NFE Networking For Everyone

Docker

Docker Network

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

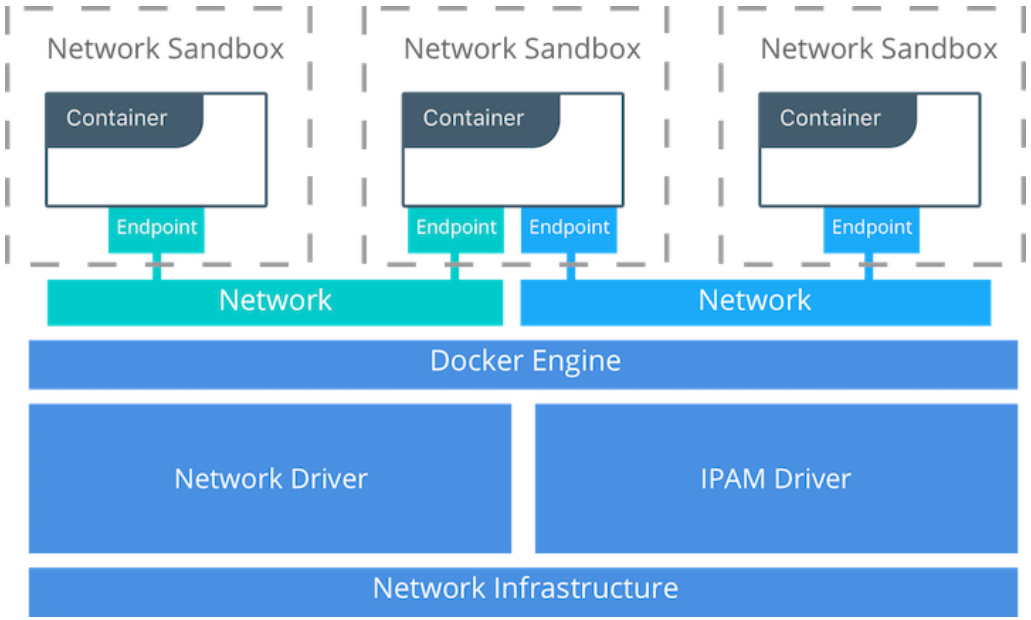
Docker Netrworking = Linux Networking

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

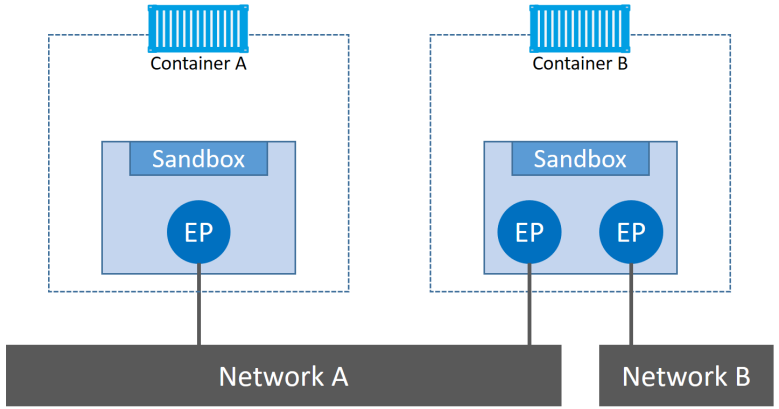
- Container Network Model (CNM)
  - проектная спецификация
  - описывает основные блоки сети Docker
- libnetwork
  - реализация CNM, написана на Go
  - Control + Management Plane
- Различные драйверы
  - реализуют специфичный функционал, напр. VxLAN

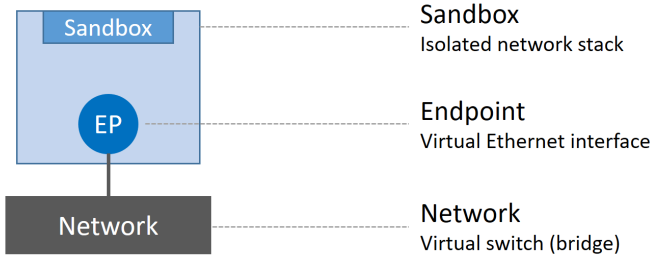


Container Network Model (CNM)

