске контейнера, для обеспечения сетевого доступа к приложению, через аргумент `-p` команды `run` можно указать порт, по которому можно обратиться к приложению. Записывается это как `-p HOST_PORT:CONTAINER_PORT`. При этом сеть контейнера все еще остается изолированной.



Docker

- Docker-container предоставляет полностью работоспособное приложение без необходимости его прямой установки. В некоторых stateless приложениях легко обновить версию, изменив тег используемого образа

```
docker run --rm -ti
-v /srv/proxy/etc/letsencrypt:/etc/letsencrypt
-v /srv/proxy/letsencrypt-site:/data/letsencrypt
certbot/certbot:v1.26.0 certonly
-d DOMAIN.DOMAIN
--webroot
-w /data/letsencrypt
--cert-path /etc/letsencrypt/live/DOMAIN.DOMAIN/cert.pem
--key-path /etc/letsencrypt/live/DOMAIN.DOMAIN/privkey.pem
--fullchain-path /etc/letsencrypt/live/DOMAIN.DOMAIN/fullchain.pem
--chain-path /etc/letsencrypt/live/DOMAIN.DOMAIN/chain.pem
-m admin@domain.com && docker kill -s HUP nginx
```

Пример выполнения команды обновления letsencrypt сертификатов

Docker

- Для создания образа контейнера используется Dockerfile
- Dockerfile – файл с описанием инструкций пошагового создания образа
- Инструкции Dockerfile.
  - FROM
  - ADD
  - COPY
  - WORKDIR
  - RUN
  - CMD
  - ENTRYPOINT
  - ENV
  - ARG
  - EXPOSE
  - VOLUME
  - LABEL
  - USER
  - ONBUILD
  - HEALTHCHECK
  - STOPSIGNAL
  - SHELL

Dockerfile: FROM

- Инструкция *FROM* предназначена для указания базового образа, который будет основой собираемого образа
- Существует специальный базовый образ *scratch* – Образ, не содержащий файлов

Dockerfile:

```
FROM ubuntu:20.04
```

Собрав и запустив образ, получаем файловую систему Ubuntu-20.04

```
$ docker build -t acs -f Dockerfile .
[+] Building 1.1s (5/5) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 212B
 => [internal] load metadata for docker.io/library/ubuntu:20.04
 => [internal] load .dockerignore
 => => transferring context: 2B
 => CACHED [1/1] FROM docker.io/library/ubuntu:20.04@sha256:0b897358ff6624825fb50d20ffb605ab0
 => exporting to image
 => => exporting layers
 => => writing image sha256:e679ba31225b1b1a2f99f2845098dbd9a8a451fc3cbbfad8441a99f3f84e4a27
 => => naming to docker.io/library/acs
$ docker run -ti --rm acs cat /etc/os-release
NAME="Ubuntu"
VERSION="20.04.6 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.6 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
```

Dockerfile: ADD, COPY , WORKDIR

- Инструкция *ADD*/*COPY* предназначена для добавления файлов в контейнер
  ADD позволяет не только копировать файлы, но и разархивировать и скачивать с remote-url
- Разместим пустой файл t.txt в папке с Dockerfile

```
COPY t.txt .
```

Копирует файл *t.txt* в рабочую директорию
Рабочая директория изначально / Может быть переопределена инструкцией *WORKDIR* (*WORKDIR* создает папку, если она не существует)

Dockerfile:

```
FROM ubuntu:20.04
ADD [Link] .
WORKDIR /app
COPY t.txt .
```

[Link] Readme проекта OpenMPI // https://raw.githubusercontent.com/open-mpi/ompi/main/README.md

Dockerfile: ADD, COPY , WORKDIR

```
$ docker build -t acs -f Dockerfile .
[+] Building 1.3s (10/10) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 209B
 => [internal] load metadata for docker.io/library/ubuntu:20.04
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [2/4] ADD https://raw.githubusercontent.com/open-mpi/ompi/main/README.md .
 => [internal] load build context
 => => transferring context: 85B
 => [1/4] FROM docker.io/library/ubuntu:20.04@sha256:0b897358ff6624825fb50d20ffb605ab0eaea77ced0adb8c6a4b756513dec6fc
 => CACHED [2/4] ADD https://raw.githubusercontent.com/open-mpi/ompi/main/README.md .
 => CACHED [3/4] WORKDIR /app
 => CACHED [4/4] COPY t.txt .
 => exporting to image
 => => exporting layers
 => => writing image sha256:0fe3bffc693e598dc594e58bf8d72a37a9addeeb37ff9da9e1d07f2b30fa80de
 => => naming to docker.io/library/acs
$ docker run -ti --rm acs bash
root@890b863d146a:/app# ls -l t.txt
-rw-r--r--. 1 root root 5 Jul 19 00:45 t.txt
root@890b863d146a:/app# head -n 5 /README.md
# Open MPI

[The Open MPI Project](https://www.open-mpi.org/) is an open source
implementation of the [Message Passing Interface (MPI)
specification](https://www.mpi-forum.org/docs/) that is developed and
```

