Контейнеризация и Основы Работы с Докером

Контейнер и Докер

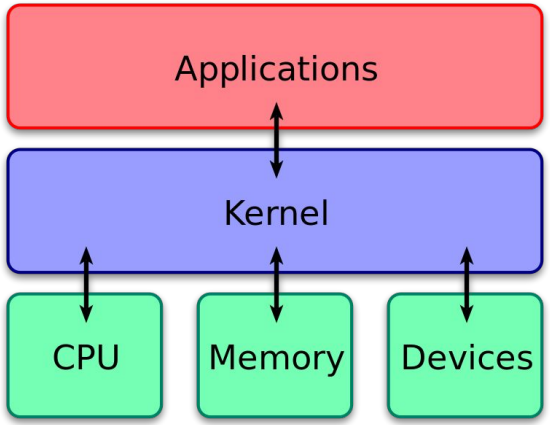
Цель лекции

1. Виртуализация и контейнеризация
2. Что такое образ и контейнер
3. Компоненты Docker
4. Основы Docker CLI
5. Лучшие практики использования докера
6. Dockerfile
7. Методы сборки
8. Мульти-контейнерные приложения и методы их создания

Проблематика

Проблема:

- Размещение одного приложения/сервиса на одном сервере не является экономически эффективным;
- Также на одном сервере не всегда это возможно и может потенциально привести к конфликтам.



Проблематика

Проблема:

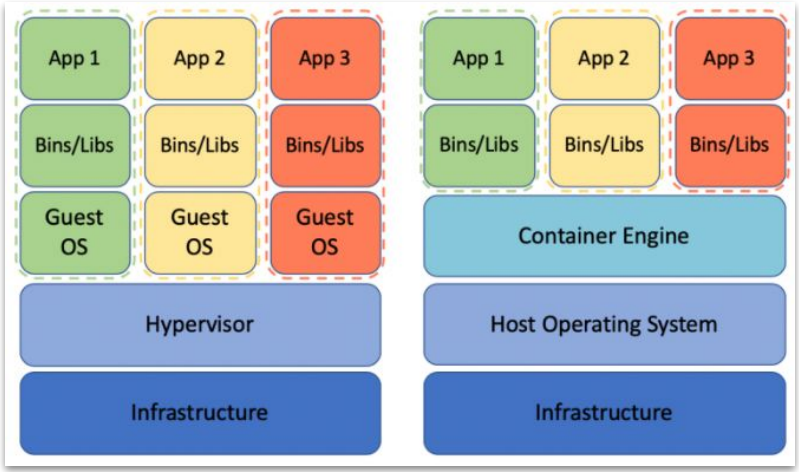
- Зависимости
  - Библиотеки и версии
  - Функции на уровне ОС
- Микросервисы

Идея:

- абстрагировать программное обеспечение от оборудования, создавать изолированные виртуальные среды в единой физической среде.

Контейнеризация и Виртуализация

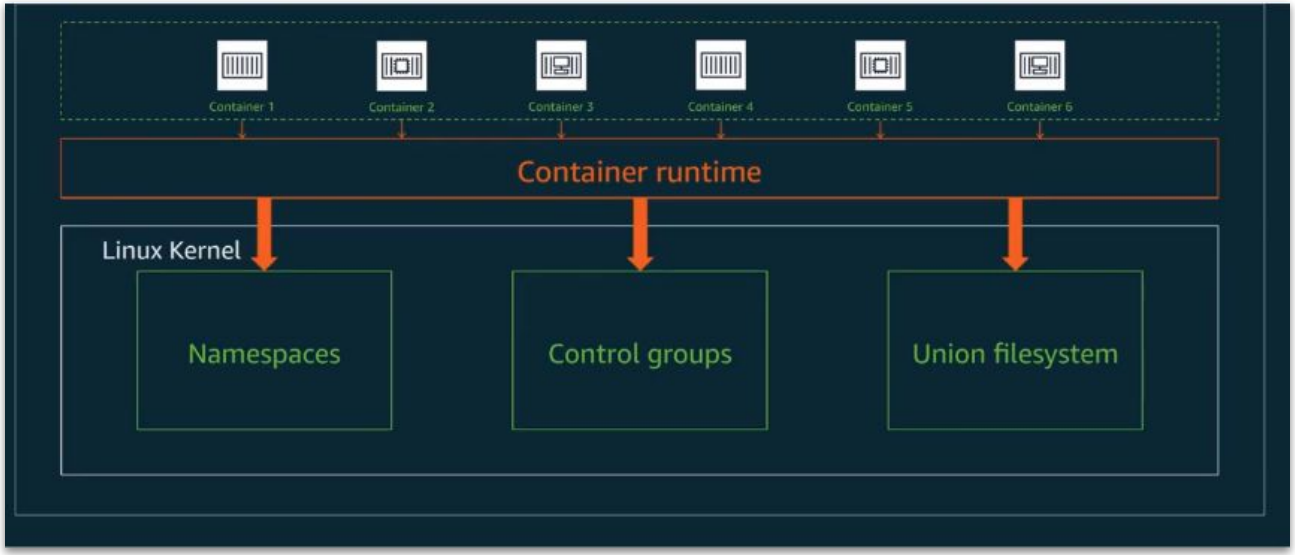
- Контейнеризация — это возможность запускать процессы в изолированной среде
- Контейнер — это работа приложения в изолированное среде со всеми его зависимостями. Благодаря этому приложение может быстро и надежно запускается на различных окружениях
- Контейнер предназначен для использования одного - основного процесса (одно приложение внутри)



Контейнеризация и Виртуализация

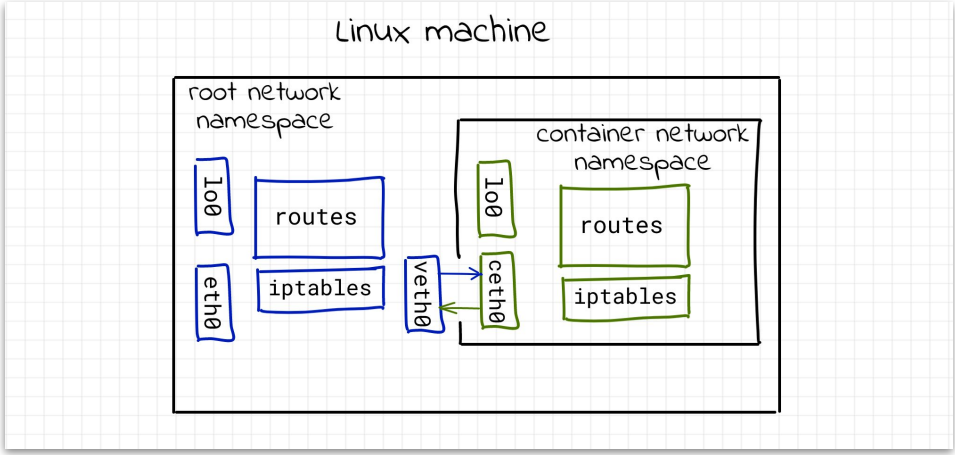
Virtualization	Containerization
Hardware level virtualization	Operating system virtualization
Abstracts OS from hardware	Abstracts Application from OS
Heavyweight	Lightweight
Minutes to startup	Few seconds to startup
More secure and isolated	Less secure and isolated

Контейнеризация - Linux Primitives



Namespaces

- Механизм изоляции ресурсов
- Изменения ресурсов в пространстве имен (namespace) могут быть невидимы за пределами этого пространства имен
- Конфигурация групп процессов, которые используют набор системных ресурсов изолированных от других (пр. сетевые интерфейсы, таблицы маршрутов, пользователи и др.)



