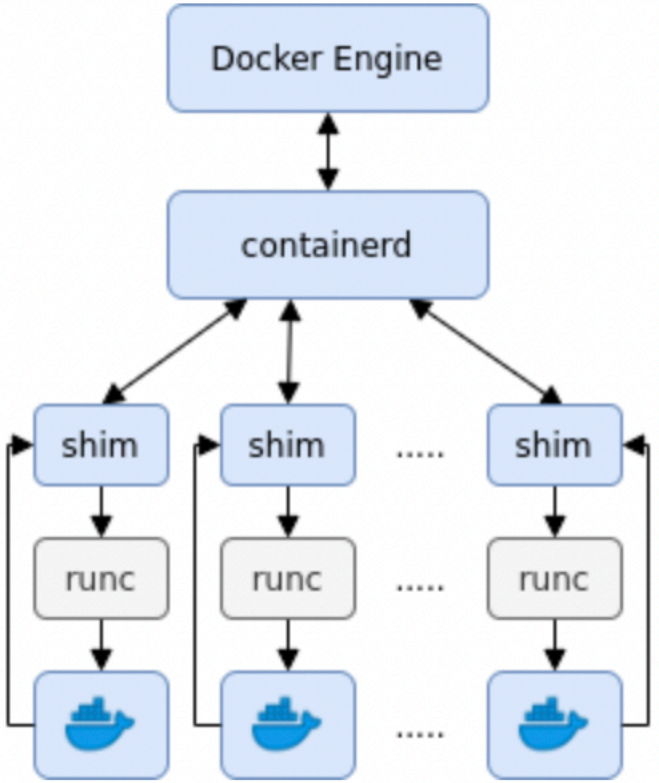
NFE Networking For Everyone

Docker

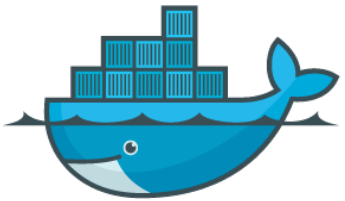
Архитектура Docker

- `Docker` – взаимодействие с пользователем (dockerd, cli, api).
- `containerD` – high-level runtime для управления образами (скачивание и размещение в registry), управление сетью, volumes, метрики Prometheus, и прочее
- `runc` – low-level runtime для запуска контейнеров, умеет только создавать и запускать контейнеры, но не управлять образами
- `shim` – прослойка, позволяющая отключить containerD демон от контейнера после его запуска



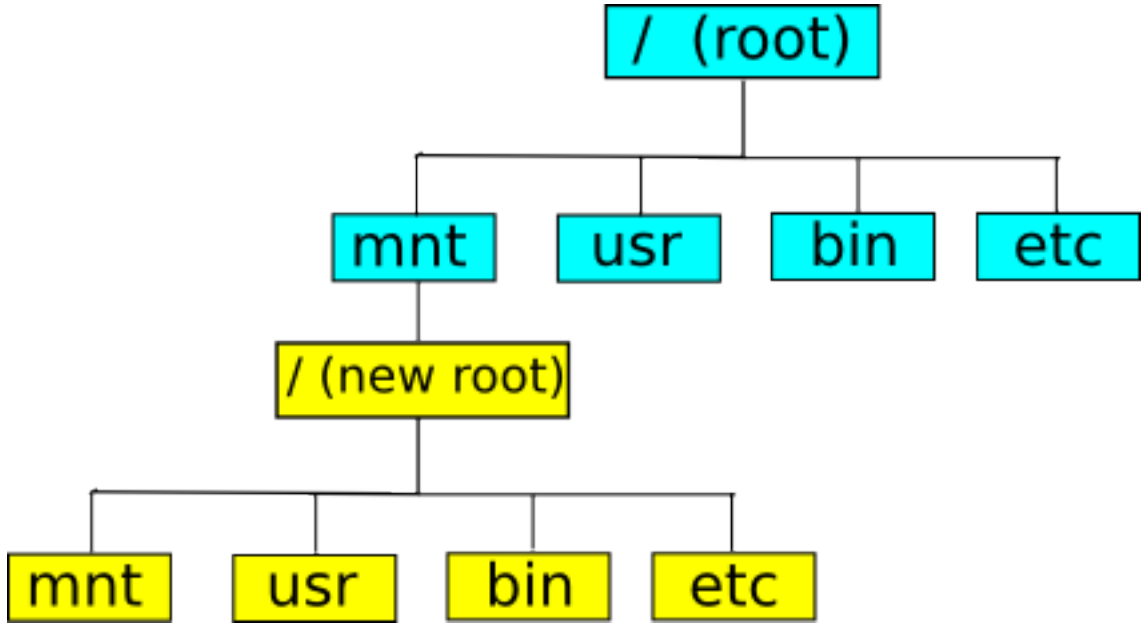
Архитектура Docker

- *capabilities* – атрибуты ядра, которые позволяют выполнять те или иные действия. Например: монтирование файловой системы, изменения PID процесса, замена MAC, установка времени и прочее.
- *namespaces* – абстракция над ресурсами системы. Несколько типов: pid, net, ipc, mnt, uts, user, cgroup.
- *cgroup* – способ управления (ограничение использования) ресурсами системы. Состоит из ядра и подсистем (ns, cpu, memory, devices, …)
- *veth* – виртуальные ethernet адаптеры
- …



namespace

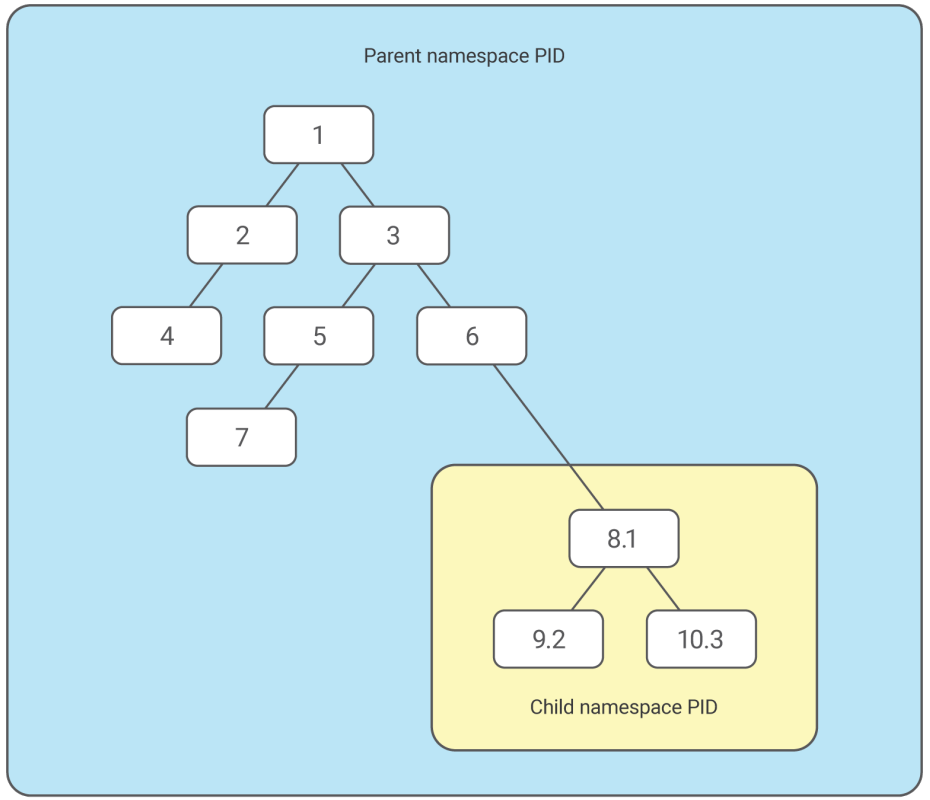
- *chroot()* – смена корневого каталога, первая попытка изолировать ресурсы



namespace (PID)

- Изначально поддерживалось только одно дерево процессов
- PID 1 - родительский (корневой) процесс, обычно systemd. Если процесс с PID 1 завершается, то весь namespace удаляется