Рабочая директория при запуске контейнера - /app

COPY t.txt .
Копирует файл *t.txt* в рабочую директорию
Рабочая директория изначально /
Может быть переопределена инструкцией *WORKDIR* (*WORKDIR* создает папку, если она не существует)

Dockerfile:

```
FROM ubuntu:20.04
ADD [Link] .
WORKDIR /app
COPY t.txt .
```

[Link] Readme проекта OpenMPI // https://raw.githubusercontent.com/open-mpi/ompi/main/README.md

Dockerfile: ADD, COPY , WORKDIR

- Используя инструкции *FROM*, *ADD* может быть собран базовый образ системы (ubuntu-focal-oci-amd64-root.tar.gz – архив файловой системы)

Dockerfile образа ubuntu:20.04

Dockerfile[Link]:

```
FROM scratch
ADD ubuntu-focal-oci-amd64-root.tar.gz /
```

[Link] Dockerfile ubuntu // https://github.com/tianon/docker-brew-ubuntu-core/blob/5ed9ebdc8e9050d5b327c7ce3cda08b62cd28f67/focal/Dockerfile

Dockerfile: RUN

- Инструкция *RUN* предназначена для запуска команд внутри образа в процессе его сборки
- Инструкция *RUN* используется для запуска команд при сборке. В качестве команд могут выступать, например, команды сборки (такие как `npm run build`) или установки пакетов (`apt update` **`and`** `apt install`)
- Базовый образ не содержит индекс пакетов – требуется обновление индекса при сборке
- Каждая инструкция *RUN* создает *слой* – по возможности команды объединяют в *пайплайн* через `&&, ||`

```
Dockerfile:
FROM ubuntu:24.04
ADD [Link] .
WORKDIR /app
COPY t.txt .
RUN apt update && apt
install -y -q git
RUN git version
```

Dockerfile: RUN

```
$ docker build -t acs -f Dockerfile .
[+] Building 26.8s (12/12) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 267B
 => [internal] load metadata for docker.io/library/ubuntu:24.04
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/6] FROM docker.io/library/ubuntu:24.04@sha256:2e863c44b718727c860746568e1d54afd13b2fa71b
 => [internal] load build context
 => => transferring context: 85B
 => [2/6] ADD https://raw.githubusercontent.com/open-mpi/ompi/main/README.md .
 => CACHED [2/6] ADD https://raw.githubusercontent.com/open-mpi/ompi/main/README.md .
 => CACHED [3/6] WORKDIR /app
 => CACHED [4/6] COPY t.txt .
 => [5/6] RUN apt update && apt install -y -q git
 => [6/6] RUN git version
 => exporting to image
 => => exporting layers
 => => writing image sha256:e23803e9e0cb53527da91495d2602124b7ce158f5b296213c8380f723b2db9b3
 => => naming to docker.io/library/acs
$ docker run -ti --rm acs git
usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--config-env=<name>=<envvar>] <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone     Clone a repository into a new directory
   init      Create an empty Git repository or reinitialize an existing one
```

```
Dockerfile:
FROM ubuntu:24.04
ADD [Link] .
WORKDIR /app
COPY t.txt .
RUN apt update && apt
install -y -q git
RUN git version
```

Собранный образ имеет установленный git

Dockerfile: CMD, ENTRYPOINT

```
$ docker run -ti --rm acs
git version 2.43.0
```

Запуск контейнера без аргументов выполняет команду, записанную в ENTRYPOINT и CMD

```
$ docker run -ti --rm acs help
usage: git [-v | --version...
```

При указании команды изменяется только значение CMD

Dockerfile:

```
FROM ubuntu:24.04
ADD [Link] .
WORKDIR /app
COPY t.txt .
RUN apt update && apt install
-y -q git
RUN git version
ENTRYPOINT [ "/usr/bin/git" ]
CMD [ "version" ]
```

Dockerfile: CMD, ENTRYPOINT

```
$ docker run -ti --rm --entrypoint /bin/ls acs -l
total 4
-rw-r--r--. 1 root root 5 Jul 19 00:45 t.txt
```

Переопределить можно оба значения – entrypoint и cmd

```
$ docker run -ti --rm --entrypoint /bin/ls acs
t.txt
```

Если переопределить только entrypoint, значение cmd будет пустым

Dockerfile:

```
FROM ubuntu:24.04
ADD [Link] .
WORKDIR /app
COPY t.txt .
RUN apt update && apt install
-y -q git
RUN git version
ENTRYPOINT [ "/usr/bin/git" ]
CMD [ "version" ]
```

Dockerfile: CMD, ENTRYPOINT

- Почему команда в ENTRYPOINT/CMD указана в ‘[ ]’ ?