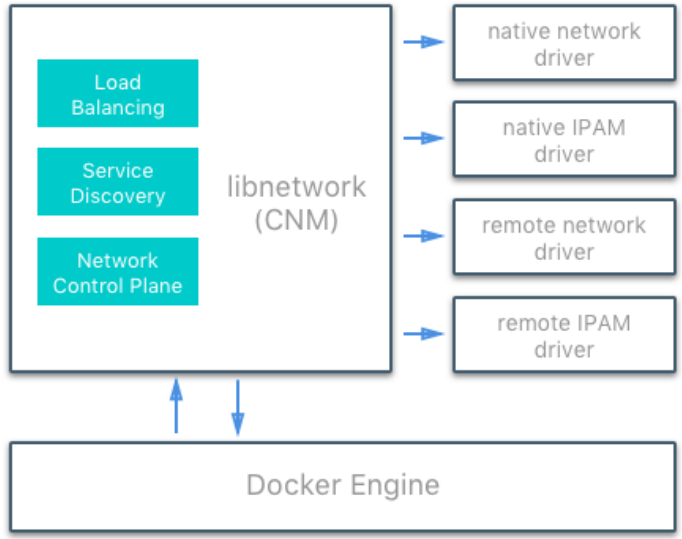
Состоит из трех блоков:

- 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





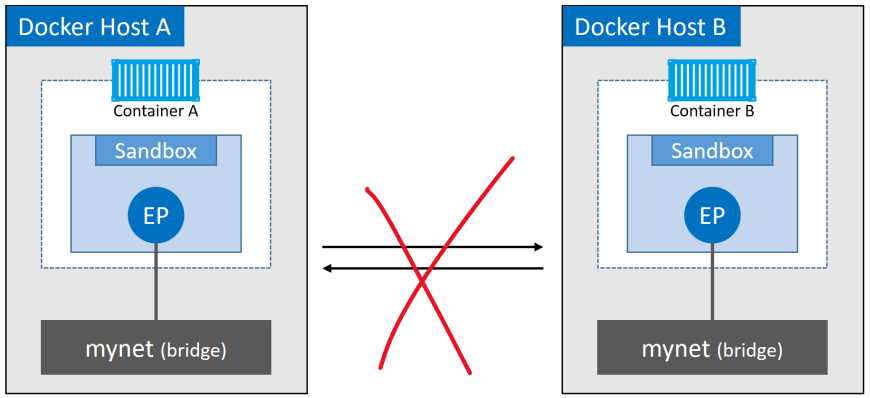
Drivers



- 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

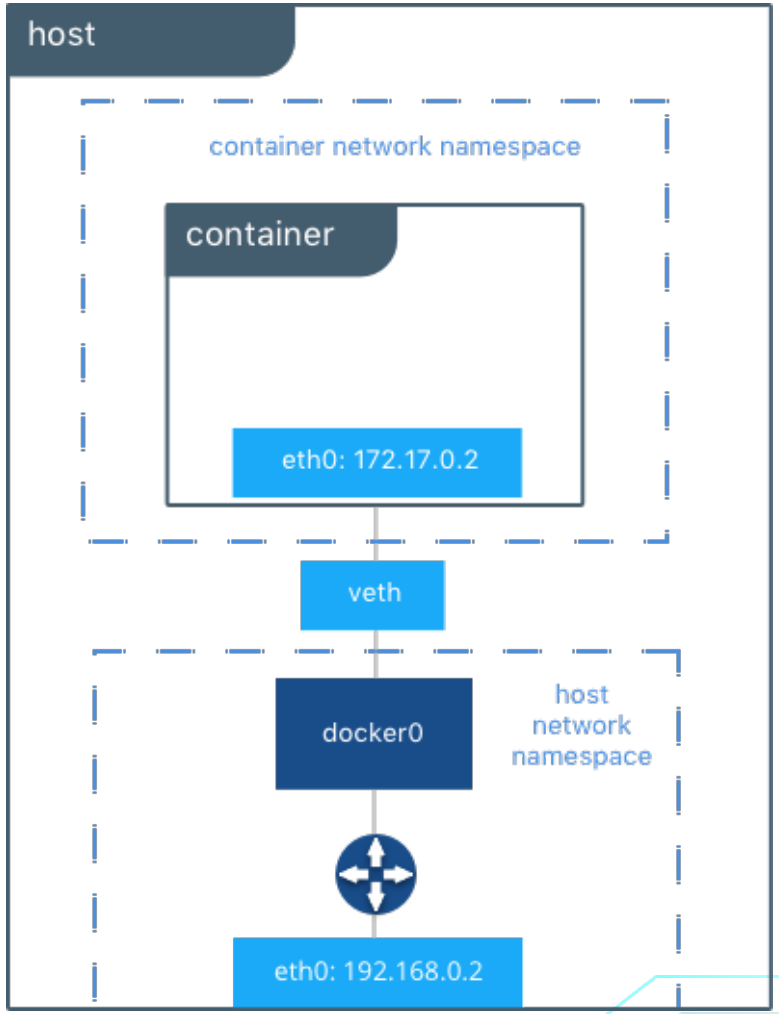
- Local scope: существует на одном docker host и не может соединять контейнеры, запущенные на разных Docker hosts
- реализация стандарта 802.1d (STP)
- Docker на Linux создает сеть с помощью встроенного драйвера bridge