- namespace позволяет создать отдельное ответвление с собственным PID 1 (корневым) процессом
- Корневой и дочерние процессы в этом ответвлении будут иметь два PID: одно в “дочернем” namespace, другое в родительском
- Процессы в дочернем дереве не знаю о процессах родительского дерева и никак не могут ними взаимодействовать
- Процессы родительского дерева знают о процессах дочернего и могут с ними взаимодействовать



namespace (PID)

- clone, unshare - различные механизмы создания нового namespace (docker использует unshare)

```
alexigna@ansible:~$ sudo unshare --pid --fork --mount-proc /bin/bash
root@ansible:/home/alexigna# ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.1   7536  3692 pts/2    S    16:03   0:00 /bin/bash
root           8  0.0  0.0   9812  1652 pts/2    R+   16:03   0:00 ps aux
```

```
alexigna@ansible:~$ ps aux | grep /bin/bash
root        1619  0.0  0.2  13364  4652 pts/1    S+   16:03   0:00 sudo unshare -fp --mount-proc /bin/bash
root        1620  0.0  0.0  13364   672 pts/2    Ss   16:03   0:00 sudo unshare -fp --mount-proc /bin/bash
root        1621  0.0  0.0   5608   792 pts/2    S    16:03   0:00 unshare -fp --mount-proc /bin/bash
root        1622  0.0  0.1   7536  3692 pts/2    S+   16:03   0:00 /bin/bash
alexigna    1643  0.0  0.0   6420  1828 pts/0    S+   16:10   0:00 grep --color=auto /bin/bash
```

```
alexigna@ansible:~$ sudo ls -l /proc/$$/ns
total 0
lrwxrwxrwx 1 alexigna alexigna 0 Sep 14 16:15 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 alexigna alexigna 0 Sep 14 16:15 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 alexigna alexigna 0 Sep 14 16:15 mnt -> 'mnt:[4026531841]'
lrwxrwxrwx 1 alexigna alexigna 0 Sep 14 16:15 net -> 'net:[4026531840]'
lrwxrwxrwx 1 alexigna alexigna 0 Sep 14 16:15 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 alexigna alexigna 0 Sep 14 16:15 time -> 'time:[4026531834]'
lrwxrwxrwx 1 alexigna alexigna 0 Sep 14 16:15 user -> 'user:[4026531837]'
lrwxrwxrwx 1 alexigna alexigna 0 Sep 14 16:15 uts -> 'uts:[4026531838]'
```

```
alexigna@ansible:~$ sudo ls -l /proc/1622/ns
total 0
lrwxrwxrwx 1 root root 0 Sep 14 16:17 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Sep 14 16:17 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0 Sep 14 16:17 mnt -> 'mnt:[4026532224]'
lrwxrwxrwx 1 root root 0 Sep 14 16:17 net -> 'net:[4026531840]'
lrwxrwxrwx 1 root root 0 Sep 14 16:17 pid -> 'pid:[4026532225]'
lrwxrwxrwx 1 root root 0 Sep 14 16:17 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Sep 14 16:17 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Sep 14 16:17 uts -> 'uts:[4026531838]'
```

namespace (PID)

Docker предоставляет обертку над стандартными механизмами ядра linux

```
alexigna@ansible:~$ sudo docker run --rm -d busybox:latest ping 8.8.8.8
```

```
alexigna@ansible:~$ pstree -p
systemd(1)─┬─ModemManager(734)─┬─{ModemManager}(758)
           │                   └─{ModemManager}(763)
           ├─agetty(718)
           ├─containerd(700)─┬─{containerd}(745)
           │                 ├─{containerd}(748)
           │                 └─{containerd}(801)
           ├─containerd-shim(1939)─┬─ping(1961)
           │                       ├─{containerd-shim}(1940)
           │                       └─{containerd-shim}(1948)
           ├─cron(684)
           ├─dbus-daemon(685)
           ├─dockerd(826)─┬─{dockerd}(835)
           │              ├─{dockerd}(843)
           │              └─{dockerd}(1054)
```

```
alexigna@ansible:~$ ps aux | grep 1961
root        1961  0.0  0.0   3988  1720 ?        Ss   16:48   0:00 ping 8.8.8.8
alexigna    1985  0.0  0.0   6420  1860 pts/1    S+   16:50   0:00 grep --color=auto 1961
```

```
alexigna@ansible:~$ sudo ls -l /proc/$$/ns
total 0
lrwxrwxrwx 1 alexigna alexigna 0 Sep 14 16:15 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 alexigna alexigna 0 Sep 14 16:15 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 alexigna alexigna 0 Sep 14 16:15 mnt -> 'mnt:[4026531841]'
lrwxrwxrwx 1 alexigna alexigna 0 Sep 14 16:15 net -> 'net:[4026531840]'
lrwxrwxrwx 1 alexigna alexigna 0 Sep 14 16:15 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 alexigna alexigna 0 Sep 14 16:15 time -> 'time:[4026531834]'
lrwxrwxrwx 1 alexigna alexigna 0 Sep 14 16:15 user -> 'user:[4026531837]'
lrwxrwxrwx 1 alexigna alexigna 0 Sep 14 16:15 uts -> 'uts:[4026531838]'
```

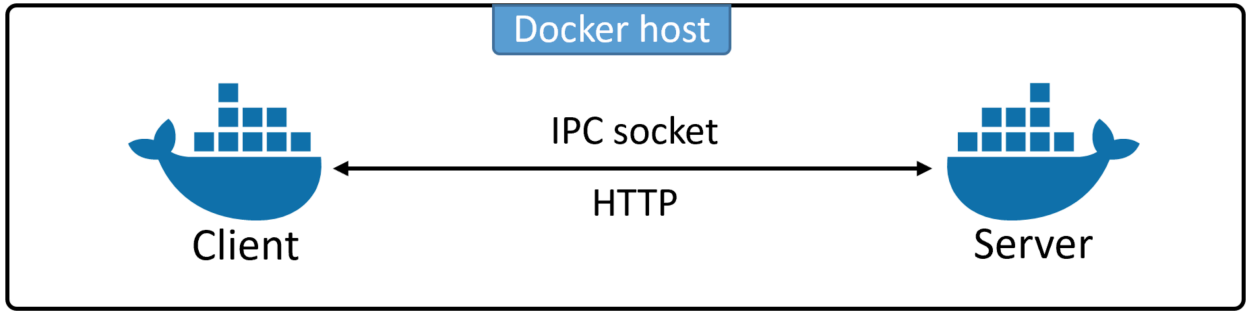
```
alexigna@ansible:~$ sudo ls -l /proc/1961/ns
total 0
lrwxrwxrwx 1 root root 0 Sep 14 16:49 cgroup -> 'cgroup:[4026532287]'
lrwxrwxrwx 1 root root 0 Sep 14 16:49 ipc -> 'ipc:[4026532228]'
lrwxrwxrwx 1 root root 0 Sep 14 16:49 mnt -> 'mnt:[4026532226]'
lrwxrwxrwx 1 root root 0 Sep 14 16:48 net -> 'net:[4026532231]'
lrwxrwxrwx 1 root root 0 Sep 14 16:49 pid -> 'pid:[4026532230]'
lrwxrwxrwx 1 root root 0 Sep 14 16:49 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Sep 14 16:49 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Sep 14 16:49 uts -> 'uts:[4026532227]'
```

```
alexigna@ansible:~$ sudo nsenter -t 1961 -m -u -i -n -p /bin/sh
/ # ps aux
PID   USER     TIME  COMMAND
    1 root      0:00 ping 8.8.8.8
   14 root      0:00 /bin/sh
   15 root      0:00 ps aux
```