Dockerfile:

```
FROM ubuntu:24.04
ADD [Link] .
WORKDIR /app
COPY t.txt .
RUN apt update && apt install
-y -q git
RUN git version
ENTRYPOINT [ "/usr/bin/git" ]
CMD [ "version" ]
```

Dockerfile: CMD, ENTRYPOINT

- Почему команда в ENTRYPOINT/CMD указана в ‘[ ]’ ?

```
$ docker run -ti --rm -d --name acs acs
a95c237b965339f8f05766150c3df6485c24abe3fff45f301ac28668b78144c9
$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS     NAMES
a95c237b9653   acs       "/bin/sh -c \"cat\""   3 seconds ago   Up 2 seconds             acs
```

Dockerfile1: При запуске контейнера, команду выполняет командная оболочка `/bin/sh`

Dockerfile1:

```
FROM ubuntu:24.04
ENTRYPOINT "cat"
```

Dockerfile2:

```
FROM ubuntu:24.04
ENTRYPOINT [ "cat" ]
```

Dockerfile: CMD, ENTRYPOINT

- Почему команда в ENTRYPOINT/CMD указана в ‘[ ]’ ?

```
$ docker run -ti --rm -d --name acs acs
a95c237b965339f8f05766150c3df6485c24abe3fff45f301ac28668b78144c9
$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS     NAMES
a95c237b9653   acs       "/bin/sh -c \"cat\""   3 seconds ago   Up 2 seconds             acs
```

Dockerfile1: При запуске контейнера, команду выполняет командная оболочка `/bin/sh`

```
$ docker run -ti --rm -d --name acs acs
3afe6ec4b6907ca6e99bf4e2a8c78b4eea249ca1836a6aca011fa829e432af6b
$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED         STATUS         PORTS     NAMES
3afe6ec4b690   acs       "cat"     2 seconds ago   Up 1 second              acs
```

Dockerfile2: При запуске контейнера, команда выполняется непосредственно как исполняемый файл

Dockerfile1:

```
FROM ubuntu:24.04
ENTRYPOINT "cat"
```

Dockerfile2:

```
FROM ubuntu:24.04
ENTRYPOINT [ "cat" ]
```

Dockerfile: ENV

- Инструкция ENV позволяет задать переменные среды окружения, определяемые при работе контейнера
- Переменные среды доступны в процессе сборки в нижестоящих слоях
- Могут быть переопределены при запуске контейнера

Dockerfile:

```
FROM ubuntu:24.04
ADD [Link] .
ENV APP_DIR=/app
WORKDIR ${APP_DIR}
COPY t.txt .
RUN apt update && apt install
-y -q git
RUN git version
ENTRYPOINT [ "/usr/bin/git" ]
CMD [ "version" ]
```

Dockerfile: ENV

```
$ docker run -ti --rm --entrypoint env acs
PATH=/usr/local/bin:/bin
HOSTNAME=0424e4324fad
TERM=xterm
APP_DIR=/app
HOME=/root
```

При запуске контейнера определена переменная APP_DIR
При сборке инструкция WORKDIR принимает значение /app

Dockerfile:

```
FROM ubuntu:24.04
ADD [Link] .
ENV APP_DIR=/app
WORKDIR ${APP_DIR}
COPY t.txt .
RUN apt update && apt install
-y -q git
RUN git version
ENTRYPOINT [ "/usr/bin/git" ]
CMD [ "version" ]
```

Dockerfile: ARG

- Инструкция ARG позволяет задать переменные среды окружения, определяемые при сборке контейнера
- Аргументы сборки доступны в процессе сборки в нижестоящих слоях
- Могут быть переопределены при запуске *сборки* контейнера
- Не определяются при запуске контейнера

Dockerfile:

```
FROM ubuntu:24.04
ADD [Link] .
ENV APP_DIR=/app
WORKDIR ${APP_DIR}
ARG TARGET_FILE=t.txt
COPY ${TARGET_FILE} .
RUN apt update && apt install
-y -q git
RUN git version
ENTRYPOINT [ "/usr/bin/git" ]
CMD [ "version" ]
```

Dockerfile: ARG

```
$ docker run --entrypoint ls acs
Dockerfile1
```

При сборке определен аргумент TARGET_FILE=Dockerfile1 [1]

```
Dockerfile:
FROM ubuntu:24.04
ADD [Link] .
ENV APP_DIR=/app
WORKDIR ${APP_DIR}
ARG TARGET_FILE=t.txt
COPY ${TARGET_FILE} .
RUN apt update && apt install
-y -q git
RUN git version
ENTRYPOINT [ "/usr/bin/git" ]
CMD [ "version" ]
```

[1] Сборка образа с аргументом // docker build --build-arg TARGET_FILE=Dockerfile1 -t acs .