Namespaces

- ❏ Network
- ❏ Filesystem (mounts)
- ❏ Processes (pid)
- ❏ Inter-process communication (ipc)
- ❏ Hostname and domain name (uts)
- ❏ User and groups IDs
- ❏ cgroup

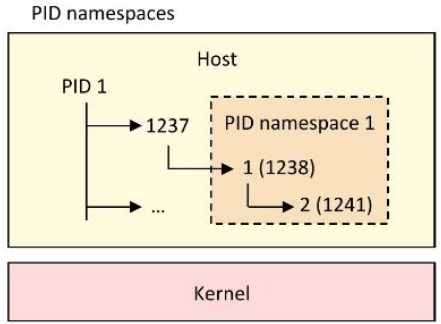
```
saltanov@UbuntuPC:~$ sudo ls -alh /proc/10/ns/
total 0
dr-x--x--x 2 root root 0 Feb 17 22:36 .
dr-xr-xr-x 9 root root 0 Feb 17 22:34 ..
lrwxrwxrwx 1 root root 0 Feb 17 22:36 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Feb 17 22:36 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0 Feb 17 22:36 mnt -> 'mnt:[4026531841]'
lrwxrwxrwx 1 root root 0 Feb 17 22:36 net -> 'net:[4026531840]'
lrwxrwxrwx 1 root root 0 Feb 17 22:36 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Feb 17 22:36 pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Feb 17 22:36 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Feb 17 22:36 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Feb 17 22:36 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Feb 17 22:36 uts -> 'uts:[4026531838]'
```

- Namespaces видны в /proc
- Файлы внутри являются символическими ссылками для каждого namespace
- Ссылка содержит тип namespace и номер inode для ее идентификации

Namespaces

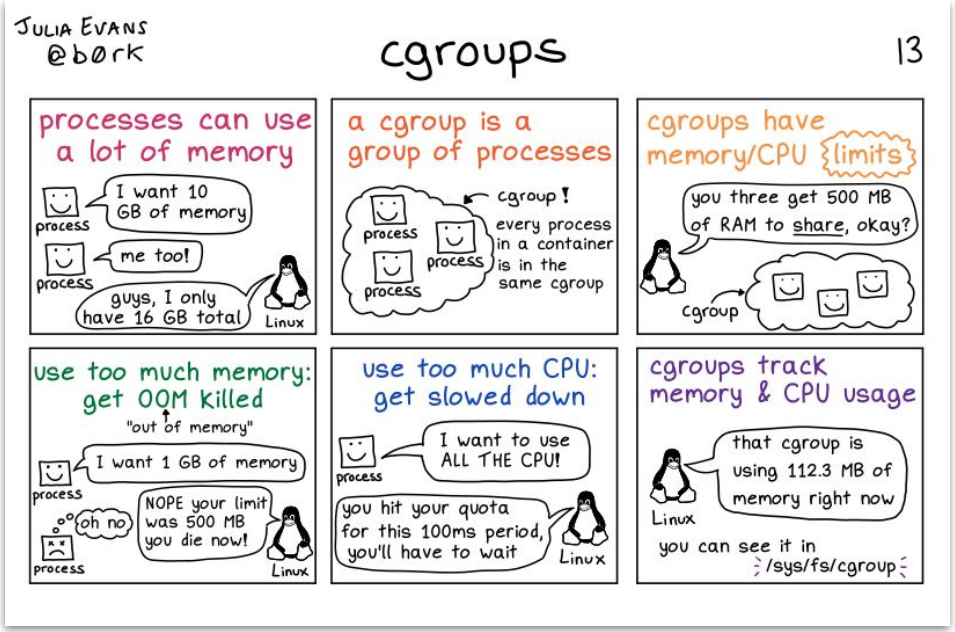
The namespace - init process

- Каждый контейнер имеет свою собственную иерархию процессов, который начинается с «PID 1» (родитель всего дерева процессов) внутри контейнера. Всякий раз, когда процесс с PID 1 завершается, все пространство имен процесса (а вместе с ним и контейнер) автоматически уничтожается ядром
  - *If the "init" process of a PID namespace terminates, the kernel terminates all of the processes in the namespace via a SIGKILL signal.*
- За пределами контейнеров (или других namespaces) PID 1 не разрешено завершаться — если это когда-либо произойдет, система намеренно аварийно завершится (kernel panic)
- Кроме того, все процессы у которых больше нет родителя, автоматически переводятся в PID 1, поэтому он всегда должен существовать



Cgroups

- Организует все процессы в системе в группы
- Учет использования ресурсов и сбор данных об их использовании
- Ограничивает или устанавливает приоритеты использования ресурсов



Cgroups

- Система Cgroups — это абстрактная структура
- Подсистемы — это конкретные реализации
- Различные подсистемы могут организовывать процессы по отдельности
- Большинство подсистем — это контроллеры ресурсов
- Примеры:
  - Memory
  - CPU time
  - Block I/O
  - Number of discrete processes (pids)
  - Network priority
  - Freezer (used by docker pause)
  - Devices

Cgroups

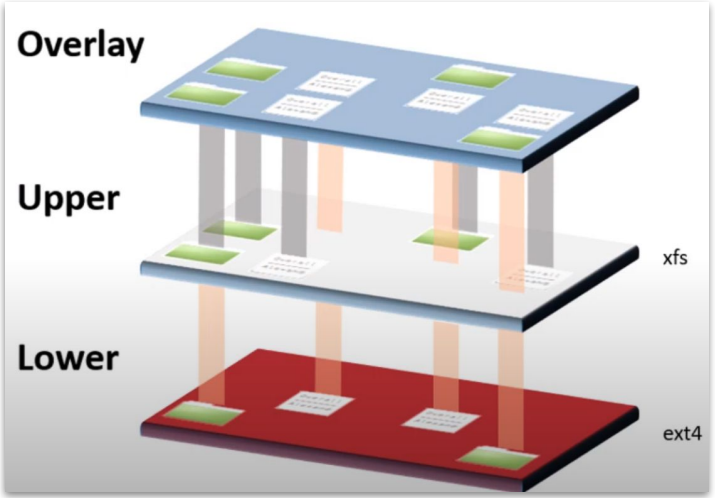
- Cgroups могут использоваться независимо от контейнеров
- Cgroup ограничивают использование ресурсов для процессов
  - Каждый новый процесс наследует конкретную cgroups от родительского процесса
- Cgroups мониторят процессы и организуют их

Пример использования cgroup для запущенного контейнера подсистемы CPU

```
saltanov@linuxpc:/sys/fs/cgroup/cpu/docker/cff6671a87984a79136cd2b689e10878cae4ac56c7b3932644ab0d836e51861e$ ll
total 0
drwxr-xr-x 2 root root 0 дек 23 01:23 ./
drwxr-xr-x 3 root root 0 дек 21 21:30 ../
-rw-r--r-- 1 root root 0 дек 23 01:27 cgroup.clone_children
-rw-r--r-- 1 root root 0 дек 23 01:23 cgroup.procs
-r--r--r-- 1 root root 0 дек 23 01:27 cpuacct.stat
-rw-r--r-- 1 root root 0 дек 23 01:27 cpuacct.usage
-r--r--r-- 1 root root 0 дек 23 01:27 cpuacct.usage_all
-r--r--r-- 1 root root 0 дек 23 01:27 cpuacct.usage_percpu
-r--r--r-- 1 root root 0 дек 23 01:27 cpuacct.usage_percpu_sys
-r--r--r-- 1 root root 0 дек 23 01:27 cpuacct.usage_percpu_user
-r--r--r-- 1 root root 0 дек 23 01:27 cpuacct.usage_sys
-r--r--r-- 1 root root 0 дек 23 01:27 cpuacct.usage_user
-rw-r--r-- 1 root root 0 дек 23 01:27 cpu.cfs_period_us
-rw-r--r-- 1 root root 0 дек 23 01:27 cpu.cfs_quota_us
-rw-r--r-- 1 root root 0 дек 23 01:27 cpu.shares
-r--r--r-- 1 root root 0 дек 23 01:27 cpu.stat
-rw-r--r-- 1 root root 0 дек 23 01:27 cpu.uclamp.max
-rw-r--r-- 1 root root 0 дек 23 01:27 cpu.uclamp.min
-rw-r--r-- 1 root root 0 дек 23 01:27 notify_on_release
-rw-r--r-- 1 root root 0 дек 23 01:27 tasks
```

Image, Layers and Union Filesystem

- OverlayFS — это современная версия Union Filesystem
- OverlayFS включает отдельные файлы и каталоги как ветви или слои
- Она позволяет виртуально объединить эти слои для представления единого унифицированного вида
- Эти слои могут объединять данные на разных разделах или на разных файловых системах (например, ext4 и xfs)

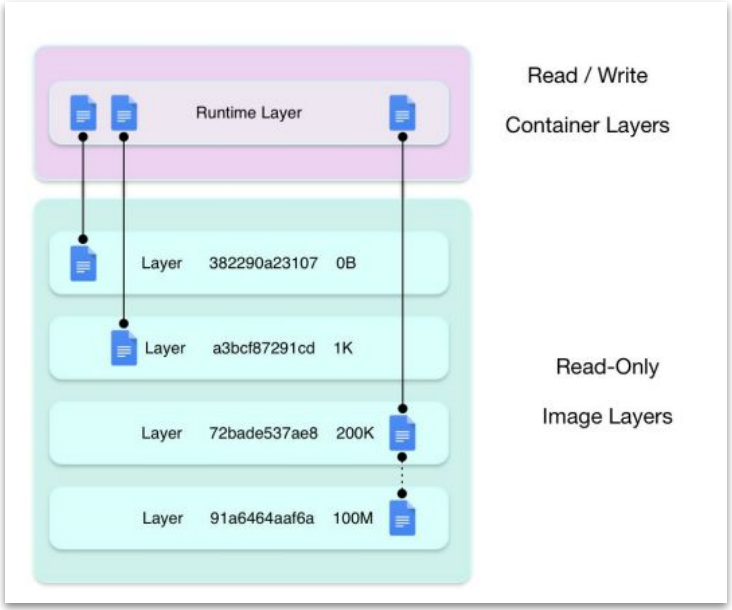


Image, Layers

Image - это шаблон, доступный только для чтения, с инструкциями по созданию контейнера.