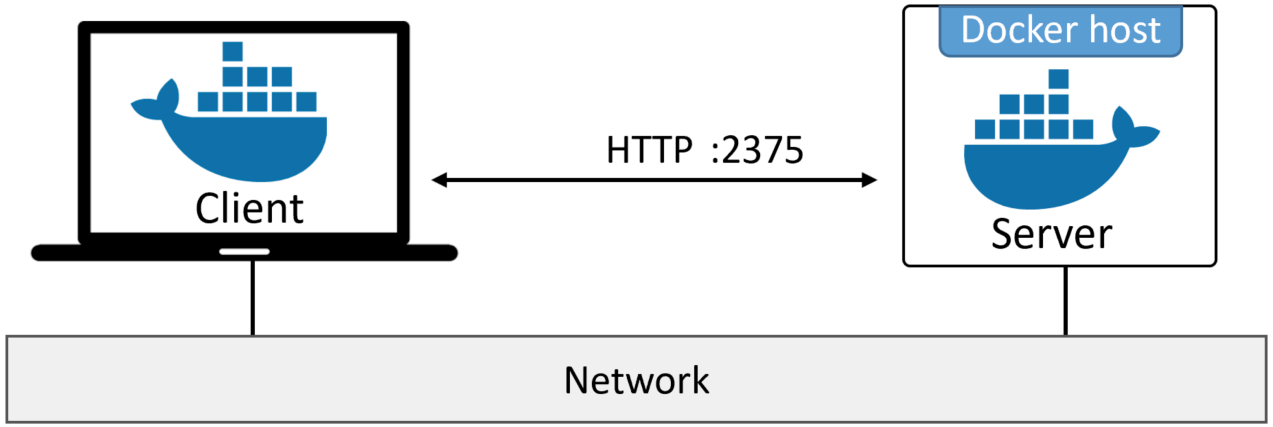
NETWORKING FOR EVERYONE

Взаимодействие клиента и демона

По-умолчанию через IPC посредством HTTP



Есть возможность коммуникации через сеть



Взаимодействие клиента и демона

Взаимодействие с dockerd напрямую через отправку команд в сокет curl'ом

Вывод версии Docker'а

```
alexigna@ansible:~$ sudo curl \
 --silent \
 --unix-socket /var/run/docker.sock http://localhost/version | jq
{
  "Platform": {
    "Name": "Docker Engine - Community"
  },
  "Components": [
    {
      "Name": "Engine",
      "Version": "24.0.6",
      "Details": {
        "ApiVersion": "1.43",
        "Arch": "arm64",
        "BuildTime": "2023-09-04T12:31:57.000000000+00:00",
        "Experimental": "false",
        "GitCommit": "1a79695",
        "GoVersion": "go1.20.7",
        "KernelVersion": "5.15.0-83-generic",
        "MinAPIVersion": "1.12",
        "Os": "linux"
      }
    },
```

Список локальных образов (docker image ls)

```
alexigna@ansible:~$ sudo curl \
 --silent \
 --unix-socket /var/run/docker.sock \
 http://localhost/images/json | jq
[
  {
    "Containers": -1,
    "Created": 1694068261,
    "Id": "sha256:91582cfffc2d0daa6f42adb6fb74665a047310f76a28e9ed5b0185a2d0f362a6",
    "Labels": {
      "maintainer": "NGINX Docker Maintainers <docker-maint@nginx.com>"
    },
    "ParentId": "",
    "RepoDigests": [
      "nginx@sha256:6926dd802f40e5e7257fded83e0d8030039642e4e10c4a98a6478e9c6fe06153"
    ],
    "RepoTags": [
      "nginx:latest"
    ],
    "SharedSize": -1,
    "Size": 192063326,
    "VirtualSize": 192063326
  },
```

https://docs.docker.com/engine/api/v1.42/

Взаимодействие клиента и демона

Создание контейнера с именем curl-test из образа nginx:latest

```
alexigna@ansible:~$ sudo curl \
 -X POST \
 --silent \
 --unix-socket /var/run/docker.sock \
 -d '{ "Image": "nginx:latest", "PortBindings": { "80/tcp": [{ "HostPort":"8080" }] } }' \
 -H 'Content-Type: application/json' \
 "http://localhost/containers/create?name=curl-test" | jq
{
  "Id": "5e2ef5eb9125eb76066170aba4e27bb853ca65786685332799b03d7fba42a330",
  "Warnings": []
}
```

Созданный (но не запущенный) контейнер

```
alexigna@ansible:~$ sudo docker ps -a
CONTAINER ID   IMAGE         COMMAND                  CREATED              STATUS      NAMES
66d19436aad4   nginx:latest  "/docker-entrypoint.…"   About a minute ago   Created     curl-test
```

Запуск контейнера

```
alexigna@ansible:~$ sudo curl \
  -X POST \
  --unix-socket /var/run/docker.sock \
  -H "Content-Type: application/json" \
  "http://localhost/containers/curl-test/start"
```

Взаимодействие клиента и демона

Работающий контейнер

```
alexigna@ansible:~$ sudo docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                                   NAMES
66d19436aad4   nginx:latest   "/docker-entrypoint.…"   3 minutes ago   Up 13 seconds   0.0.0.0:8080->80/tcp, :::8080->80/tcp   curl-test

alexigna@ansible:~$ curl 127.0.0.1:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Остановка и удаление контейнера

```
alexigna@ansible:~$ sudo curl -X POST --unix-socket /var/run/docker.sock http://localhost/containers/curl-test/stop
alexigna@ansible:~$ sudo curl -X DELETE --unix-socket /var/run/docker.sock http://localhost/containers/curl-test
```

Работа с образами

`docker image ls` - просмотр образов на локальной машине

`docker image tag SRC_name[:tag] TGT_name[:tag]` - проставить теги образу

`docker image push <name>[:<tag>]` - загрузить образ в репозиторий

`docker image pull <name>[:<tag>]` - скачать образ из репозитория

`docker image rm <name>:<tag>` - удалить образ (docker rmi)

`docker image save/load` - сохраняет и загружает образ в/из локального файла

Dockerfile

- Описывает все шаги, необходимые для создания образа
- Каждая строка в файле создает новый слой образа
- Инструкции обрабатываются по очереди сверху вниз
- Основные инструкции: FROM, COPY, CMD, ENTRYPOINT, EXPOSE, ENV, LABEL, ARG, RUN
- Dockerfile - имя файла (D большая)
- `docker build .` - минимальная команда для сборки образа
- Всегда начинается с FROM или ARG.

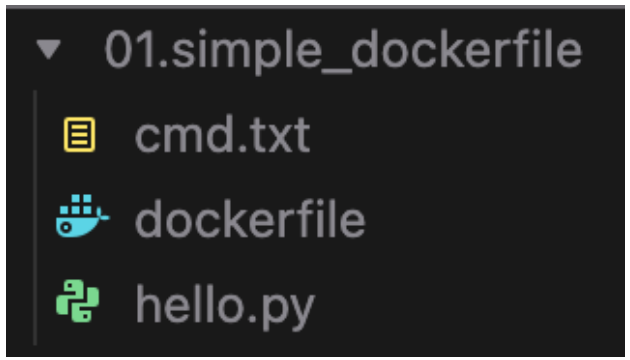
Dockerfile

dockerfile

```
FROM python:3.10-slim

COPY ./hello.py .

CMD [ "python", "hello.py" ]
```

hello.py

```
print("hello from container")
```




```
alexigna@mbp-alexigna % docker build .
[+] Building 0.1s (7/7) FINISHED
docker:desktop-linux
 => [internal] load .dockerignore                                                0.0s
 => => transferring context: 2B                                                  0.0s
 => [internal] load build definition from dockerfile                             0.0s
 => => transferring dockerfile: 99B                                              0.0s
 => [internal] load metadata for docker.io/library/python:3.10-slim              0.0s
 => [internal] load build context                                                0.0s
 => => transferring context: 126B                                                0.0s
 => CACHED [1/2] FROM docker.io/library/python:3.10-slim                         0.0s
 => [2/2] COPY . .                                                               0.0s
 => exporting to image                                                           0.0s
 => => exporting layers                                                          0.0s
 => => writing image sha256:b40dcf4c5aaff97bab758ce7454e449be18090b8b1efe22380dbe9c775cf74ad 0.0s

alexigna@mbp-alexigna % docker image ls
REPOSITORY   TAG         IMAGE ID       CREATED                  SIZE
<none>       <none>      e23e2a485724   Less than a second ago   170MB
python       3.10-slim   e0cbf388c936   4 weeks ago              170MB

alexigna@mbp-alexigna % docker run --rm e23e2a485724
hello from container
```

Пример Dockerfile

- `FROM <name>` - определяет базовый образ
- Если локального его нет, то при сборке он автоматически скачается с репозитория
- `COPY <src> <dst>` - копирование файлов
- `CMD [ <cmd_list> ]`- команда, которая будет исполнена при запуске контейнера

dockerfile