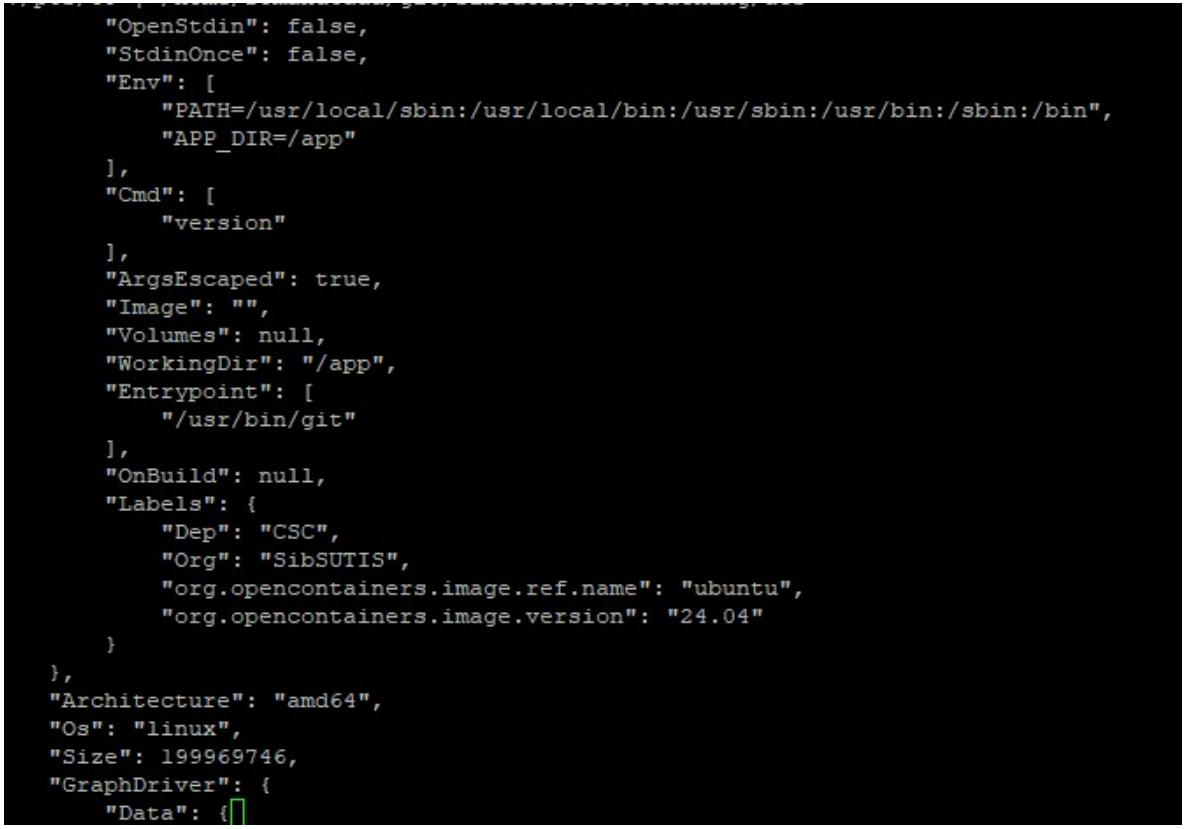
Dockerfile: LABEL

- Инструкция LABEL позволяет задать метаданные образа
- Обычно используется для указания автора, версий
  и т.п.
- Не влияют на процесс сборки и запуск (кроме создания слоя сборки)
- Инструкция MAINTAINER помечена как deprecated

Dockerfile:

```
FROM ubuntu:24.04
ADD [Link] .
ENV APP_DIR=/app
WORKDIR ${APP_DIR}
ARG TARGET_FILE=t.txt
COPY ${TARGET_FILE} .
RUN apt update && apt install
-y -q git
RUN git version
ENTRYPOINT [ "/usr/bin/git" ]
CMD [ "version" ]
LABEL Dep="CSC" \
      Org="SibSUTIS"
```

Dockerfile: LABEL



```
        "OpenStdin": false,
        "StdinOnce": false,
        "Env": [
            "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
            "APP_DIR=/app"
        ],
        "Cmd": [
            "version"
        ],
        "ArgsEscaped": true,
        "Image": "",
        "Volumes": null,
        "WorkingDir": "/app",
        "Entrypoint": [
            "/usr/bin/git"
        ],
        "OnBuild": null,
        "Labels": {
            "Dep": "CSC",
            "Org": "SibSUTIS",
            "org.opencontainers.image.ref.name": "ubuntu",
            "org.opencontainers.image.version": "24.04"
        }
    },
    "Architecture": "amd64",
    "Os": "linux",
    "Size": 199969746,
    "GraphDriver": {
        "Data": {
```

Dockerfile:

```
FROM ubuntu:24.04
ADD [Link] .
ENV APP_DIR=/app
WORKDIR ${APP_DIR}
ARG TARGET_FILE=t.txt
COPY ${TARGET_FILE} .
RUN apt update && apt install
-y -q git
RUN git version
ENTRYPOINT [ "/usr/bin/git" ]
CMD [ "version" ]
LABEL Dep="CSC" \
      Org="SibSUTIS"
```

При просмотре свойств образа параметры находятся в секции labels [1]

[1] Просмотр свойств образа acs // docker image inspect acs

Dockerfile: LABEL

```
$ docker run --label runtime='It.s demo' acs
```

Запуск контейнера с меткой runtime_label

```
"Labels": {
  "Dep": "CSC",
  "Org": "SibSUTIS",
  "runtime": "It.s demo"
}
```

Метки образа и метки контейнера объединяются [1]

В случае определения при запуске контейнера label с существующем в образе именем, будет использоваться значение, определенное при запуске