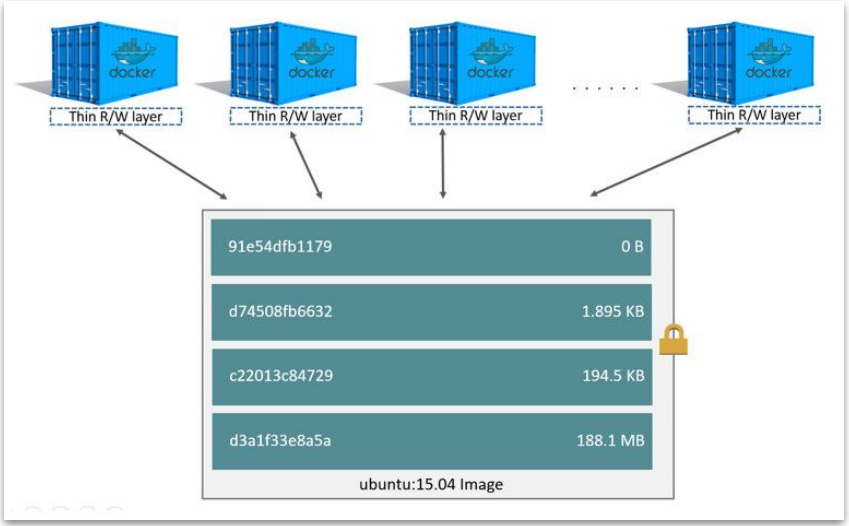
Образ состоит из слоев

- Каждый слой хранит дельту изменений (например, добавляет какой-либо файл или изменяет предыдущий)
- Стратегия копирования при записи - Copy-on-Write (CoW)
- **$docker history <image_name>** — история команд в Dockerfile



Image, Layers

- Images представляют собой файловую систему
- Images популярны для виртуализации и контейнерных систем
- Слои могут одновременно использоваться несколькими запущенными контейнерами (общая библиотека)
  - Легкое масштабирование
  - Запуск нескольких экземпляров (развертывание нескольких сред одновременно)



Image, Layers

- *Пример создание слоев*

```
# syntax=docker/dockerfile:1
FROM ubuntu:18.04
RUN mkdir /test1 && echo "Hello" > /test1/file.txt
```

```
saltanov@linuxpc:~/temp$ docker image build -t testimage1 ./
Sending build context to Docker daemon  2.048kB
Step 1/2 : FROM ubuntu:18.04
18.04: Pulling from library/ubuntu
284055322776: Pull complete
Digest: sha256:0fedbd5bd9fb72089c7bbca476949e10593cebed9b1fb9edf5b79dbbacddd7d6
Status: Downloaded newer image for ubuntu:18.04
 ---> 5a214d77f5d7
Step 2/2 : RUN mkdir /test1 && echo "Hello" > /test1/file.txt
 ---> Running in 731f1a8efb45
Removing intermediate container 731f1a8efb45
 ---> ae5da9297e0e
Successfully built ae5da9297e0e
Successfully tagged testimage1:latest
```

```
saltanov@linuxpc:/var/lib$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
redis         latest    40c68ed3a4d2   5 weeks ago     113MB
ubuntu        latest    ba6acccedd29   2 months ago    72.8MB
hello-world   latest    d1165f221234   9 months ago    13.3kB
saltanov@linuxpc:/var/lib$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
testimage1    latest    ae5da9297e0e   12 seconds ago   63.1MB
redis         latest    40c68ed3a4d2   5 weeks ago      113MB
ubuntu        latest    ba6acccedd29   2 months ago     72.8MB
ubuntu        18.04     5a214d77f5d7   2 months ago     63.1MB
hello-world   latest    d1165f221234   9 months ago     13.3kB
```

```
root@linuxpc:/var/lib/docker/overlay2# ls -tl
total 36
drwx------ 2 root root 4096 дек 27 01:41 l
drwx--x--- 4 root root 4096 дек  2 07:51 8e354e047da87ac59583345c7bf1a567ee83d47fe1645d9bbbe0e9ef9ab36814
drwx--x--- 4 root root 4096 дек  2 07:51 e457e0306d26e4123314700f08e352b67421879cda9bf68a14b4a045194acd32
drwx--x--- 4 root root 4096 дек  2 07:51 36420716cba4df5320ec70996a20d5320c75442e6bb1e38cd715562730b79caa
drwx--x--- 4 root root 4096 дек  2 07:51 59abcf5f57d9487ed5163d62c0102e038e5ede6dc7bfc195c86f52fedafc46af
drwx--x--- 4 root root 4096 дек  2 07:51 d069dc0460caafe47f6fc0a58f5a83ecb9763a75e8a1e057dfccf7061f643e84
drwx--x--- 3 root root 4096 дек  2 07:51 04154f77448cf13da51071c2a5b363caf92907541b1172bd9f60cf2711e3a2d4
drwx--x--- 3 root root 4096 ноя 20 01:00 795b8303720b409948618915bc52d5e3ea580a92b53f13cb2707b00eaffbb6fc
drwx-----x 3 root root 4096 авг 24 20:55 2e0122b1eadd400c40a5b6ddfcc2cc906f295bb8f3e933cf49c7e964f6652585
root@linuxpc:/var/lib/docker/overlay2# ls -tl
total 44
drwx------ 2 root root 4096 дек 27 01:43 l
drwx--x--- 4 root root 4096 дек 27 01:41 67b29b2c2c8c2d6a3b04fc4edd93981bea255dc0309009701efc7f0eb38344d2
drwx--x--- 3 root root 4096 дек 27 01:42 3974af62148a49cc98848dc963313fea18ca7ccb09555c2882902df68b45cba3
drwx--x--- 4 root root 4096 дек  2 07:51 8e354e047da87ac59583345c7bf1a567ee83d47fe1645d9bbbe0e9ef9ab36814
drwx--x--- 4 root root 4096 дек  2 07:51 e457e0306d26e4123314700f08e352b67421879cda9bf68a14b4a045194acd32
drwx--x--- 4 root root 4096 дек  2 07:51 36420716cba4df5320ec70996a20d5320c75442e6bb1e38cd715562730b79caa
drwx--x--- 4 root root 4096 дек  2 07:51 59abcf5f57d9487ed5163d62c0102e038e5ede6dc7bfc195c86f52fedafc46af
drwx--x--- 4 root root 4096 дек  2 07:51 d069dc0460caafe47f6fc0a58f5a83ecb9763a75e8a1e057dfccf7061f643e84
drwx--x--- 3 root root 4096 дек  2 07:51 04154f77448cf13da51071c2a5b363caf92907541b1172bd9f60cf2711e3a2d4
drwx--x--- 3 root root 4096 ноя 20 01:00 795b8303720b409948618915bc52d5e3ea580a92b53f13cb2707b00eaffbb6fc
drwx-----x 3 root root 4096 авг 24 20:55 2e0122b1eadd400c40a5b6ddfcc2cc906f295bb8f3e933cf49c7e964f6652585
```

```
root@linuxpc:/var/lib/docker/overlay2/67b29b2c2c8c2d6a3b04fc4edd93
981bea255dc0309009701efc7f0eb38344d2/diff# ls
test1
```