```
FROM python:3.10-slim
COPY ./hello.py .
CMD [ "python", "hello.py" ]
```

Инструкция LABEL

- Задает метаданные
- Может быть несколько инструкций
- Используются двойные кавычки
- Метки наследуются из родительского образа

dockerfile

```
LABEL org.opencontainers.image.authors="Alexey Pustovalov <alexey.pustovalov@zabbix.com>" \
      org.opencontainers.image.licenses="GPL v2.0" \
      org.opencontainers.image.title="Zabbix build base" \
      org.opencontainers.image.url="https://zabbix.com/" \
      org.opencontainers.image.vendor="Zabbix LLC" \
      org.opencontainers.image.version="${ZBX_VERSION}"
```

Инструкция ENV

- Задает переменные среды
- Значения видны для следующих инструкций
- Присутствуют пока контейнер работает, можно менять в процессе работы

dockerfile

```
ENV TERM=xterm \
    ZBX_VERSION=${ZBX_VERSION} \
    PATH=/usr/lib/go-1.18/bin:$PATH

ENV TERM=xterm
ENV ZBX_VERSION=${ZBX_VERSION}
ENV PATH=/usr/lib/go-1.18/bin:$PATH
```

Инструкция RUN

- Выполняет команду при сборке образа
- Самое частое использование – установка пакетов
- Имеет две формы записи
  - RUN <command>
  - RUN [“executable”, “param1”, “param2”]

dockerfile

```
RUN apt update && \
    apt install -y \
    wget \
    supervisor \
    unit=1.30.0-1~jammy \
    unit-python3.10=1.30.0-1~jammy

RUN /opt/netbox/venv/bin/pip cache purge
```

Инструкция ARG

- Позволяет задать переменные, которые передаются во время сборки образа
- Может быть несколько инструкций
- Можно задать значение по умолчанию
- Не то же самое, что ENV

dockerfile

```
ARG python_version=3.10-slim

FROM python:${python_version}

CMD [ "python", "--version" ]
```

```
alexigna@mbp-alexigna % docker build -t arg-test --build-arg python_version=3.11-slim .
[+] Building 0.0s (5/5) FINISHED
…
What's Next?
  View summary of image vulnerabilities and recommendations → docker scout quickview

alexigna@mbp-alexigna % docker run --rm arg-test
Python 3.11.4
```

```
alexigna@mbp-alexigna % docker build -t arg-test  .
[+] Building 0.0s (5/5) FINISHED
…
What's Next?
  View summary of image vulnerabilities and recommendations → docker scout quickview

alexigna@mbp-alexigna % docker run --rm arg-test
Python 3.10.12
```

Инструкция ENTRYPOINT

- Позволяет выполнить определенную команду во время исполнения контейнера
- Не может быть переопределена

cat.py

```
print(“meaw-meaw")
```

dog.py

```
print(“woof-woof")
```

dockerfile

```
FROM python:3.10-slim

COPY ./*.py .

ENTRYPOINT [ "python" ]

CMD [ “cat.py" ]
```

```
alexigna@mbp-alexigna % docker build -t entrypoint_example  .
[+] Building 0.1s (7/7) FINISHED

alexigna@mbp-alexigna % docker run --rm entrypoint_example
meaw-meaw

alexigna@mbp-alexigna % docker run --rm entrypoint_example dog.py
woof-woof
```

Инструкция WORKDIR

- Устанавливает рабочую директорию для RUN, CMD, ENTRYPOINT, COPY и ADD
- По-умолчанию /

dockerfile

```
ARG python_version=3.10-slim
FROM python:${python_version}
COPY . /app
WORKDIR /app
CMD [ “python", “hello.py” ]
```

Инструкция EXPOSE

- Проброс порта в контейнер
- Информационная нагрузка: сама по себе ничего не открывает или прокидывает в контейнер, а только говорит “приложение будет слушать порт <num>”
- Можно не указывать, на функционал не влияет, назначение - документирование

dockerfile

```
ARG python_version=3.10-slim

FROM python:${python_version}

…

EXPOSE 8080
EXPOSE 80/tcp
EXPOSE 80/udp
```

Инструкция VOLUME

- Создает анонимный volume для постоянного хранения данных

dockerfile

```
FROM busybox

RUN mkdir /var/logs
VOLUME /var/logs

COPY ./redirect.sh /redirect.sh
RUN chmod +x /redirect.sh

CMD [ "/redirect.sh", "/etc/hostname", "/var/logs/app.log" ]
```

```
alexigna@mbp-alexigna 05.volume % docker build -t vlm-test .
alexigna@mbp-alexigna 05.volume % docker run --name vlm-test001 vlm-test

alexigna@mbp-alexigna 05.volume % docker inspect vlm-test001
…
    "Mounts": [
        {
            "Type": "volume"
            "Name": "53bd18b21c99e5c512804c05e4ca29fddaf01fb69220a1653227fdd88623026c"
            "Source": "/var/lib/docker/volumes/53bd18b21c99e5c512804c05e4ca29fddaf01fb69220a1653227fdd88623026c/_data"
            "Destination": "/var/logs"
            "Driver": "local"
            "Mode": ""
            "RW"
            "Propagation": ""
        }
    ],
```

Файл .dockerignore

- Прежде чем docker CLI отправляет контекст демону, проверяется файл .dockerignore в корневой директории контекста
- Файлы и директории, попадающие под шаблоны игнорирования, исключаются
- Позволяет исключить случайную отправку ненужных файлов в образ

Работа с контейнерами

- После создания/загрузки образа на его основе создается и запускается контейнер:
  - docker create (создать)
  - docker start (запустить уже созданный)
  - docker run (создать и запустить)
- После выполнения программы контейнер завершается самостоятельно. Но можно это сделать принудительно командами
  - docker stop
  - docker kill
- Для удаления контейнера:
  - docker rm

docker run

```
docker run \
 --rm \
 --name c1 \
 -d \
 -p 8080:80/tcp \
 -p 8443:443/tcp \
 -v ./:/usr/share/nginx/html:ro \
 -e APP_NAME="my website" \
 -e APP_VERSION="0.0.2" \
 nginx:latest
```

`--rm` - удаление контейнера после завершения

`--name` - имя контейнера

`-d` - запуск контейнера в фоне

`-p` - проброс портов в контейнер, <host>:<container>

`-v` - подключение volume

`-e` - задание переменных окружения

`nginx:latest` - образ, на базе которого запускается контейнер

```
alexigna@mbp-alexigna % docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                                                  NAMES
af61960c69ee   nginx:latest   "/docker-entrypoint.…"   2 seconds ago   Up 2 seconds   0.0.0.0:8080->80/tcp, 0.0.0.0:8443->443/tcp   c1
```

docker run (limits)

```
alexigna@ansible:~$ sudo docker run -it --rm \
jfleach/docker-arm-stress-ng --cpu 2 --vm 8  --timeout 15s –metrics-brief

stress-ng: info: [1] dispatching hogs: 2 cpu, 8 vm
stress-ng: info: [1] successful run completed in 15.10s
stress-ng: info: [1] stressor      bogo ops real time  usr time  sys time   bogo ops/s   bogo ops/s
stress-ng: info: [1]                          (secs)    (secs)    (secs)   (real time) (usr+sys time)
stress-ng: info: [1] cpu                712     15.09      6.71      0.20        47.19       103.04
```

```
alexigna@ansible:~$ sudo docker stats --format "table {{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}\t{{.MemPerc}}"

NAME                CPU %     MEM USAGE / LIMIT     MEM %
peaceful_chatelet   175.21%   1.627GiB / 1.921GiB   84.70%
```