Dockerfile:

```
FROM ubuntu:24.04
ADD [Link] .
ENV APP_DIR=/app
WORKDIR ${APP_DIR}
ARG TARGET_FILE=t.txt
COPY ${TARGET_FILE} .
RUN apt update && apt install
-y -q git
RUN git version
ENTRYPOINT [ "/usr/bin/git" ]
CMD [ "version" ]
LABEL Dep="CSC" \
      Org="SibSUTIS"
```

[1] Просмотр свойств контейнера acs // docker container inspect acs

Dockerfile: EXPOSE

- Инструкция EXPOSE определяет в метаданных образа, какой порт использует приложение, которое будет работать в образе
- Несет как правило информативный характер – не «открывает» автоматически порты
- (Кроме опции `docker run -P`). Приложение может быть запущено на другом порту (Отличного от порта по умолчанию)
- Не влияют на процесс сборки и запуск (кроме создания слоя сборки)

Dockerfile:

```
FROM ubuntu:24.04
ADD [Link] .
ENV APP_DIR=/app
WORKDIR ${APP_DIR}
ARG TARGET_FILE=t.txt
COPY ${TARGET_FILE} .
RUN apt update && apt install
-y -q git
RUN git version
ENTRYPOINT [ "/usr/bin/git" ]
CMD [ "version" ]
LABEL Dep="CSC" \
      Org="SibSUTIS"
EXPOSE 80/tcp 99/udp
```

Dockerfile: EXPOSE

```
$ docker ps --format '{{ .Names }} \
{{ .Image }} {{ .Status }} {{ .Ports }}'

nginx nginx Up 2 seconds 80/tcp
acs acs Up 24 seconds 80/tcp, 99/udp

$ telnet 127.0.0.1 80
Trying 127.0.0.1...
telnet: connect to address 127.0.0.1: Connection
refused
```

У запущенного контейнера показана информация о портах приложения, сеть контейнера изолирована

```
Dockerfile:
FROM ubuntu:24.04
ADD [Link] .
ENV APP_DIR=/app
WORKDIR ${APP_DIR}
ARG TARGET_FILE=t.txt
COPY ${TARGET_FILE} .
RUN apt update && apt install
-y -q git
RUN git version
ENTRYPOINT [ "cat" ]
LABEL Dep="CSC" \
      Org="SibSUTIS"
EXPOSE 80/tcp 99/udp
```

Dockerfile: USER

- Инструкция USER определяет пользователя, от имени которого будут выполняться *последующие этапы сборки*
- От имени последнего определенного инструкцией пользователя будет запущен процесс внутри контейнера
- Пользователь по умолчанию – root
- Предпочтительнее использовать утилиту gosu [1]

```
$ docker run --rm --entrypoint id acs -u
0
$ docker run --rm --entrypoint id acs -u
1001
```

Запуск контейнера до и после определения инструкции USER.
После определения процесс запускается от имени nonroot

[1] Simple Go-based setuid+setgid+setgroups+exec // https://github.com/tianon/gosu/

Dockerfile:

```
FROM ubuntu:24.04
ADD [Link] .
ENV APP_DIR=/app
WORKDIR ${APP_DIR}
ARG TARGET_FILE=t.txt
COPY ${TARGET_FILE} .
RUN apt update && apt install
-y -q git && useradd nonroot
RUN git version
ENTRYPOINT [ "cat" ]
LABEL Dep="CSC" \
      Org="SibSUTIS"
EXPOSE 80/tcp 99/udp
USER nonroot
```

Dockerfile

- Изменение копируемых файлов изменяет *хэш слоя* – последующие слои собираются заново
- От результата инструкций RUN зависит только USER
- Инструкции RUN не зависят друг от друга – можно объединить в одну инструкцию
- Инструкции от которых не зависят другие инструкции можно вынести в начало Dockerfile

Dockerfile:

```
FROM ubuntu:24.04
ADD [Link] .
ENV APP_DIR=/app
WORKDIR ${APP_DIR}
ARG TARGET_FILE=t.txt
COPY ${TARGET_FILE} .
RUN apt update && apt install
-y -q git && useradd nonroot
RUN git version
ENTRYPOINT [ "cat" ]
LABEL Dep="CSC" \
      Org="SibSUTIS"
EXPOSE 80/tcp 99/udp
USER nonroot
```

Dockerfile

```
$ docker build -t acs -f Dockerfile .
[+] Building 25.1s (12/12) FINISHED
…
 => [1/6] FROM docker.io/library/ubuntu:24.04
 => CACHED [2/6] ADD [link] .
 => CACHED [3/6] WORKDIR /app
 => [4/6] COPY t.txt .
 => [5/6] RUN apt update && apt install -y -q git && useradd nonroot
 => [6/6] RUN git version
…
```

Изменение файла t.txt, результат – cache miss на 4 этапе из 6
Как следствие слои порождаемые командами RUN и тд создаются заново