Image, Layers

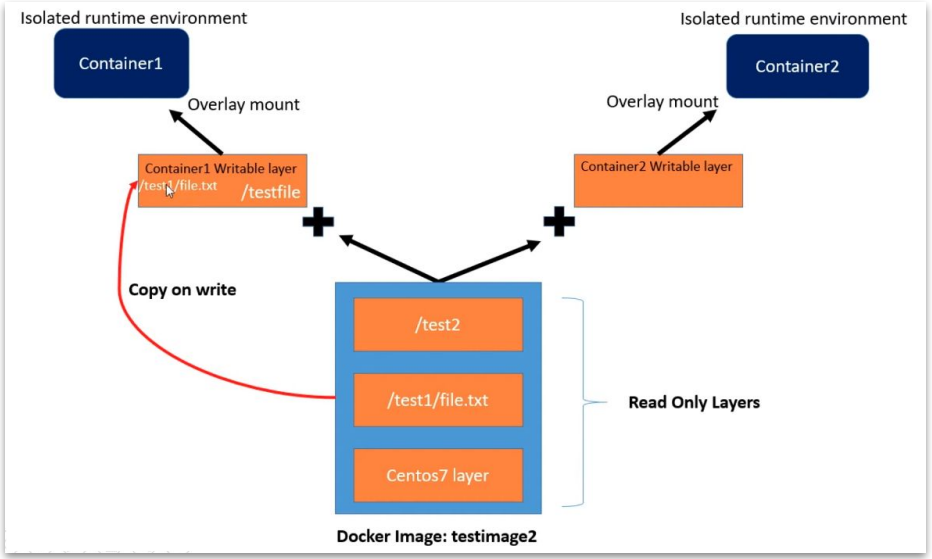
- *Связь слоев друг с другом*

```
root@linuxpc:/var/lib/docker/overlay2/67b29b2c2c8c2d6a3b04fc4edd93981bea255dc0309009701efc7f0eb38344d2# cat lower
l/POZLA6DCVR36LUXLNJNEKPL2JZroot@linuxpc:/var/lib/docker/overlay2/67b29b2c2c8c2d6a3b04fc4edd93981bea255dc030900970

root@linuxpc:/var/lib/docker/overlay2# ls -tl
total 44
drwx------ 2 root root 4096 дек 27 01:43 l
drwx--x--- 4 root root 4096 дек 27 01:43 67b29b2c2c8c2d6a3b04fc4edd93981bea255dc0309009701efc7f0eb38344d2
drwx--x--- 3 root root 4096 дек 27 01:42 3974af62148a49cc98848dc963313fea18ca7ccb09555c2882902df68b45cba3
drwx--x--- 4 root root 4096 дек  2 07:51 8e354e047da87ac59583345c7bf1a567ee83d47fe1645d9bbbe0e9ef9ab36814
drwx--x--- 4 root root 4096 дек  2 07:51 e457e0306d26e4123314700f08e352b67421879cda9bf68a14b4a045194acd32
drwx--x--- 4 root root 4096 дек  2 07:51 36420716cba4df5320ec70996a20d5320c75442e6bb1e38cd715562730b79caa
drwx--x--- 4 root root 4096 дек  2 07:51 59abcf5f57d9487ed5163d62c0102e038e5ede6dc7bfc195c86f52fedafc46af
drwx--x--- 4 root root 4096 дек  2 07:51 d069dc0460caafe47f6fc0a58f5a83ecb9763a75e8a1e057dfccf7061f643e84
drwx--x--- 3 root root 4096 дек  2 07:51 04154f77448cf13da51071c2a5b363caf92907541b1172bd9f60cf2711e3a2d4
drwx--x--- 3 root root 4096 ноя 20 01:00 795b8303720b409948618915bc52d5e3ea580a92b53f13cb2707b00eaffbb6fc
drwx-----x 3 root root 4096 авг 24 20:55 2e0122b1eadd400c40a5b6ddfcc2cc906f295bb8f3e933cf49c7e964f6652585
root@linuxpc:/var/lib/docker/overlay2# cd 3974af62148a49cc98848dc963313fea18ca7ccb09555c2882902df68b45cba3/
root@linuxpc:/var/lib/docker/overlay2/3974af62148a49cc98848dc963313fea18ca7ccb09555c2882902df68b45cba3# ll
total 24
drwx--x---  3 root root  4096 дек 27 01:42 ./
drwx--x--- 13 root root 12288 дек 27 01:43 ../
-rw-------  1 root root     0 дек 27 01:43 committed
drwxr-xr-x 21 root root  4096 дек 27 01:42 diff/
-rw-r--r--  1 root root    26 дек 27 01:42 link
root@linuxpc:/var/lib/docker/overlay2/3974af62148a49cc98848dc963313fea18ca7ccb09555c2882902df68b45cba3# cat link
POZLA6DCVR36LUXLNJNEKPL2JZroot@linuxpc:/var/lib/docker/overlay2/3974af62148a49cc98848dc963313fea18ca7ccb09555c288
```

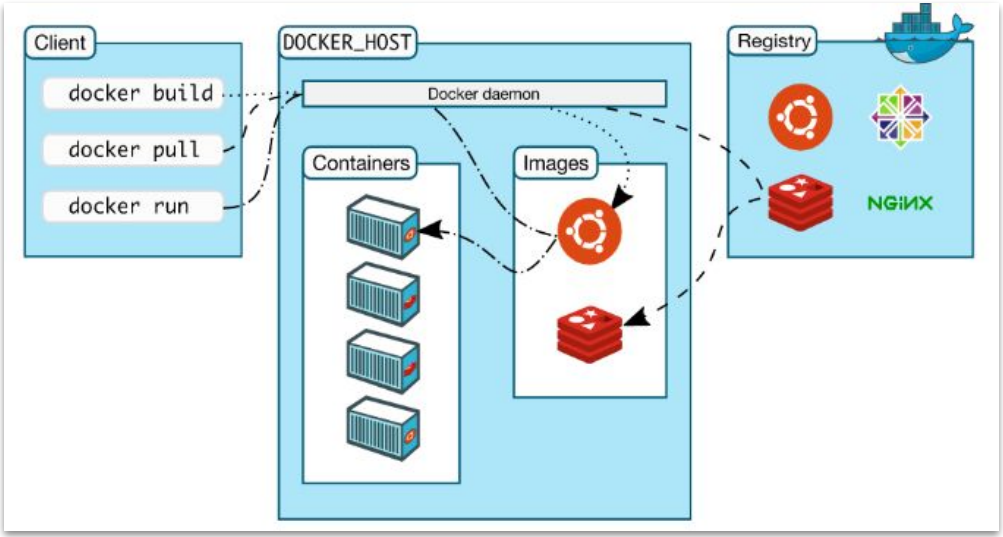
Image, Layers - Copy on Write

- Любые новые изменения вносятся только для конкретного контейнера в пределах его записываемого слоя.
- Если мы хотим изменить любой файл/каталог из базового слоя, файл будет сначала скопирован в записываемый слой и изменен там.
- Операции копирования при записи являются тяжелыми и могут повлиять на производительность, поэтому образ должен быть тщательно спроектирован.
- Удаленные файлы скрыты, не удаляются физически