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

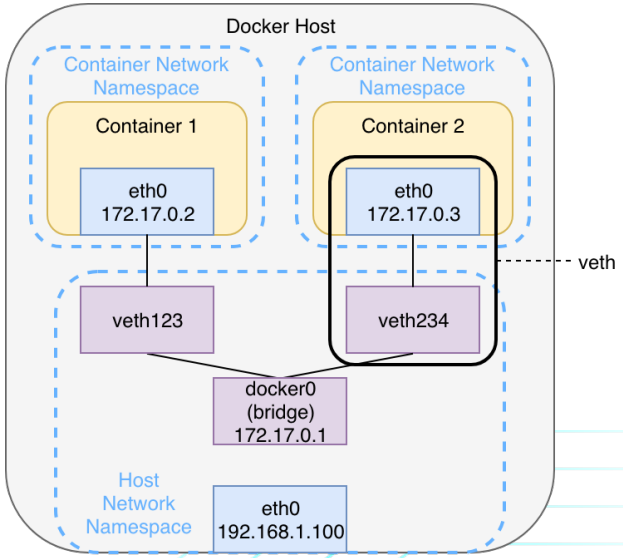
bridge реализован через Linux Network Namespace.

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)
```

namespace NET

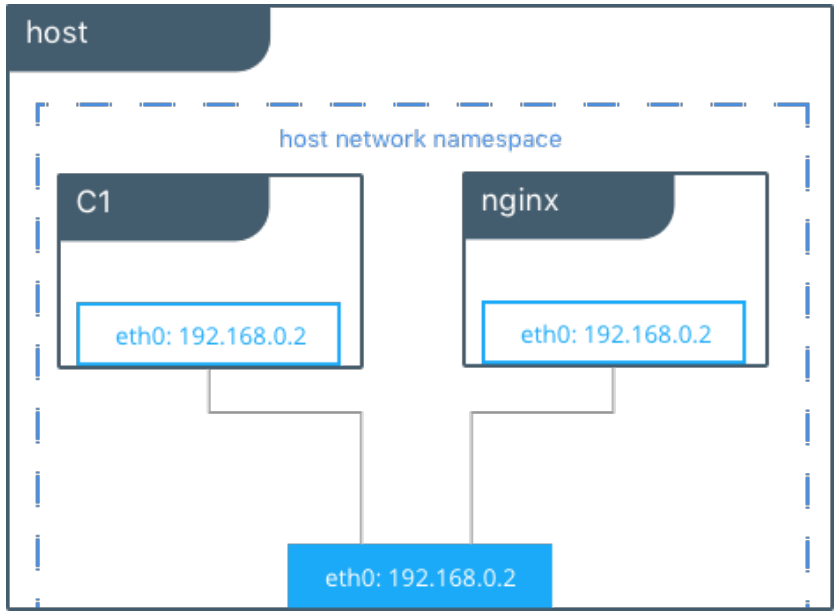
```
alexigna@ansible:~$ sudo docker run -d --rm busybox ping 8.8.8.8