Время повторной сборки – 25 секунд за счет установки пакетов

Dockerfile:

```
FROM ubuntu:24.04
ADD [Link] .
ENV APP_DIR=/app
WORKDIR ${APP_DIR}
ARG TARGET_FILE=t.txt
COPY ${TARGET_FILE} .
RUN apt update && apt install
-y -q git && useradd nonroot
RUN git version
ENTRYPOINT [ "cat" ]
LABEL Dep="CSC" \
      Org="SibSUTIS"
EXPOSE 80/tcp 99/udp
USER nonroot
```

Dockerfile

- Чаще пакеты не изменяются в процессе сборки образа и целесообразно вынести их в начало
- Изменение файла t.txt приводит к cache-miss на этапе 6 из 6.

```
$ docker build -t acs -f Dockerfile .
[+] Building 1.6s (12/12) FINISHED
…
 => [1/6] FROM docker.io/library/ubuntu:24.04
 => CACHED [2/6] WORKDIR /app
 => CACHED [3/6] RUN apt update && apt install -y -q git && useradd
nonroot
 => CACHED [4/6] RUN git version
 => CACHED [5/6] ADD [link] /
 => [6/6] COPY t.txt
…
```

Dockerfile:

```
FROM ubuntu:24.04
ENV APP_DIR=/app
WORKDIR ${APP_DIR}
ENTRYPOINT [ "cat" ]
ARG TARGET_FILE=t.txt
RUN apt update && apt install
-y -q git && useradd nonroot
RUN git version
ADD [Link] /
COPY ${TARGET_FILE} .
LABEL Dep="CSC" \
      Org="SibSUTIS"
EXPOSE 80/tcp 99/udp
USER nonroot
```

Время повторной сборки – 1.6 секунды

Dockerignore

- При выполнении команды `docker build` создается *контекст сборки*
- Контекст сборки создает индекс файлов внутри контекста
- Докер может работать только с файлами, входящими в контекст
- *Контекст сборки* оказывает влияние при использовании инструкций вида *COPY . .*
- Файлы, которые не являются частью файлов кода проекта или его непосредственных зависимостей не требуются при сборке и их не обязательно включать в контекст
- Синтаксис похож на **`.gitignore`**

`.dockerignore:`

```
node_modules
dist
```

Пароли, секреты, конфиги

- Использование volume
- Использование env
- Использование cmd

- Внешние средства, например, Vault

Live section

Романюта Алексей Андреевич

alexey-r.98@yandex.ru

Кафедра вычислительных систем
Сибирский государственный университет телекоммуникаций и информатики

Dockerfile: VOLUME

- Инструкция VOLUME определяет в метаданных образа необходимость использовать внешнее хранилище и точку его монтирования
- Не влияет на процесс сборки
  При запуске без внешнего хранилища – создает его

Dockerfile:

```
FROM ubuntu:20.04
ADD [Link] .
ENV APP_DIR=/app
WORKDIR ${APP_DIR}
ARG TARGET_FILE=t.txt
COPY ${TARGET_FILE} .
RUN apt update && apt install
-y -q git
RUN git version
ENTRYPOINT [ "/usr/bin/git" ]
CMD [ "version" ]
LABEL Org=„SWC" \
      Author="A. Romanuta"
EXPOSE 80/tcp 99/udp
USER nonroot
VOLUME /data
```

Dockerfile: VOLUME

```
romanutaaa@r9odt-cat:meet [04:18 ] $ docker volume list
DRIVER    VOLUME NAME
romanutaaa@r9odt-cat:meet [04:18 ] $ docker run meet
git version 2.25.1
romanutaaa@r9odt-cat:meet [04:18 ] $ docker volume list
DRIVER    VOLUME NAME
local     6e82d117a8605e2c0d102b61ed8806a10695baa3728708065e0725c
romanutaaa@r9odt-cat:meet [04:18 ] $ sudo ls -al /var/lib/docker/
итого 8
drwxr-xr-x 2 root root 4096 мая 28 04:18 .
drwx-----x 3 root root 4096 мая 28 04:18 ..
romanutaaa@r9odt-cat:meet [04:20 ] $
```

Запуск контейнера привел к созданию нового объекта volume [1]

```
romanutaaa@r9odt-cat:meet [04:21 ] $ docker volume list
DRIVER    VOLUME NAME
romanutaaa@r9odt-cat:meet [04:21 ] $ docker run -v $(pwd):/data meet
git version 2.25.1
romanutaaa@r9odt-cat:meet [04:21 ] $ docker volume list
DRIVER    VOLUME NAME
romanutaaa@r9odt-cat:meet [04:21 ] $
```

Явное указание монтирования в /data не создает новый volume

Dockerfile:

```
FROM ubuntu:20.04
ADD [Link] .
ENV APP_DIR=/app
WORKDIR ${APP_DIR}
ARG TARGET_FILE=t.txt
COPY ${TARGET_FILE} .
RUN apt update && apt install
-y -q git
RUN git version
ENTRYPOINT [ "/usr/bin/git" ]
CMD [ "version" ]
LABEL Org=„SWC" \
      Author="A. Romanuta"
EXPOSE 80/tcp 99/udp
USER nonroot
VOLUME /data
```