```
alexigna@ansible:~$ sudo docker run --cpus 1 --memory 256MB -it --rm \
jfleach/docker-arm-stress-ng --cpu 2 --vm 8 --timeout 15s --metrics-brief

stress-ng: info: [1] dispatching hogs: 2 cpu, 8 vm
stress-ng: info: [1] successful run completed in 15.10s
stress-ng: info: [1] stressor      bogo ops real time  usr time  sys time   bogo ops/s   bogo ops/s
stress-ng: info: [1]                          (secs)    (secs)    (secs)   (real time) (usr+sys time)
stress-ng: info: [1] cpu                461     15.07      4.59      0.56        30.59        89.51
```

```
alexigna@ansible:~$ sudo docker stats --format "table {{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}\t{{.MemPerc}}"

NAME             CPU %     MEM USAGE / LIMIT   MEM %
loving_babbage   100.12%   256MiB / 256MiB     99.99%
```

docker logs, attach

```
alexigna@mbp-alexigna % docker logs -f --since=10m c1
172.17.0.1 - - [18/Jul/2023:11:38:54 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/
605.1.15 (KHTML, like Gecko) Version/16.4.1 Safari/605.1.15" "-"
```

`-f` - follow опция

`--since=10m` - логи за последние 10 минут

```
alexigna@mbp-alexigna % docker attach c1

^C2023/07/18 11:41:21 [notice] 1#1: signal 2 (SIGINT) received, exiting
2023/07/18 11:41:21 [notice] 30#30: exiting
…
2023/07/18 11:41:21 [notice] 1#1: exit
alexigna@mbp-alexigna 06.nginx %
```

docker exec

```
alexigna@mbp-alexigna % docker exec -it c1 /bin/bash
root@8908628d5413:/#
root@8908628d5413:/# top

top - 11:47:54 up 2 days,  4:11,  0 user,  load average: 0.11, 0.05, 0.04
Tasks:   8 total,   1 running,   7 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.2 us,  0.3 sy,  0.0 ni, 99.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   7851.6 total,   4820.9 free,   1090.5 used,   2447.2 buff/cache
MiB Swap:   1024.0 total,   1024.0 free,      0.0 used.   6761.0 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    1 root      20   0   11136   7104   5988 S   0.0   0.1   0:00.03 nginx
   29 nginx     20   0   11592   2944   1284 S   0.0   0.0   0:00.00 nginx
   30 nginx     20   0   11592   2944   1284 S   0.0   0.0   0:00.00 nginx
   31 nginx     20   0   11592   2944   1284 S   0.0   0.0   0:00.00 nginx
   32 nginx     20   0   11592   2944   1284 S   0.0   0.0   0:00.00 nginx
```

`-i -t` - интерактивный режим (stdin не закрывается) с созданием tty терминала. Так же используются вместе как -it

`/bin/bash` - что запускаем в терминале

docker stop/start, kill, rm

```
alexigna@mbp-alexigna 06.nginx % docker stop c1
alexigna@mbp-alexigna 06.nginx % docker start c1
alexigna@mbp-alexigna 06.nginx % docker kill c1
alexigna@mbp-alexigna 06.nginx % docker rm c1
```

`stop` - gracefull остановка контейнера через SIGTERM. Если через 10с контейнер не остановлен, тогда SIGKILL

`start` - возобновление работы контейнера

`kill` - остановка контейнера через SIGKILL

`rm` - удаление контейнера

Volume

- Папка (bind mounts) или том (volume) монтируются к файловой системе контейнера
- Подключение может быть в режиме ReadOnly или ReadWrite
- Тома могут быть анонимными и именованными

```
alexigna@mbp-alexigna % docker run \
…
 -v ./my/host/path:/usr/share/nginx/html:ro \
```

монтируем папку с относительным путём ./my/host/path в контейнер в папку /usr/share/nginx/html. Режим ReadOnly.

```
alexigna@mbp-alexigna % docker volume create my-vol-001
my-vol-001

alexigna@mbp-alexigna % docker run \
…
 -v my-vol-001:/var/log:rw \
…
```

создание тома и его подключение

```
alexigna@mbp-alexigna % docker run \
…
 -v :/var/log:rw \
…
```

неименованный том

Volume

- Могут использоваться как постоянное хранилище для данных или как средство обмена данными между контейнерами

```
alexigna@ansible:~$ sudo docker volume create my-vol-001
```

создаём именованный том

```
alexigna@ansible:~$ sudo docker run --rm -it -v my-vol-001:/var/log busybox:latest sh
/ # ping 8.8.8.8 > /var/log/ping.txt
```

Запускаем контейнер с подключенным томом и сохраняем на нём данные