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
```

host

На контейнере используется Network Namespace Docker хоста

Контейнер будет видеть все интерфейсы хоста

Низкая степень безопасности и полное отсутствие изоляции



macvlan / ipvlan

Способ подключения контейнера в существующую сеть

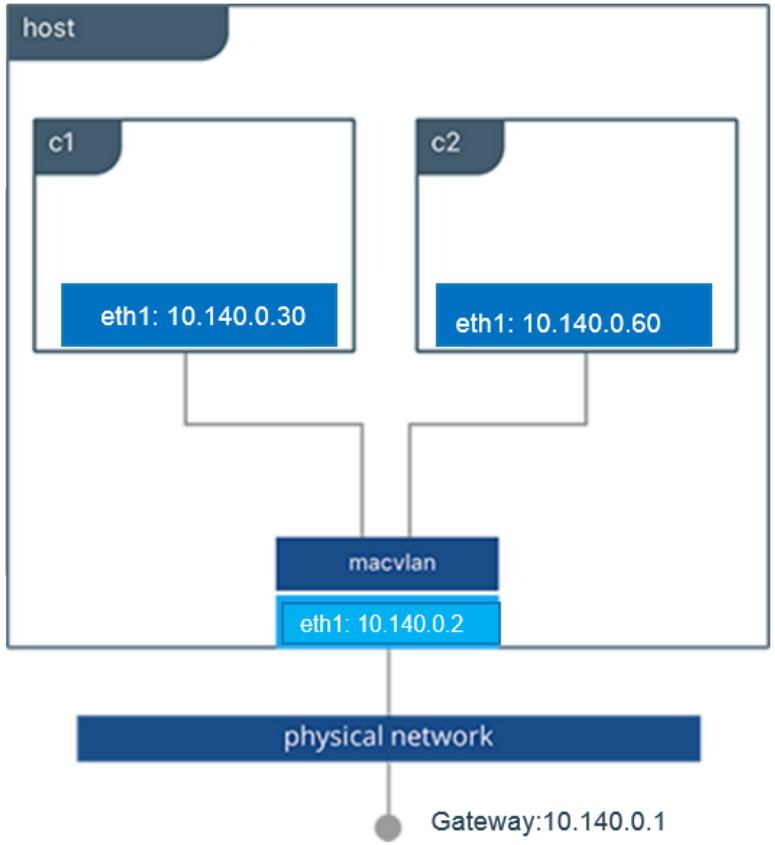
Может использоваться VLAN (802.1q)

Каждый контейнер получает MAC/IP в underlay сети

Каждый контейнер становится доступен в underlay сети без использования пробросов портов

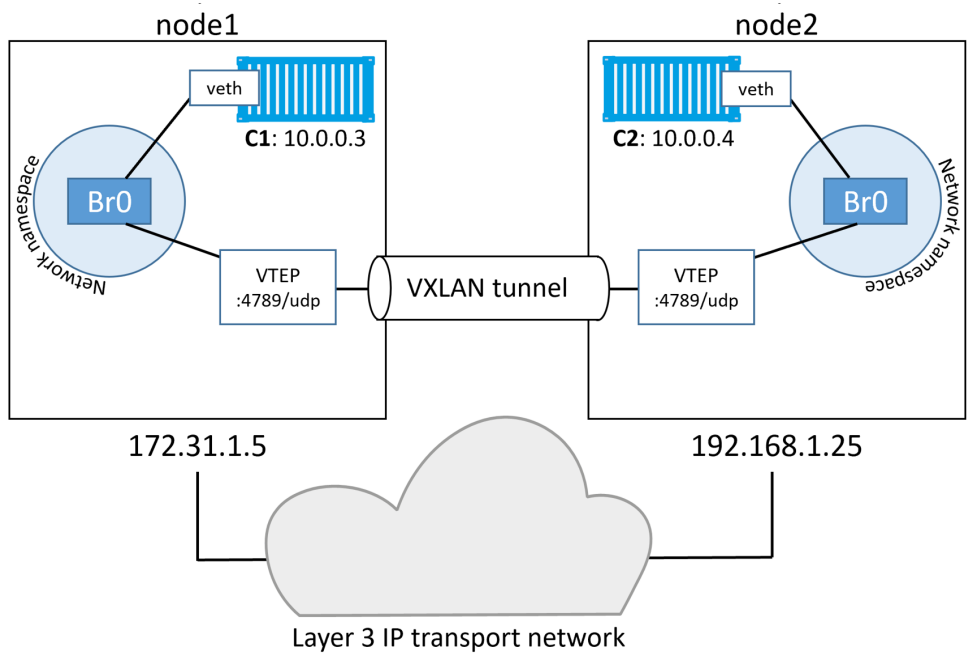
Требуется promiscuous mode на родительском интерфейсе