[1] Просмотр содержимого volume // sudo ls -al /var/lib/docker/volumes/6e82d117a8605e2c0d102b61ed8806a10695baa3728708065e0725c13edcb5f6/_data

Dockerfile: VOLUME

```
romanutaaa@r9odt-cat:meet [04:23 ] $ docker volume list
DRIVER    VOLUME NAME
romanutaaa@r9odt-cat:meet [04:23 ] $ docker run meet
git version 2.25.1
romanutaaa@r9odt-cat:meet [04:24 ] $ docker volume list
DRIVER    VOLUME NAME
local     3ee5a8aa6735baba1b41984e330d5b7f64310fbda490455b0247f224eb450b28
romanutaaa@r9odt-cat:meet [04:24 ] $ sudo ls -al /var/lib/docker/volumes/3ee5
итого 12
drwxr-xr-x 2 root root 4096 мая 28 04:24 .
drwx-----x 3 root root 4096 мая 28 04:24 ..
-rw-r--r-- 1 root root    5 мая 28 04:23 init.file
romanutaaa@r9odt-cat:meet [04:24 ] $ sudo cat /var/lib/docker/volumes/3ee5a8a
Init
romanutaaa@r9odt-cat:meet [04:24 ] $
```

Запуск контейнера привел к созданию нового объекта volume и размещению в нем файла init.file, созданного на этапе сборки образа [1] [2]

```
romanutaaa@r9odt-cat:meet [04:27 ] $ ls
Dockerfile  Dockerfile1  Dockerfile2  t.txt
romanutaaa@r9odt-cat:meet [04:27 ] $ docker run -v $(pwd):/data meet
git version 2.25.1
romanutaaa@r9odt-cat:meet [04:27 ] $ ls
Dockerfile  Dockerfile1  Dockerfile2  t.txt
```

Не пустые volume не инициализируются

Dockerfile:

```
FROM ubuntu:20.04
ADD [Link] .
ENV APP_DIR=/app
WORKDIR ${APP_DIR}
ARG TARGET_FILE=t.txt
COPY ${TARGET_FILE} .
RUN apt update && apt install
-y -q git
RUN git version
ENTRYPOINT [ "/usr/bin/git" ]
CMD [ "version" ]
LABEL Org=„SWC" \
      Author="A. Romanuta"
EXPOSE 80/tcp 99/udp
USER nonroot
VOLUME /data
```

[1] Просмотр содержимого volume // sudo ls -al /var/lib/docker/volumes/3ee5a8aa6735baba1b41984e330d5b7f64310fbda490455b0247f224eb450b28/_data

[2] Просмотр содержимого файла // sudo cat /var/lib/docker/volumes/3ee5a8aa6735baba1b41984e330d5b7f64310fbda490455b0247f224eb450b28/_data/init.file

Dockerfile: STOPSIGNAL, ONBUILD, HEALTHCHECK, SHELL[1]

- Инструкция STOPSIGNAL определяет какой UNIX-сигнал [2] использовать для остановки приложения в контейнере
- Инструкция ONBUILD определяет набор инструкций, которые необходимо выполнить, если собранный образ будет использоваться в качестве *базового* (FROM *image*)
- Инструкция HEALTHCHECK определяет метод проверки работоспособности приложения (Например, проверка что nginx отдает страницу при обращении на порт 80). Код завершения проверки определяет статус (0 – success, 1 – error)
- Инструкция SHELL определяет какую оболочку использовать для shell-команд (RUN, ENTRYPOINT "command", CMD "command"), по умолчанию – ["/bin/sh", "-c"]

[1] Документация Dockerfile // https://docs.docker.com/engine/reference/builder
[2] Сигналы UNIX // https://ru.wikipedia.org/wiki/Сигнал_(Unix)

Dockerfile: Multistage

- Используются для сокращения размера образов
- Каждый этап не зависит от предыдущего
- Каждая инструкция FROM – один этап

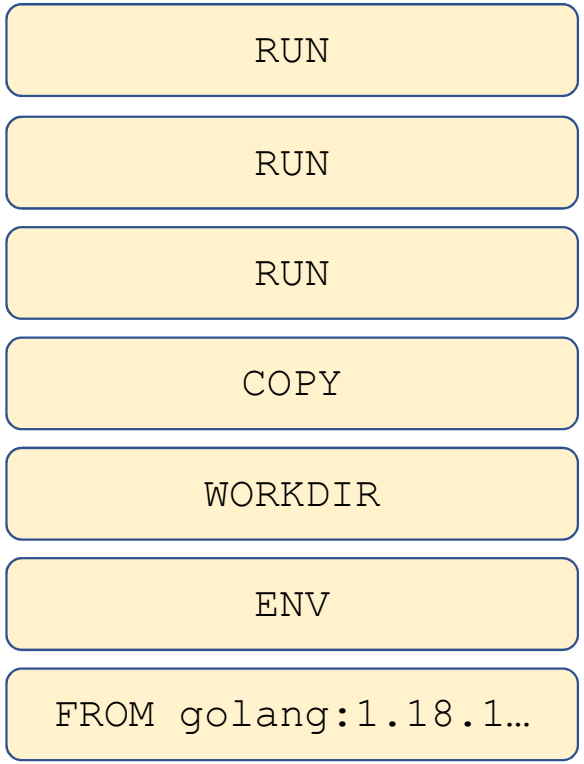
Dockerfile:

```
FROM golang:1.18.1-alpine

ENV GO_SRC=/usr/local/go/src
WORKDIR ${GO_SRC}/tg-bot
COPY . .
RUN apk add --no-cache git make
RUN go get -d ./... && go mod vendor

RUN make build && ${GO_SRC}/tg-bot/bin/app -version
```