```
alexigna@ansible:~$ sudo docker run --rm -it -v my-vol-001:/var/log:ro busybox:latest sh
/ #
/ # tail -f /var/log/ping.txt
64 bytes from 8.8.8.8: seq=24 ttl=127 time=21.674 ms
64 bytes from 8.8.8.8: seq=25 ttl=127 time=23.715 ms
64 bytes from 8.8.8.8: seq=26 ttl=127 time=21.840 ms
64 bytes from 8.8.8.8: seq=27 ttl=127 time=21.394 ms
64 bytes from 8.8.8.8: seq=28 ttl=127 time=21.798 ms
```

Запускаем второй контейнер с подключенным томом (тем же самым) и читаем данные

Docker Network

Docker использует сетевой стек ядра для предоставления сетевого функционала (namespace net)

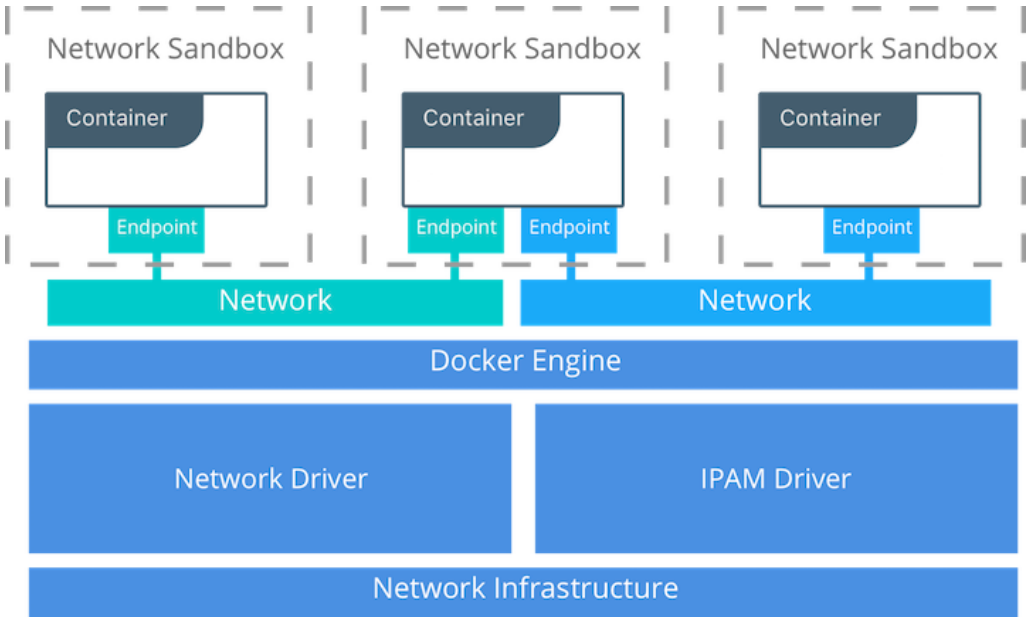
Docker Netrworking = Linux Networking

Составляющие:

- Linux Bridge
  - Layer 2 виртуальная реализация коммутатора внутри ядра Linux
  - Пересылает пакеты на основе MAC адреса
- Network Namespace
  - Изолированный сетевой стек, со своими интерфейсами, таблицей маршрутизации, правилами фильтрации
- veth (virtual eth)
  - Виртуальный интерфейс для обеспечения связности между двумя Network Namespaces
- iptables
  - Правила фильтрации, NAT - управление netfilter.

Docker Network

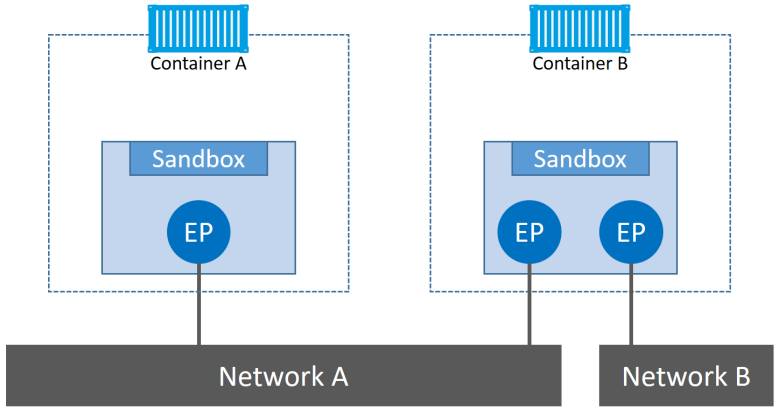
- Container Network Model (CNM)
  - проектная спецификация
  - описывает основные блоки сети Docker
- libnetwork
  - реализация CNM, написана на Go
  - Control + Management Plane
- Различные драйверы
  - реализуют специфичный функционал, напр. VxLAN

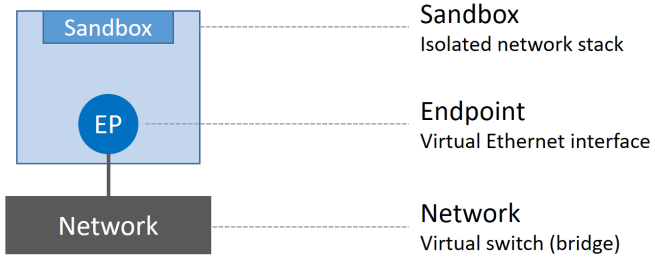


Container Network Model (CNM)

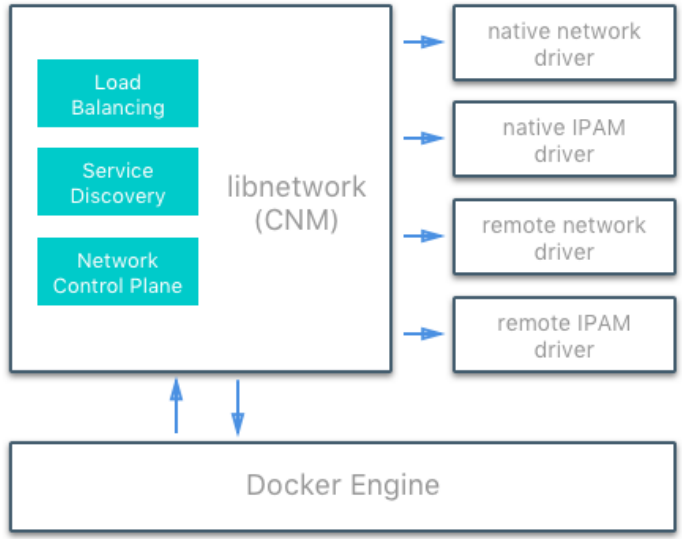
Состоит из трех блоков:

- Sandbox
  - Конфигурация сетевого стека контейнера: настройки интерфейсов, таблица маршрутизации, DNS
  - Может содержать несколько endpoints от нескольких networks
  - Реализация через Linux Network Namespace
- Endpoint
  - Присоединяет Sandbox к Network (логическое представление интерфейса)
  - Принадлежит только одному Sandbox и Network
- Network
  - Набор Endpoint, которые могут взаимодействовать друг с другом
  - Реализация в виде Linux Bridge, VLAN





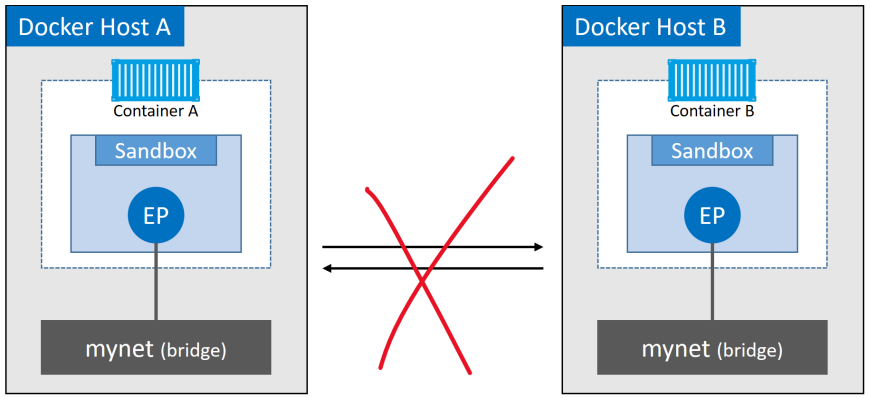
Drivers



- Два типа:
  - Network Drivers
  - IPAM Drivers
- Docker предоставляет встроенные драйвера (network)
  - bridge - драйвер по умолчанию
  - host - удаляет изоляцию между контейнером и Docker host. Контейнер использует Network Namespace хоста
  - overlay - объединяет несколько docker hosts для построения растянутой топологии и используется в Docker Swarm
  - macvlan / ipvlan - позволяет назначит уникальные MAC / IP адреса на интерфейсе контейнера и эти адресе становятся видимыми на сетевом оборудовании. В случае ipvlan используется одинаковый MAC адрес
  - none - полностью изолированный от сети контейнер
- Существует возможность подключения сторонних драйверов

bridge

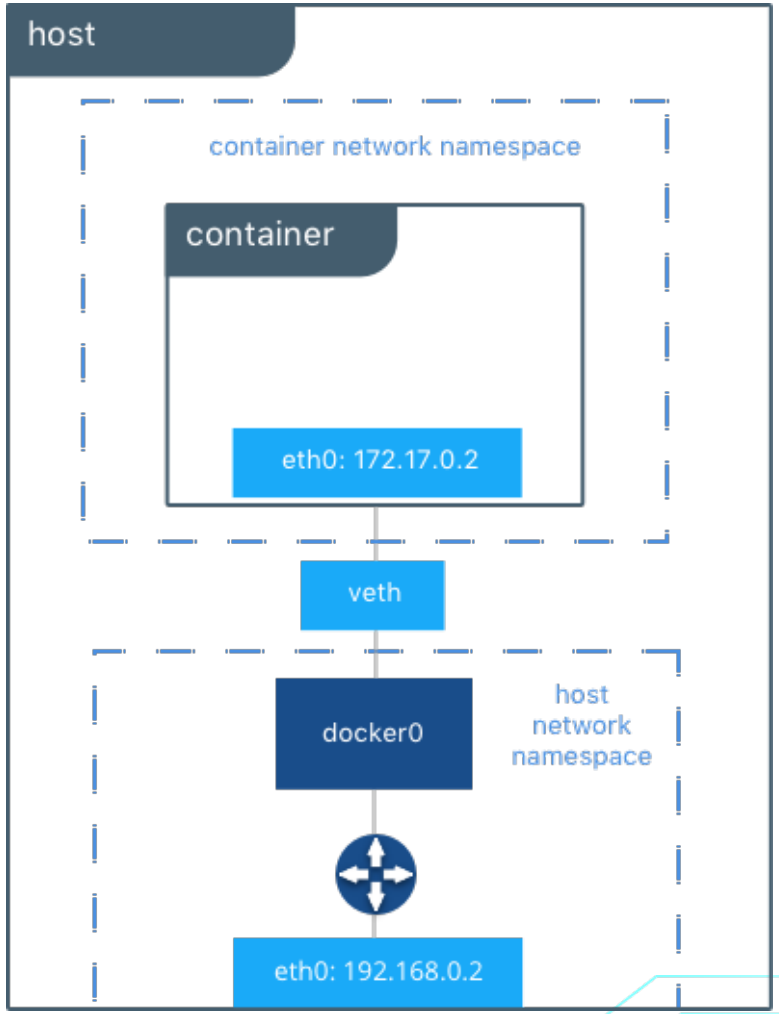
- Local scope: существует на одном docker host и не может соединять контейнеры, запущенные на разных Docker hosts
- реализация стандарта 802.1d (STP)
- Docker на Linux создает сеть с помощью встроенного драйвера bridge



bridge

Два типа:

- default docker0 bridge
- user-defined bridge:
  - Embedded DNS
  - Лучше изоляция (безопасность)
  - Можно подключать/отключать контейнеры во время их работы
  - Отдельный bridge интерфейс, который можно настраивать. Изменения не требуют рестарта Docker



bridge

Тип bridge создает отдельный bridge (br) интерфейс на Docker host, в который помещаются интерфейсы контейнеров.