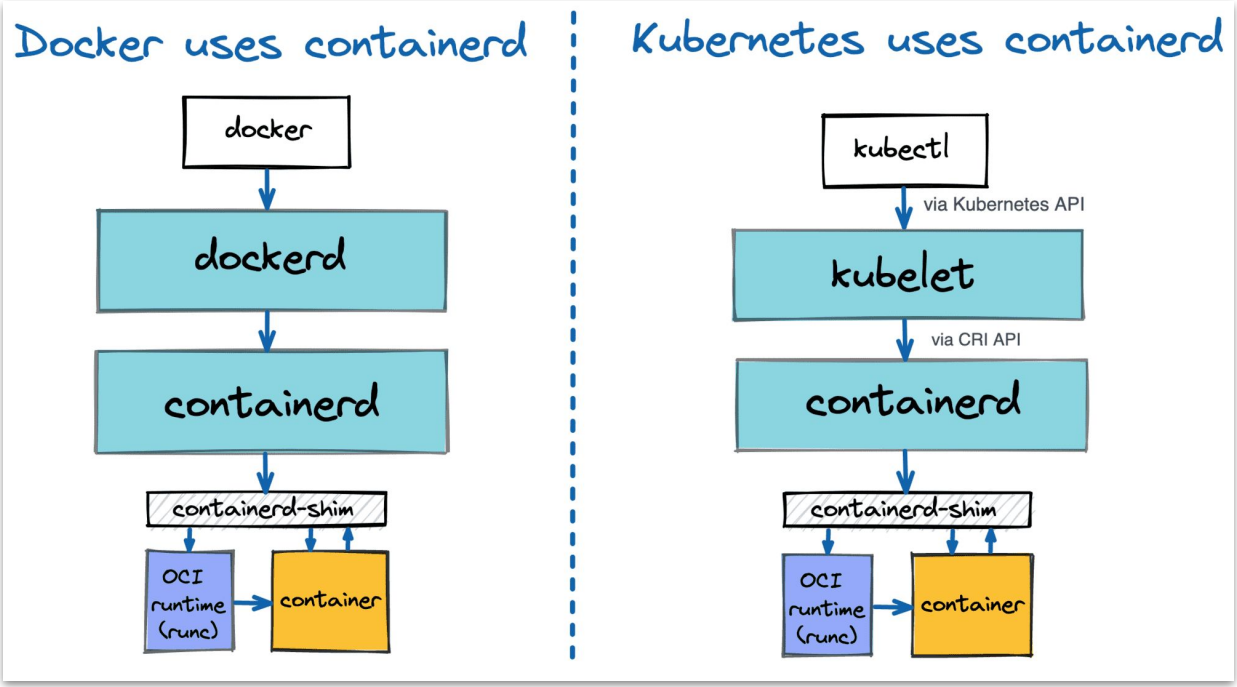


Компоненты Docker

- **Docker daemon (dockerd)** — слушает API запросы и управляет объектами Docker, такими как образы, контейнеры, сеть, разделы
- **Docker client (docker)** — основной способ взаимодействия пользователей с Docker
- **Docker registry** — хранит образы Docker. Docker Hub — это общедоступный реестр образов и Docker по умолчанию настроен на работу с DockerHub.



Компоненты Docker



Docker CLI

Container - это экземпляр запущенного образа

- ❏ `$docker run -it ubuntu /bin/bash` # run container interactively; `-i` - *keep open stdin*, `-t` - *allocate a pseudo-TTY*
- ❏ `$docker run -d nginx` # run container as an application in the background
- ❏ `$docker ps [-a]` # list running or (with -a) all containers including the stopped ones
- ❏ `$docker inspect` # get info about container

manipulate container lifecycle

- ❏ `$docker stop/start/restart/kill/rm <ID>/<name>`

execute command [interactively]

- ❏ `$docker exec [-it] <ID>/<name> <command>`

```
saltanov@UbuntuPC:~$ docker ps -a
CONTAINER ID   IMAGE             COMMAND       CREATED              STATUS
   PORTS     NAMES
6aa6de39f32a   python:3.9-slim   "python3"     About a minute ago   Exited (1) 44 seconds ago
             awesome_engelbart
c450bc840880   python:3.9-slim   "python3"     2 minutes ago        Exited (1) About a minute ago
             fervent_satoshi
a1240d1c726f   python:3.9-slim   "python3"     2 minutes ago        Exited (1) 2 minutes ago
             cranky_solomon
172c17b2ec46   python:3.9-slim   "/bin/bash"   3 minutes ago        Exited (130) 3 minutes ago
             mystifying_gagarin
31c63ec2d431   python:3.9-slim   "python3"     3 minutes ago        Exited (0) 3 minutes ago
             practical_lehmann
b17b6378e37d   python:3.9-slim   "python3"     4 minutes ago        Exited (0) 4 minutes ago
             silly_swirles
dc0ba1304f5e   hello-world       "/bin/sh"     7 minutes ago        Created
             exciting_bouman
550ea856dc92   hello-world       "/bin/bash"   7 minutes ago        Created
             elegant_merkle
24e415e33cb0   hello-world       "/hello"      7 minutes ago        Exited (0) 7 minutes ago
             sharp_euler
6e0ab1ce5c80   hello-world       "/hello"      8 minutes ago        Exited (0) 8 minutes ago
             gracious_kirch
b4656ef734c1   hello-world       "/hello"      8 minutes ago        Exited (0) 8 minutes ago
             wonderful_stonebraker
saltanov@UbuntuPC:~$ docker rm $(docker ps -aq)
6aa6de39f32a
c450bc840880
a1240d1c726f
172c17b2ec46
31c63ec2d431
b17b6378e37d
dc0ba1304f5e
550ea856dc92
24e415e33cb0
6e0ab1ce5c80
b4656ef734c1
saltanov@UbuntuPC:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
saltanov@UbuntuPC:~$
```

Docker CLI

Ref: https://docs.docker.com/engine/reference/run/

$docker run -d --rm --name webserver -p 8080:80 nginx

- -d # run as daemon
- --rm # remove container after its exit
- --name # specify future container name
- -p <host_port>:<container_port> # publish ports

Limit container resources: Ref: https://docs.docker.com/config/containers/resource_constraints/ - преимущество исользования cgroups в контейнерах

CPU:

- --cpus=<max_amount_of_CPUs_available_for_the_container>
- --cpuset-cpus=<list_of_specific_CPUs_numbers>

Memory:

- --memory="<max_amount_of_memory_available_for_the_container>"
- --memory-reservation="<amount of memory>" # soft limit - активируется, когда хост занят и необходимо взять часть ресурсов памяти из этого контейнера, чтобы обеспечить хост, в противном случае OOM Linux killer может начать уничтожать процессы

Docker CLI

- ```
  $docker run -d \
  -e <env1_name>=<env1_value> \
  -e <env2_name>=<env2_value> \
  <image_name>
  ```

- ```
  $docker run -d \
  --name some-postgres \
  -e POSTGRES_PASSWORD=mysecretpassword \
  -e PGDATA=/var/lib/postgresql/data/pgdata \
  -v /custom/mount:/var/lib/postgresql/data \
  postgres
  ```

> **`PGDATA`**
>
> This optional variable can be used to define another location - like a subdirectory - for the database files. The default is `/var/lib/postgresql/data`. If the data volume you're using is a filesystem mountpoint (like with GCE persistent disks) or remote folder that cannot be chowned to the `postgres` user (like some NFS mounts), Postgres `initdb` recommends a subdirectory be created to contain the data.

Docker CLI

- Очистка остановленных, завершенных и запущенных контейнеров

```
saltanov@UbuntuPC:~$ docker system df
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          6         1         2.37GB    2.245GB (94%)
Containers      1         0         166.3kB   166.3kB (100%)
Local Volumes   0         0         0B        0B
Build Cache     39        0         2.339GB   2.339GB
saltanov@UbuntuPC:~$ docker ps -a
CONTAINER ID   IMAGE             COMMAND     CREATED         STATUS                       PORTS     NAMES
917dc825a73f   python:3.9-slim   "python3"   15 months ago   Exited (137) 15 months ago             distracted_varahamihira
saltanov@UbuntuPC:~$ docker rm $(docker ps -aq) -f
917dc825a73f
```

- Очистка образов

```
saltanov@UbuntuPC:~$ docker images -a
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
python        3.9-slim  6cd7b6dc1340   17 months ago   125MB
<none>        <none>    d94369111b63   21 months ago   992MB
<none>        <none>    0dbf82e6835f   21 months ago   169MB
<none>        <none>    7b6e59279c38   21 months ago   275MB
hello-world   latest    feb5d9fea6a5   2 years ago     13.3kB
<none>        <none>    121454ddad72   3 years ago     810MB
saltanov@UbuntuPC:~$ docker rmi $(docker images -qa)
Untagged: python:3.9-slim
Untagged: python@sha256:d95dc32274f817debe886e6c5a6164bf4e0d996632d8cb56fde89189134db9d7
Deleted: sha256:6cd7b6dc13400ab636d84edf9d5818a6ed55219b9149e0b447255d7abe824af9
Deleted: sha256:d94369111b63e4994131e9e8151de1f6afc5f49e83aae805dcc095b7b6b672d8
Deleted: sha256:dba91b509702d3fd0a49c9aacc3068ab6cded7c79d049416c96d608151cd9bdc
Deleted: sha256:006093555b30e03fd779c435b25282d1b95250060a2df1e0600953105170fb21
Deleted: sha256:89e8e088ed044a748ef54eaa7d2e22e8442e4f3176eff31ef756927fc8f3c3a8
```