В случае ipvlan используется один и тот же MAC адрес



overlay

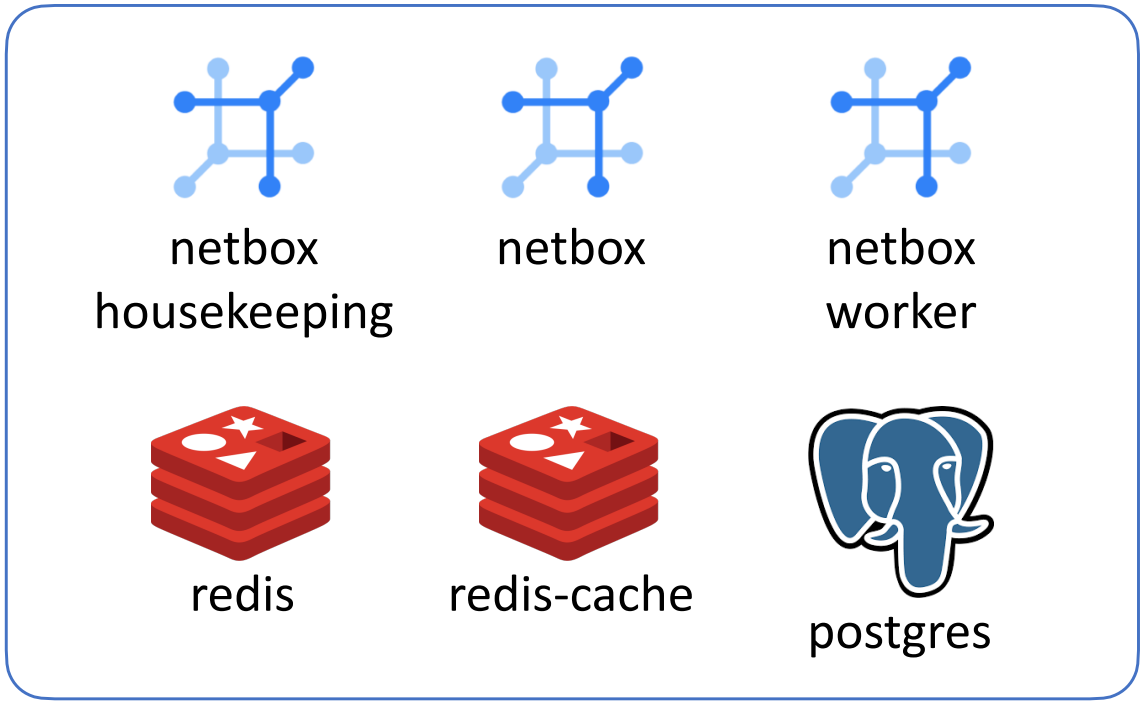
- В качестве транспорта используется VxLAN
- VxLAN = UDP туннель между Virtual Tunnel Endpoint (VTEP)
- В качестве VTEP выступают узлы Docker
- Используется в Docker Swarm
- Трафик может быть зашифрован



Docker compose

Современные приложения состоят из большого количества микросервисов: web, db, logging, etc.

Сложно внедрять и управлять большим количеством таких сервисов.



Императивный подход

Императивный подход: текущее состояние системы не учитывается, каждая команда (инструкция) меняет состояние системы.