```
alexigna@runner:~$ sudo docker network create my-bridge-001
77ac285fe573bf693a47e0a2172aff74f6f430ca697b19673b48f615bc89c417

alexigna@runner:~$ sudo docker network ls -f driver=bridge
NETWORK ID     NAME            DRIVER    SCOPE
ec247d631b20   bridge          bridge    local
77ac285fe573   my-bridge-001   bridge    local

alexigna@runner:~$ ip a
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:63:b6:c0:94 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:63ff:feb6:c094/64 scope link
       valid_lft forever preferred_lft forever
20: br-77ac285fe573: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:d8:5f:1e:2f brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 brd 172.18.255.255 scope global br-77ac285fe573
       valid_lft forever preferred_lft forever

alexigna@runner:~$ brctl show
bridge name             bridge id           STP enabled interfaces
br-77ac285fe573         8000.0242d85f1e2f       no          vethcf947ab
                                                            vetheeda70e
docker0                 8000.024263b6c094       no          vethe51573b
```

bridge

Embedded DNS работает только в user-defined сетях

```
alexigna@runner:~$ sudo docker run --rm --name c1 -itd busybox
6c5de9dfee27bfe499788a460815afea15ba9f923dff25215fbf336408a1ef1b

alexigna@runner:~$ sudo docker run --rm --name c2 -itd --network my-bridge-001 busybox
647a39f73e18efe3ff59a9ec8c284d47678dc7a54ec76a1dff4555208ca526ef

alexigna@runner:~$ sudo docker run --rm --name c3 busybox ping -c 2 c1
ping: bad address 'c1'

alexigna@runner:~$ sudo docker run --rm --name c3 busybox ping -c 2 c2
ping: bad address 'c2'

alexigna@runner:~$ sudo docker run --rm --name c3 --network my-bridge-001 busybox ping -c 2 c2
PING c2 (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=9.422 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.140 ms

--- c2 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.140/4.781/9.422 ms

alexigna@runner:~$ sudo docker run --rm --name c3 --network my-bridge-001 busybox ping -c 2 c1
ping: bad address 'c1'
```

bridge

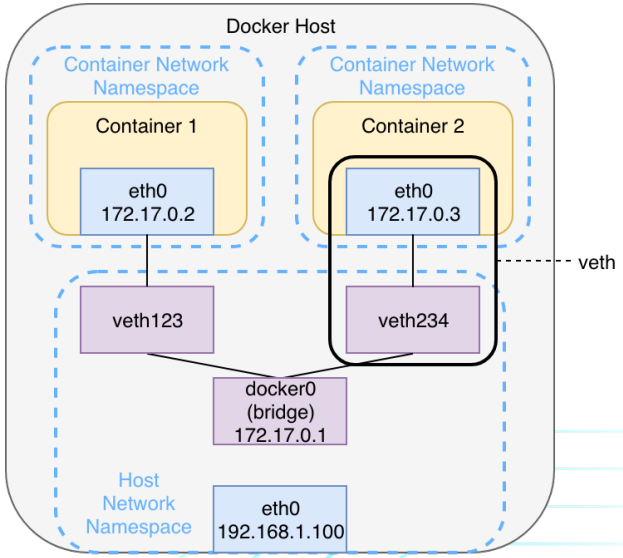
bridge реализован через Linux Network Namespace.

Каждый контейнер видит только свои интерфейсы.

На Docker host создается соответствующая виртуальная копия.

```
/ # ip a
68: eth0@if69: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.2/16 brd 172.18.255.255 scope global eth0
       valid_lft forever preferred_lft forever

/ # ping -c 2 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=60 time=15.615 ms
64 bytes from 8.8.8.8: seq=1 ttl=60 time=15.721 ms

--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
```

```
alexigna@runner:~$ ip a
69: veth5a700b4@if68: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-77ac285fe573
state UP group default
    link/ether 22:9a:7f:93:30:d9 brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet6 fe80::209a:7fff:fe93:30d9/64 scope link
       valid_lft forever preferred_lft forever

alexigna@runner:~$ sudo tcpdump -i veth5a700b4 icmp
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on veth5a700b4, link-type EN10MB (Ethernet), snapshot length 262144 bytes
16:14:20.549261 IP 172.18.0.2 > dns.google: ICMP echo request, id 27, seq 0, length 64
16:14:20.564780 IP dns.google > 172.18.0.2: ICMP echo reply, id 27, seq 0, length 64
16:14:21.550939 IP 172.18.0.2 > dns.google: ICMP echo request, id 27, seq 1, length 64
16:14:21.566568 IP dns.google > 172.18.0.2: ICMP echo reply, id 27, seq 1, length 64
^C
```



bridge

Маршрутизации между сетями нет:

```
alexigna@runner:~$ sudo docker run --rm --name c1 --network my-bridge-001 -itd busybox
c32e4d057b1df2345c17cbb2cf9bbba1fbeb1047fd8a8c628c556e06a0519202

alexigna@runner:~$ sudo docker run --rm --name c2 --network my-bridge-002 busybox ping -c 3 172.18.0.2
PING 172.18.0.2 (172.18.0.2): 56 data bytes

--- 172.18.0.2 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss

<your homework here to get connectivity>

alexigna@runner:~$ sudo docker run --rm --name c2 --network my-bridge-002 busybox ping -c 3 172.18.0.2
PING 172.18.0.2 (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=63 time=0.182 ms
64 bytes from 172.18.0.2: seq=1 ttl=63 time=0.135 ms
64 bytes from 172.18.0.2: seq=2 ttl=63 time=0.147 ms

--- 172.18.0.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.135/0.154/0.182 ms
alexigna@runner:~$
```

bridge

Пример создания bridge с опциями

```
alexigna@runner:~$ sudo docker network create \
--driver bridge \    указание драйвера
--subnet 172.25.0.0/16 \    подсеть на bridge интерфейсе
--ip-range 172.25.100.0/24 \    диапазон, из которого контейнерам будут назначаться ip адреса
--opt com.docker.network.container_iface_prefix=intf \    префикс имени интерфейса в контейнере (83: intf0@if84: …)
--opt com.docker.network.bridge.enable_ip_masquerade=false \    отключаем NAT
--opt com.docker.network.bridge.enable_icc=false \    отключаем взаимодействие между контейнерами в рамках одной сети
my-bridge-003
```

```
При выключенном NAT нет правил в iptables
Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    3   252 MASQUERADE  all  --  any    !br-1ca11e3cc8a3  172.19.0.0/16        anywhere

Chain DOCKER (2 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 RETURN     all  --  br-1ca11e3cc8a3 any     anywhere             anywhere

При выключенном ICC в iptables появляется правило DROP
Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
   0     0 DROP       all  --  br-2065641e206c br-2065641e206c  anywhere             anywhere
```

bridge

При пробросе портов Docker самостоятельно настраивает iptables и начинает слушать порты на Docker host