Docker CLI

- Очистка кешей (BuildKit)

```
saltanov@UbuntuPC:~$ docker system df
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          0         0         0B        0B
Containers      0         0         0B        0B
Local Volumes   0         0         0B        0B
Build Cache     39        0         2.339GB   2.339GB
saltanov@UbuntuPC:~$ docker buildx prune -f
ID                                              RECLAIMABLE     SIZE            LAST ACCESSED
o9fmmciorpnp4bc8sber5mlcw                       true            14.69MB         17 months ago
xrvm6kiyxu45ox1cmk3iekal6*                      true    0B              17 months ago
sn2rkpomfvlgnjpcnqk4dp178                       true    14.71MB         16 months ago
dpro55ap6ck3mshtyqssfc2fl*                      true    0B              16 months ago
j8gey8a6helae9e6iay3s7r93*                      true    15.83MB         16 months ago
1w8mxccj367ru2mtn2bfshbph*                      true    15.81MB         17 months ago
n4of1fdbnrw8cb6xiz87c7n6j*                      true    602B            17 months ago
```

```
saltanov@UbuntuPC:~$ docker system df
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          0         0         0B        0B
Containers      0         0         0B        0B
Local Volumes   0         0         0B        0B
Build Cache     0         0         0B        0B
```

Контейнер анализ логов и ресурсов (со стороны хоста)

- Общий лог активности индивидуального контейнера `/var/lib/docker/containers/CONTAINER_ID/CONTAINER_ID.log`
  - Однако, удалить контейнер (docker rm) → то лог будет удален
- Список всех образов локального репозитория `/var/lib/docker/image/overlay2/repositories.json`
- Общая информация о конфигурации контейнера `/var/lib/docker/containers/CONTAINER_ID/config.v2.json`
  - *Name*
  - *Image ID*
  - *Driver*
  - *State.StartedAt & State.FinishedAt*
  - *Path & Args*
  - *CMD commands when docker run*
  - *Mount points*
  - *etc*

Контейнер анализ логов и ресурсов (внутри контейнера)

Как правило внутри контейнера отсутствует возможности инструментов мониторинга ввиду ограниченности ресурсов, однако можно анализировать данные вручную:

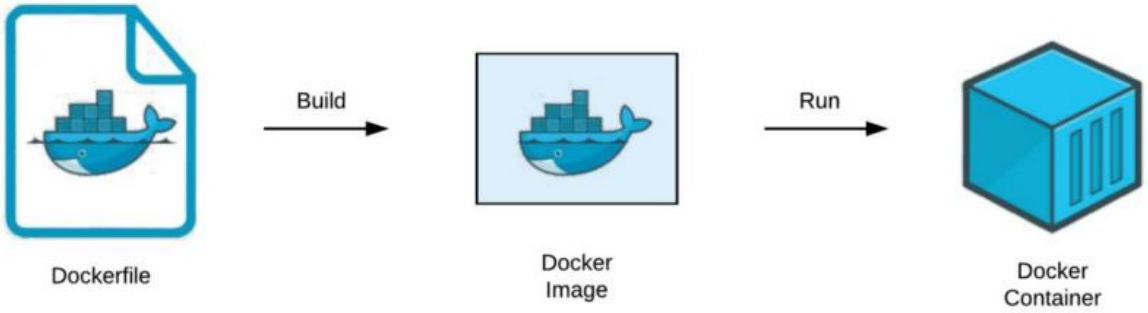
- `$grep MemTotal /proc/meminfo` - общая память на физическом хосте
- `$cat /proc/uptime` - время с момента старта системы
- `$cat /proc/<pid>/environ` - можно просмотреть какие переменные окружения использовали при запуске контейнера
- `$cat /proc/<pid>/cmdline` - команда которая использовалась при запуске контейнера

Основные выводы и рекомендации - лучшие практики

- Не используйте контейнер для развертывания базы данных в Prod
  - Тяжелая многослойная модель файловой системы - неэффективная для частых изменений (операции CoW)
  - Контейнер ненадежен для хранения и синхронизации критически важных данных (ограниченные ресурсы с Cgroups, дополнительный уровень сложности с overlayFS)
- Разрабатывайте одно приложение — один контейнер для запуска
  - Масштабирование контейнера и установки новых обновлений более очевидно и просто в управлении с одним приложением внутри
  - Не возможно отслеживать, отправлять сигналы unix подпроцессам (вторичным приложениям) с точки зрения контейнера (в пространствах имен только PID 1 получает сигналы от хоста)
  - Если основной процесс PID 1 завершается, ломается, то другие субпроцессы автоматически также будут завершены ядром
- Если у вас есть скрипт bash в качестве точки входа запуска контейнера, то один из вариантов обновления PID1 (из bash в приложение) — это использование команды $exec

Сборка приложений с Докером

Сборка образа



Dockerfile

Команды сборки образа:

Ref: https://docs.docker.com/engine/reference/commandline/build

- `docker build .`
- `docker build -t <image-name>[:<image-tag>] .`
- `docker build -f Dockerfile.dev .`

Основные инструкции:

Установить основной образ

- `FROM <base-image-name>[:<base-image-tag>]`

Установить metadata, например Author (можно просмотреть в запущенном контейнере)

- `LABEL <label-name>="<label-value>"`

Запустить команды

- `RUN <command> or RUN ["executable", "arg1", "arg2"]`

Установить metadata об используемых портах и для inter-container взаимодействия

- `EXPOSE <port-number>`

Установить переменные окружения в контейнере или во время сборки

- `ENV <name>=<value> or ARG <name>[=<default-value>]`

```
FROM python:3.7-alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev
linux-headers
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
EXPOSE 5000
COPY . .
CMD ["flask", "run"]
```

Dockerfile

ADD and COPY:

COPY

- ❏ `COPY <src> <dst>`

ADD тоже самое назначение однако:

- ❏ Может загружать удаленные файлы
- ❏ Разархивировать архивы

ПРИМЕЧАНИЕ: для обычных операций копирования использовать COPY

- ❏ более прозрачно, чем ADD. COPY поддерживает только базовое копирование локальных файлов в контейнер
- ❏ лучшее использование ADD — это автоматическое извлечение локального tar-файла в образ, как например **ADD rootfs.tar.xz /**.

```
FROM python:3.7-alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev
linux-headers
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
EXPOSE 5000
COPY . .
CMD ["flask", "run"]
```

Dockerfile

ADD and COPY:

- Если есть несколько файлов частота изменения которых предопределена, КОПИРУЙТЕ их по отдельности, а не все сразу.
  - Гарантирует, что кэш сборки каждого шага будет аннулирован только в случае изменения конкретного файла в этом шаге.

```
1 COPY requirements.txt /tmp/
2 RUN pip install --requirement /tmp/requirements.txt
3 COPY . /tmp/
```

В данном примере разделение помогает закешировать слой установки приложения, если не меняются зависимости. Этот слой отделен от изменений остальных файлов приложения

Докерфайл

CMD и ENTRYPOINT — обе инструкции определяют, какая команда выполняется при запуске контейнера.

Существует несколько правил, описывающих их взаимодействие:

- Dockerfile должен указать хотя бы одну команду CMD или ENTRYPOINT.
- ENTRYPOINT должен быть определен при использовании контейнера в качестве пользовательского исполняемого файла.
- Точка входа по умолчанию: /bin/sh -c.
- CMD следует использовать как способ определения аргументов по умолчанию для команды ENTRYPOINT или для выполнения специальной команды в контейнере.
- CMD может быть переопределен при запуске контейнера с альтернативными аргументами.
- ENTRYPOINT можно перезаписать во время выполнения с помощью опции --entrypoint.