```
alexigna@mbp-alexigna ~ % docker run -itd --rm --network my-net-001 --name c1 busybox

alexigna@mbp-alexigna ~ % docker container inspect c1
…
        "NetworkSettings": {
…
            "Networks": {
                "my-net-001": {
…

alexigna@mbp-alexigna ~ % docker network connect my-net-002 c1

alexigna@mbp-alexigna ~ % docker container inspect c1
…
        "NetworkSettings": {
…
            "Networks": {
                "my-net-001": {
…
                "my-net-002": {
…
```

Декларативный подход

Декларативный подход: описывается желаемое состояние системы, и окружение, анализируя текущее состояние системы, самостоятельно определяет что и как нужно поменять

docker-compose.yaml

```yaml
version: "3"

services:
  server:
    build:
      dockerfile: ./server/dockerfile
    image: server:latest
    ports:
      - 8080:80
    networks:
      - client_server_network_002

networks:
  client_server_network_002:
    driver_opts:
      com.docker.network.enable_ipv6: "false"
    ipam:
      config:
        - subnet: 100.64.16.0/24
```

```
alexigna@mbp-alexigna server-client-compose % docker compose up
[+] Running 1/0
 ✔ Container server-client-compose-server-1  Created
0.0s
Attaching to server-client-compose-server-1
server-client-compose-server-1  | INFO:     Started server process [1]
server-client-compose-server-1  | INFO:     Waiting for application startup.
server-client-compose-server-1  | INFO:     Application startup complete.
server-client-compose-server-1  | INFO:     Uvicorn running on http://
0.0.0.0:80 (Press CTRL+C to quit)
^CGracefully stopping... (press Ctrl+C again to force)
Aborting on container exit...
[+] Stopping 1/1
 ✔ Container server-client-compose-server-1  Stopped
0.4s
canceled
```

Docker compose

Текущая версия 3. Минорную можно не указывать, берется актуальная. V1 объявлена как устаревшая.

Именование файла: compose.yaml, compose.yml, docker-compose.yaml и docker-compose.yml (последние два для обратной совместимости).

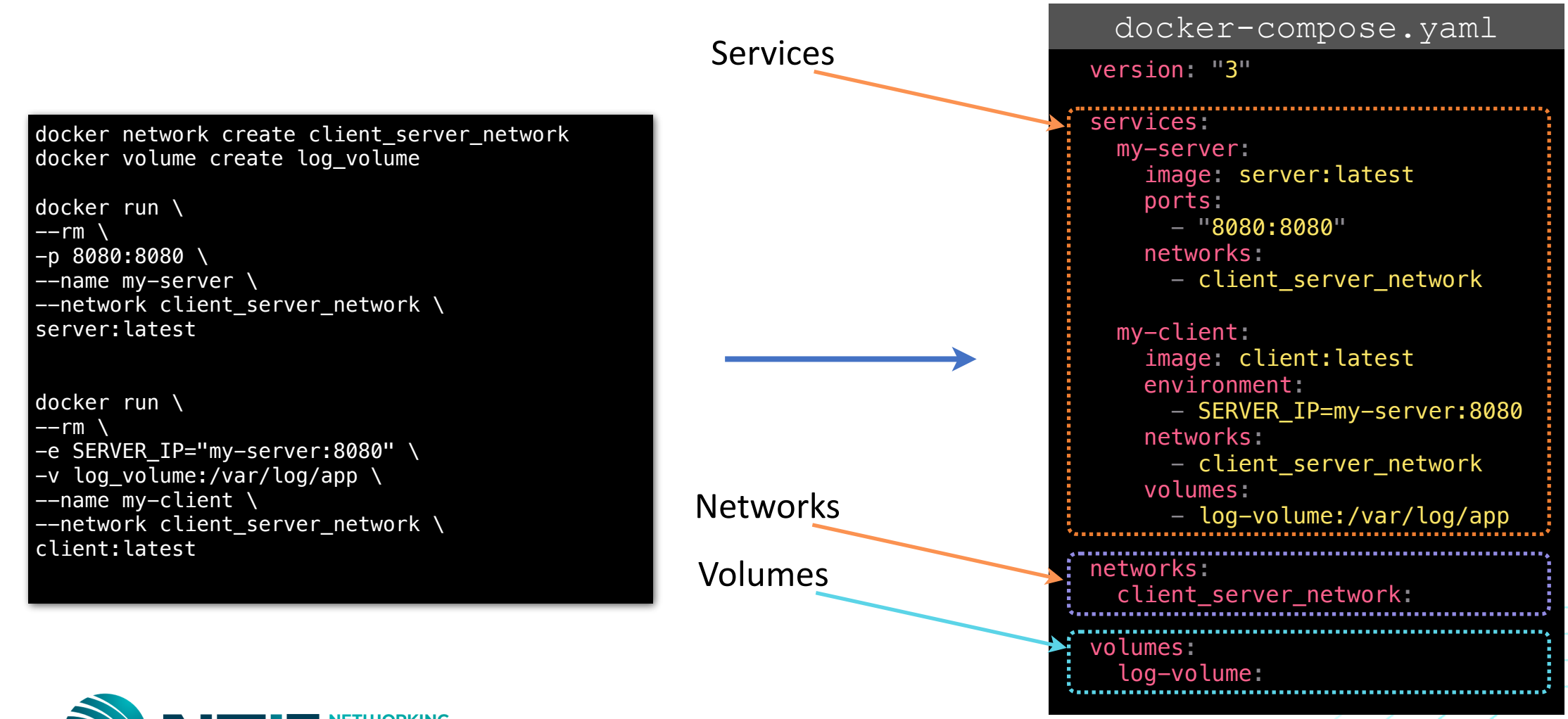
Файлы yaml могут быть объединены (merged): например compose.yml содержит основное описание сервиса, compose.override.yml - какие-то специфичные настройки.

Основные секции:

- service - описывает вычислительные ресурсы и их параметры
- networks - параметры сети и сетевых интерфейсов
- volumes - постоянные (persistent) хранилища

Docker compose