```
alexigna@runner:~$ sudo docker run -d -p 8080:80/tcp -p 8443:443/tcp --name c1 --rm --network my-bridge-001 nginx
```

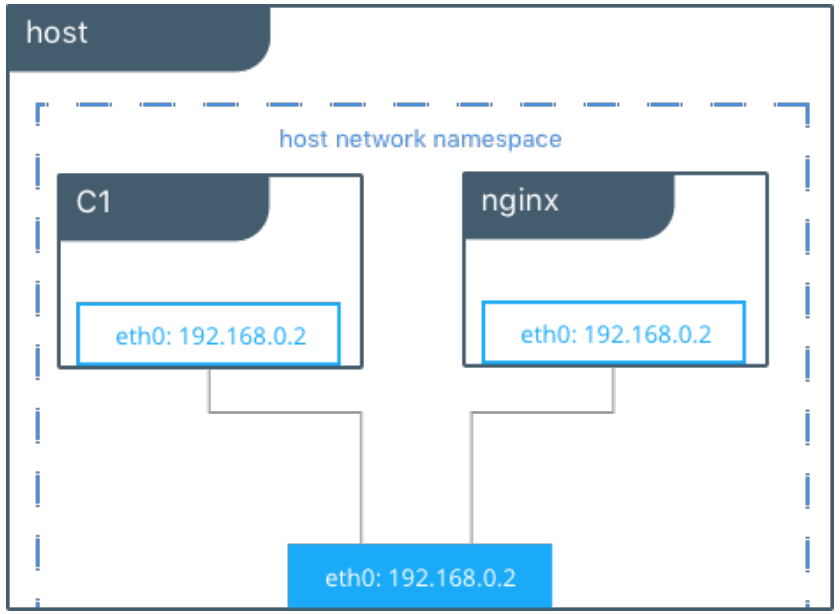
```
alexigna@runner:~$ sudo iptables -t nat -L -v
Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 MASQUERADE  tcp  --  any    any     172.18.0.2           172.18.0.2           tcp dpt:https
    0     0 MASQUERADE  tcp  --  any    any     172.18.0.2           172.18.0.2           tcp dpt:http

Chain DOCKER (2 references)
 pkts bytes target     prot opt in     out     source               destination
    3   128 DNAT       tcp  --  !br-77ac285fe573 any     anywhere             anywhere             tcp dpt:8443 to:172.18.0.2:443
    1    64 DNAT       tcp  --  !br-77ac285fe573 any     anywhere             anywhere             tcp dpt:http-alt to:172.18.0.2:80

alexigna@runner:~$ sudo iptables -L -v
Chain DOCKER (4 references)
 pkts bytes target     prot opt in     out     source               destination
    3   128 ACCEPT     tcp  --  !br-77ac285fe573 br-77ac285fe573  anywhere             172.18.0.2           tcp dpt:https
    1    64 ACCEPT     tcp  --  !br-77ac285fe573 br-77ac285fe573  anywhere             172.18.0.2           tcp dpt:http

alexigna@runner:~$ sudo lsof -i4 -P
COMMAND    PID            USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
docker-pr 9334            root    4u  IPv4  72105      0t0  TCP *:8443 (LISTEN)
docker-pr 9355            root    4u  IPv4  73043      0t0  TCP *:8080 (LISTEN)
```

host

На контейнере используется Network Namespace Docker хоста

Контейнер будет видеть все интерфейсы хоста

Низкая степень безопасности и полное отсутсвие изоляции



macvlan / ipvlan

Способ подключения контейнера в существующую сеть

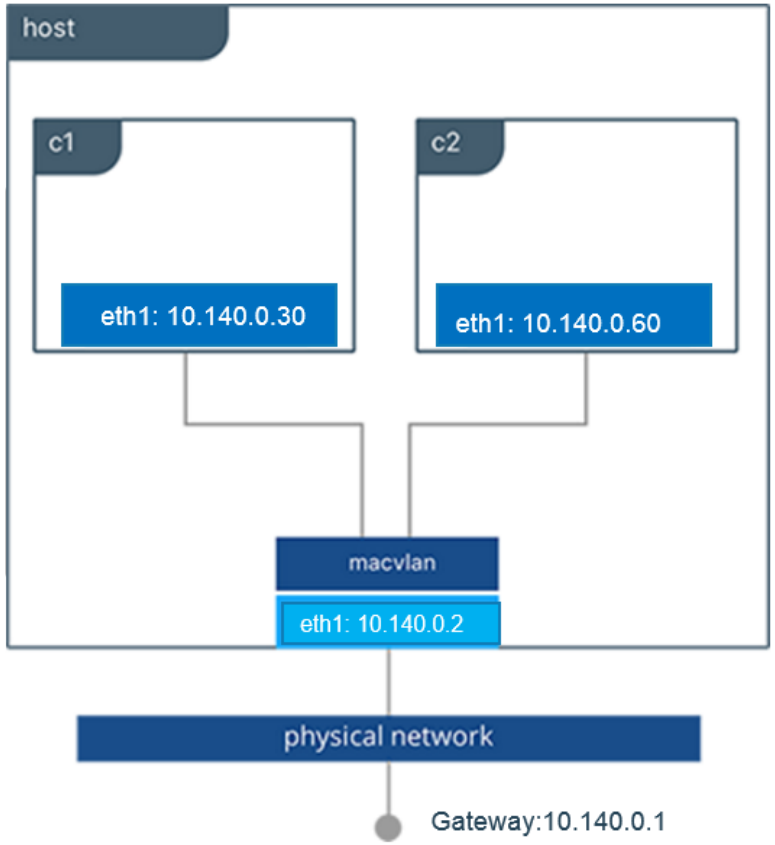
Может использоваться VLAN (802.1q)

Каждый контейнер получает MAC/IP в underlay сети

Каждый контейнер становится доступен в underlay сети без использования пробросов портов

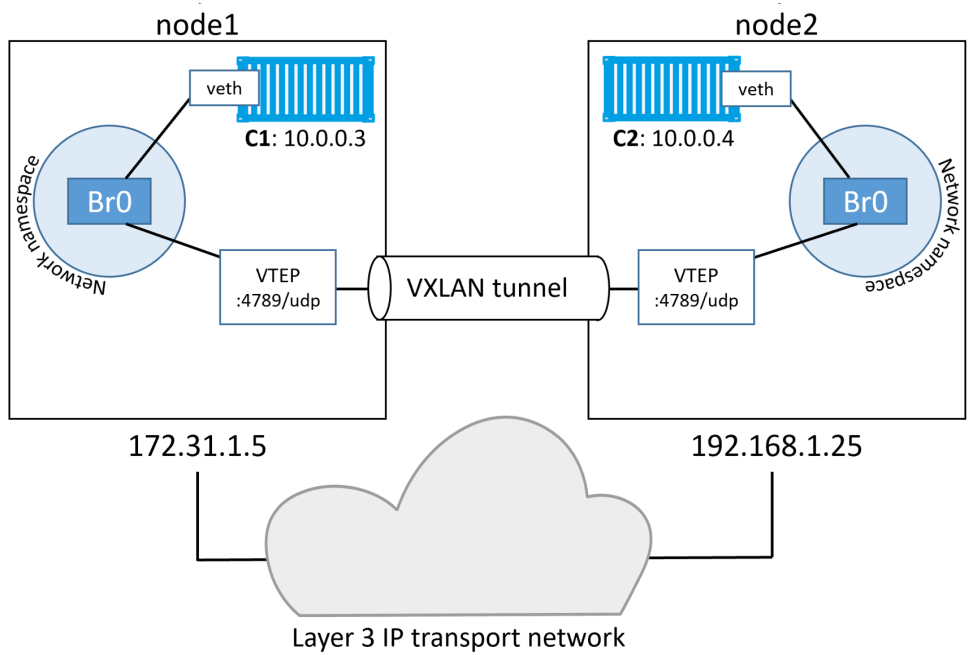
Требуется promiscuous mode на родительском интерфейсе

В случае ipvlan используется один и тот же MAC адрес



overlay

- В качестве транспорта используется VxLAN
- VxLAN = UDP туннель между Virtual Tunnel Endpoint (VTEP)
- В качестве VTEP выступают узлы Docker
- Используется в Docker Swarm
- Трафик может быть зашифрован



namespace NET

```
alexigna@ansible:~$ sudo docker run -d --rm busybox ping 8.8.8.8

alexigna@ansible:~$ sudo docker inspect ef2fefae9058 | grep netns
            "SandboxKey": “/var/run/docker/netns/55bf11bcf27a",

alexigna@ansible ~$ sudo ls -l /var/run/docker/netns/
total 0
-r--r--r-- 1 root root 0 Sep 15 00:02 55bf11bcf27a

alexigna@ansible ~$ sudo ls -l /var/run/netns
total 0

alexigna@ansible ~$ sudo ln -s /var/run/docker/netns/55bf11bcf27a /var/run/netns/55bf11bcf27a

alexigna@ansible ~$ sudo ip netns list
55bf11bcf27a (id: 0)

alexigna@ansible ~$ sudo ip netns exec 55bf11bcf27a ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
9: eth0@if10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
```

NFE Networking For Everyone