Лучшее использование ENTRYPOINT — установить основную команду запуска контейнера, позволяя запускать этот образ так, как если бы это была его основная команда (а затем использовать CMD в качестве флагов по умолчанию).

```
Entrypoint ["nginx"]
CMD ["--помощь"]
```

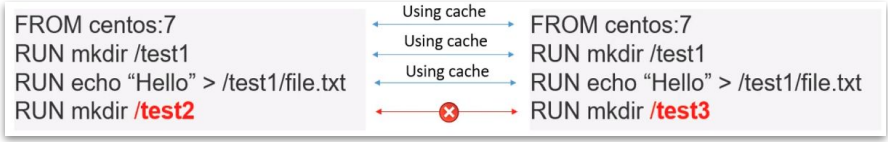
Докерфайл

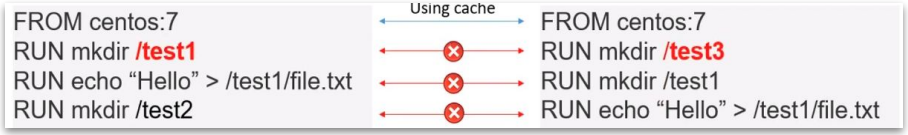
Поведение **CMD, Entrypoint** и **shell** зависят от синтаксиса, как сконфигурированы команды в shell или exec формах:

	No ENTRYPOINT	ENTRYPOINT exec_entry p1_entry	ENTRYPOINT [“exec_entry”, “p1_entry”]
No CMD	*error, not allowed*	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry
CMD [“exec_cmd”, “p1_cmd”]	exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry exec_cmd p1_cmd
CMD [“p1_cmd”, “p2_cmd”]	p1_cmd p2_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry p1_cmd p2_cmd
CMD exec_cmd p1_cmd	/bin/sh -c exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry /bin/sh -c exec_cmd p1_cmd

Докерфайл - рекомендации при сборке

- Собирать образ с как можно наименьшим размером:
  - Использовать правильный базовый образ
  - Не создавать лишних слоев
- Один контейнер - один образ
- Оптимизировать слой для кеширования – порядок инструкций с менее вероятным изменением к более вероятному





Докерфайл - рекомендации при сборке

- Комбинация команд где возможно

```
RUN echo "the first command" && \
    echo "the second command"
```

- Не включать лишних файлов
  - Можно использовать файл конфигурации сборки `dockerignore` чтобы игнорировать ненужные файлы при копировании

```
COPY ./src ./Makefile /src
```

- Использовать мульти ступенчатую сборку

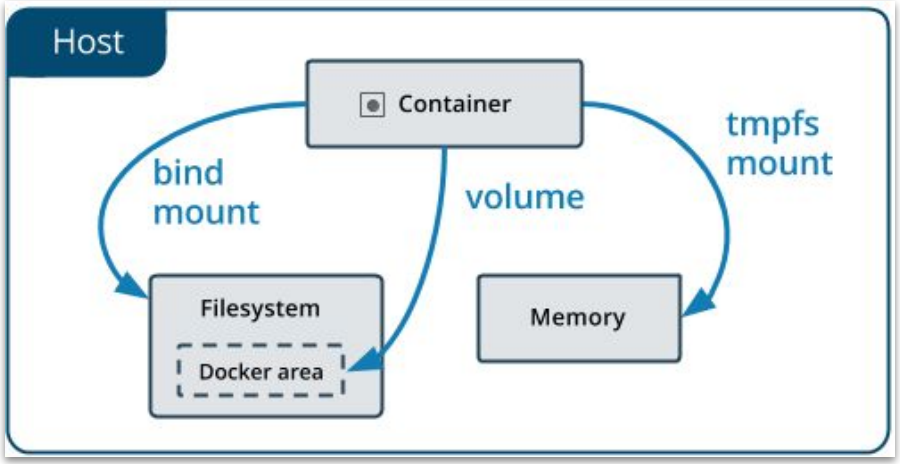
```
# stage 1
FROM alpine as git
RUN apk add git

# stage 2
FROM git as fetch
WORKDIR /repo
RUN git clone https://github.com/your/repository.git .

# stage 3
FROM nginx as site
COPY --from=fetch /repo/docs/ /usr/share/nginx/html
```

Хранение данных

Типы монтирования



> **Volumes находятся внутри системных докер директориях** */var/lib/docker/volumes*

> **Mount Binds** это как символические ссылки - soft links на файловую систему хоста

> **tmpfs** существует только в памяти, поэтому при остановке контейнера данные теряются

Bind mount

- Файл или каталог с хост-машины
- Данные уже существуют на хосте или вам нужно взаимодействовать с ними не из Docker, а с хоста
- Изначально использовался для хранения любых данных
- Совместное использование файлов конфигурации (например, сертификатов SSL, конфигурации приложения и т. д.)
- Доступ к артефактам (исходный код, исполняемые файлы и т. д.)
- Необходимо измените разрешения для папки/файла с тем же UID, что и в контейнере.
- Если bind файл к непустому файлу, он игнорирует любой файл из контейнера и перезаписывает его, однако данные в папках будут объединены.

- Использование Docker CLI:

--volume (-v) <host-path>:<container-path>[:<mode>]
--mount type=bind,source=<host-path>,target=<container-path>,<mode>

```
saltanov@UbuntuPC:~$  echo "Hello world" > hello_world.txt
saltanov@UbuntuPC:~$ docker run -it -v "$PWD"/hello_world.txt:/usr/bin/mkdir ubuntu /bin/bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
01007420e9b0: Pull complete
Digest: sha256:f9d633ff6640178c2d0525017174a688e2c1aef28f0a0130b26bd5554491f0da
Status: Downloaded newer image for ubuntu:latest
root@7f4864bdb162:/# cat /usr/bin/mkdir
Hello world
```

Volumes

- Файл или каталог, абстрагированный от пользователя (управляемый Docker)
- Может находиться в корневом каталоге Docker или в другом месте хранения (на основе Volume Driver).
- Сохранение любых данных приложения, которые могут быть абстрагированы администратором/конечным пользователем (все, что приложение должно хранить, например данные из базы данных) или должны быть использованы совместно между контейнерами.
- Если volume пуст и смонтирован в непустую папку, то содержимое будет скопировано из контейнера.
- Использование Docker CLII:

```
saltanov@UbuntuPC:~$ docker volume create myvol1
myvol1
saltanov@UbuntuPC:~$ docker volume ls
DRIVER    VOLUME NAME
local     myvol1
saltanov@UbuntuPC:~$ docker volume inspect myvol1
[
    {
        "CreatedAt": "2024-02-28T01:24:35+04:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/myvol1/_data",
        "Name": "myvol1",
        "Options": null,
        "Scope": "local"
    }
]
```

docker volume create/rm/ls <volume-name>
--volume (-v) [<volume-name>:]<container-path>[:<mode>]
--mount type=volume,source=<volume-name>,target=<container-path>,<mode>

```
saltanov@UbuntuPC:/$ docker run --name MyJankins1 -v myvol1:/var/jenkins_home -p 8282:8080 -p 40710:8080 jenkins/jenkins
Running from: /usr/share/jenkins/jenkins.war
webroot: /var/jenkins_home/war
2024-02-27 22:27:37.509+0000 [id=1]     INFO    winstone.Logger#logInternal: Beginning extraction from war file
```

```
saltanov@UbuntuPC:/$ docker run --name MyJankins2 --volumes-from MyJankins1 --name MyJankins3 -p 8283:8080 -p 40711:8080 jenkins/jenkins
Running from: /usr/share/jenkins/jenkins.war
webroot: /var/jenkins_home/war
2024-02-27 22:29:02.375+0000 [id=1]     INFO    winstone.Logger#logInternal: Beginning extraction from war file
```

tmpfs

- Данные хранятся в памяти на хоста Docker (не на диске)
- Не сохраняет никаких данных
- Не может совместно использоваться контейнерами
- Полезно из соображений безопасности (данные не сохраняются на жестком диске)
- Или для производительности (так как быстрее работать с памятью)

- Использование Docker CLI:

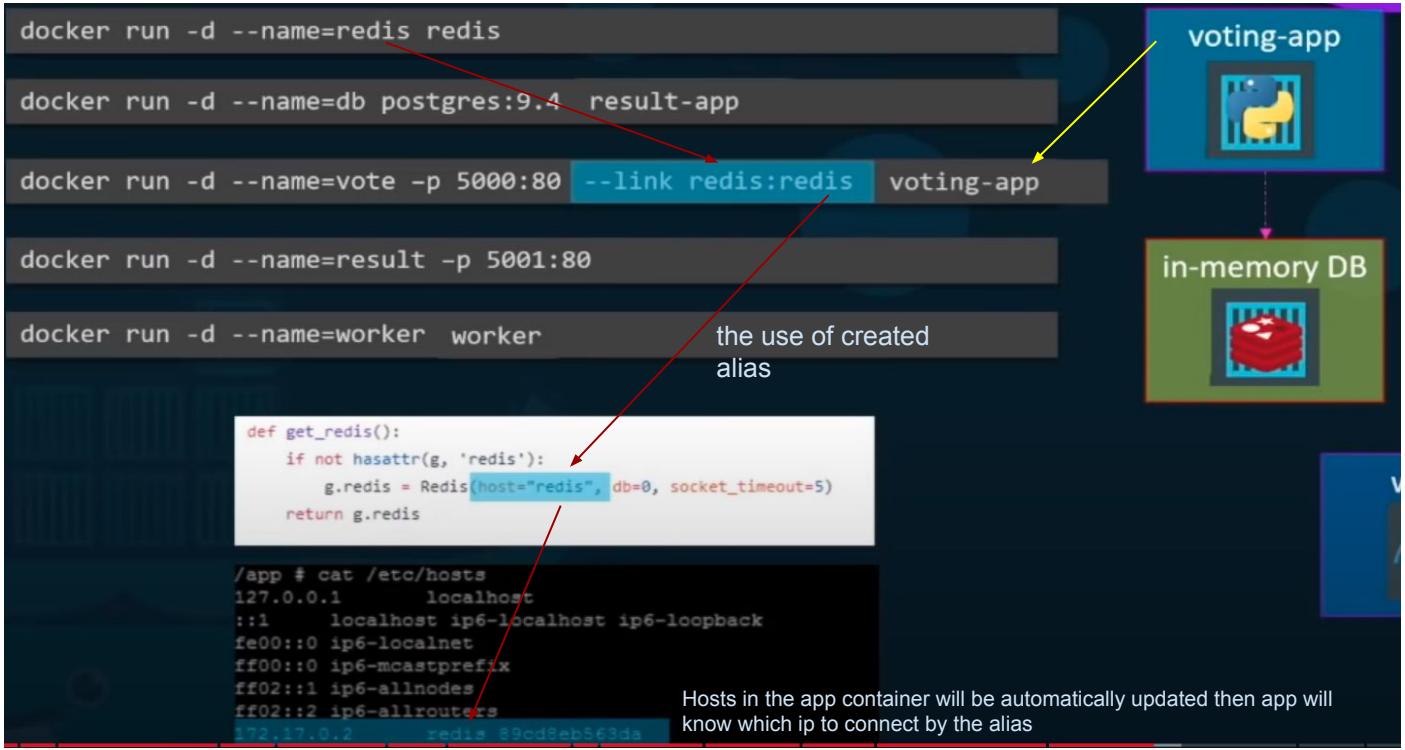
```
--tmpfs <container-path>
--mount type=tmpfs,destination=app[,options]
```

Сетевое управление

Сетевые докер драйверы

- **bridge (используется по умолчанию создает default bridge network)**
  - Docker bridge driver автоматически устанавливает правила на хост-компьютере, чтобы контейнеры в разных сетях bridge не могли напрямую взаимодействовать друг с другом.
  - контейнеры в **default bridge network** могут обращаться друг с другом только по IP-адресам, если только не используется опцию **--link**, которая считается устаревшей. В сети bridge которая создается отдельно пользователем контейнеры могут разрешать IP адрес друг друга **по имени или по alias**.
- **host**
  - контейнер разделяет сетевое пространство хоста - network namespace и не получает отдельный IP-адрес.
  - при отсутствии IP-адреса при использовании сети в режиме host сопоставление портов - port mapping не имеет значения, а опция -p, --publish игнорируется
- **none**

Сетевые докер драйверы



Что еще необходимо для работы с контейнерами?

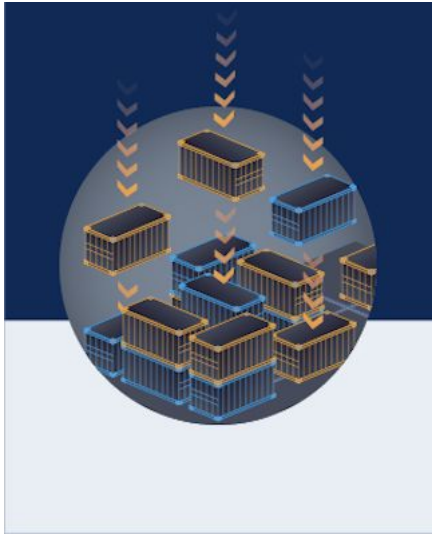
Элементарная оркестровка контейнеров

Декларативное определение сервисов

Храните конфигурацию окружений (сеть, данные) рядом с сервисами

Храните связанные контейнеры вместе

Единый жизненный цикл для связанных ресурсов



Docker Compose

Преимущества и недостатки Docker Compose

+ Простой синтаксис
+ Простота управления (никаких специальных сервисов, демонов и т.п.)
+ Позволяет использовать декларативную конфигурацию вместо множества команд.
+ Позволяет управлять несколькими ресурсами одновременно (разделять жизненный цикл)

- Не настоящий «инструмент оркестровки»
- Не контролирует контейнеры
- Ограничено одним хостом

Installation

- Installation https://docs.docker.com/compose/install/
- Syntax reference https://docs.docker.com/compose/compose-file/compose-file-v3/
- Относится частью управления Docker CLI

Синтаксис

- Каждый контейнер может быть описан полем services.
- Если нам нужно собрать из dockerfile, используется build, в противном случае используйте поле image, чтобы указать уже собранный образ

```
version: "3.9"
services:
  web:
    build: .
    ports:
      - "8000:5000"
    volumes:
      - .:/code
    environment:
      FLASK_ENV: development
  redis:
    image: "redis:alpine"
```

Синтаксис

- `$docker compose <build|up>`
- **build** может указывать на директорию или быть словарем
  - **context** может быть путь или URL на dockerfile
  - **dockerfile** может лежать отдельно от контекста и быть указан
  - **image** поле может быть пропущено, если указано то добавить тег к собранному образу

```
services:
  frontend:
    image: awesome/webapp
    build: ./webapp

  backend:
    image: awesome/database
    build:
      context: backend
      dockerfile: ../backend.Dockerfile

  custom:
    build: ~/custom
```

Синтаксис

- Пример как можно переиспользовать уже собранный образ для другого контейнера
- **depends_on** - зависимость сборки от другого

```
services:
  es-master:
    build: ./elasticsearch
    image: porter/elasticsearch
    ports:
      - "9200:9200"
    container_name: es_master

  es-node:
    image: porter/elasticsearch
    depends_on:
      - es-master
    ports:
      - "9200"
    command: elasticsearch --discovery.zen.ping.unicast.hosts=es_master
```

Если указать только - "9200", Docker автоматически сопоставляет порт контейнера 9200 с эфемерным портом на хосте. Это также называется динамическим назначением порта.
Фактический порт на хосте будет отличаться, и Docker выбирает его из доступного диапазона.

Синтаксис - сетевая конфигурация

- Если не указывать поле networks, тогда для каждого контейнера в service – создается default bridge network
- Внутри поля можно кастомизировать сетевые конфигурации (тип драйвера и проч.)
- Отдельно созданные сети - custom networks могут быть указаны индивидуально для каждого контейнера
- Контейнера для индивидуальных сетей, внутри доступны по имени контейнера
- `external` - указывает на уже созданную сеть - не создается внутри

```
<...>

networks:
  default:
    external:
      name: my-pre-existing-network
```

Хранение данных

```yaml
version: "3.9"
services:
  frontend:
    image: node:lts
    volumes:
      - myapp:/home/node/app
volumes:
  myapp:
```

```yaml
version: "3.9"
services:
  frontend:
    image: node:lts
    volumes:
      - myapp:/home/node/app
volumes:
  myapp:
    external: true
```

- **`external: true`** - если установлено значение true, указывает, что этот volume был создан вне Compose. docker-compose up не пытается создать его и выдает ошибку, если он не существует.

Хранение данных

```yaml
version: "3.9"
services:
  web:
    image: nginx:alpine
    volumes:
      - type: volume
        source: mydata
        target: /data
        volume:
          nocopy: true
      - type: bind
        source: ./static
        target: /opt/app/static

  db:
    image: postgres:latest
    volumes:
      - "/var/run/postgres/postgres.sock:/var/run/postgres/postgres.sock"
      - "dbdata:/var/lib/postgresql/data"

volumes:
  mydata:
  dbdata:
```

- Пример показывает, как определить типы volume и сконфигурировать разными способами
- Именованные volume должны быть указаны в поле глобальных volumes

Основные команды docker compose

- `$docker-compose <up|down>` – останавливает и удаляет контейнеры, сеть, volumes и образы, созданные с помощью up.
- **Networks и volumes, определенные как внешние - external, никогда не удаляются**
- `$docker-compose <start|stop>` – останавливает запущенные контейнеры, не удаляя их. Их можно запустить снова с помощью команды `docker-compose start`.
- `$docker-compose restart` – перезапускает все остановленные и работающие контейнеры.
- Команды которые описаны выше можно также сделать не для всех а для индивидуальных контейнеров
  - `$docker-compose up -d <имя_контейнера>`