На этапе COPY копируются все файлы проекта. Эти файлы существуют в образе после сборки даже если они не требуются, такие как файлы с исходным кодом



Dockerfile: Multistage

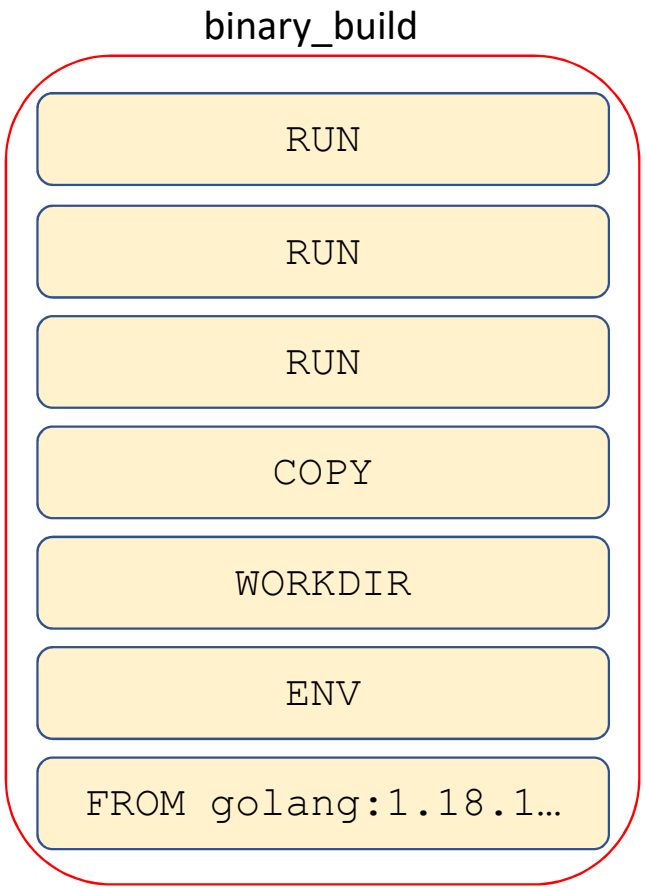
- Этапы сборки необходимо именовать

Dockerfile:

```
FROM golang:1.18.1-alpine as binary_build

ENV GO_SRC=/usr/local/go/src
WORKDIR ${GO_SRC}/tg-bot
COPY . .
RUN apk add --no-cache git make
RUN go get -d ./... && go mod vendor

RUN make build && ${GO_SRC}/tg-bot/bin/app -version
```

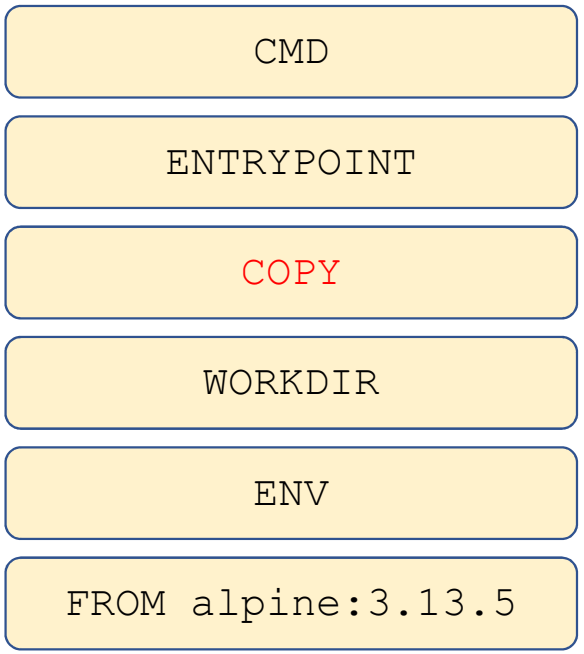


Dockerfile: Multistage

- К результатам сборки на определенном этапе можно обратиться по имени этого этапа

Dockerfile:

```
FROM golang:1.18.1-alpine as binary_build
…
FROM alpine:3.13.5 as release
ENV GO_SRC=/usr/local/go/src
WORKDIR /app
COPY --from=binary_build ${GO_SRC}/tg-bot/bin /app/
ENTRYPOINT ["/app/docker-entrypoint.sh"]
CMD ["/app/app"]
```



На этапе COPY из результатов сборки `binary_build` копируются только исполняемые файлы. В данном примере это исполняемый файл `app` и скрипт запуска `docker-entrypoint.sh`
Остальные файлы проекта в итоговый образ не включаются