```
docker network create client_server_network
docker volume create log_volume

docker run \
--rm \
-p 8080:8080 \
--name my-server \
--network client_server_network \
server:latest


docker run \
--rm \
-e SERVER_IP="my-server:8080" \
-v log_volume:/var/log/app \
--name my-client \
--network client_server_network \
client:latest
```



Docker compose. build

buil позволяет описать правила сборки образа в docker-compose файле.

docker-compose.yaml

```yaml
version: "3"

services:
  my-server:
    build:
      context: ./server/
      args:
        - FROM=python:3.11-slim


    image: server:latest
```

```
(venv) alexigna@mbp % docker compose build
```

https://docs.docker.com/compose/compose-file/build/

Docker compose. healthcheck

healthcheck - инструкции docker'у, как проверять, что контейнер все еще работает.

В качестве теста может быть любая shell команда, exit code определяет состояние контейнера:

0: контейнер работает и готов к использованию (success)

1: контейнер не работает корректно (unhealthy)

Тестовая команда выполняется внутри контейнера, поэтому соответствующие утилиты должны быть установлены при сборке (например curl).

docker-compose.yaml

```yaml
healthcheck:
  test: "curl --fail http://localhost:8080/status || exit 1"
  interval: 5s
  timeout: 2s
  retries: 3
  start_period: 5s
```

https://docs.docker.com/compose/compose-file/compose-file-v3/#healthcheck
https://docs.docker.com/engine/reference/builder/#healthcheck

```
(venv) alexigna@mbp % docker ps
CONTAINER ID   IMAGE           COMMAND                  CREATED         STATUS                   PORTS     NAMES
ef1342df1881   server:latest   "python -m app.main" 11 seconds ago  Up 10 seconds (healthy) 8080/tcp 03override
```

Homework

Обеспечить связность между двумя контейнерами в двух разных сетях

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

alexigna@ansible:~$ sudo docker inspect c1 | jq '.[].NetworkSettings.Networks."my-bridge-001".IPAddress'
"172.19.0.2"

alexigna@ansible:~$ sudo docker run --rm --name c2 --network my-bridge-002 busybox ping -c 3 172.19.0.2
PING 172.19.0.2 (172.19.0.2): 56 data bytes

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

                              <your homework here to get connectivity>

alexigna@ansible:~$ sudo docker run --rm --name c2 --network my-bridge-002 busybox ping -c 3 172.19.0.2
PING 172.19.0.2 (172.19.0.2): 56 data bytes
64 bytes from 172.19.0.2: seq=0 ttl=63 time=0.182 ms
64 bytes from 172.19.0.2: seq=1 ttl=63 time=0.135 ms
64 bytes from 172.19.0.2: seq=2 ttl=63 time=0.147 ms

--- 172.19.0.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.135/0.154/0.182 ms
```

Homework

NetBox в контейнере netbox-worker запускает один процесс rqworker.

Нужно сделать 5-10 что бы обеспечить возможность одновременную работу нескольких отчетов/скриптов.

https://github.com/netbox-community/netbox-docker/blob/release/docker-compose.yml

```
netbox-worker:
  <<: *netbox
  depends_on:
    netbox:
      condition: service_healthy
  command:
  - /opt/netbox/venv/bin/python
  - /opt/netbox/netbox/manage.py
  - rqworker
  healthcheck:
    start_period: 20s
    timeout: 3s
    interval: 15s
    test: "ps -aux | grep -v grep | grep -q rqworker || exit 1"
```

NFE Networking For Everyone