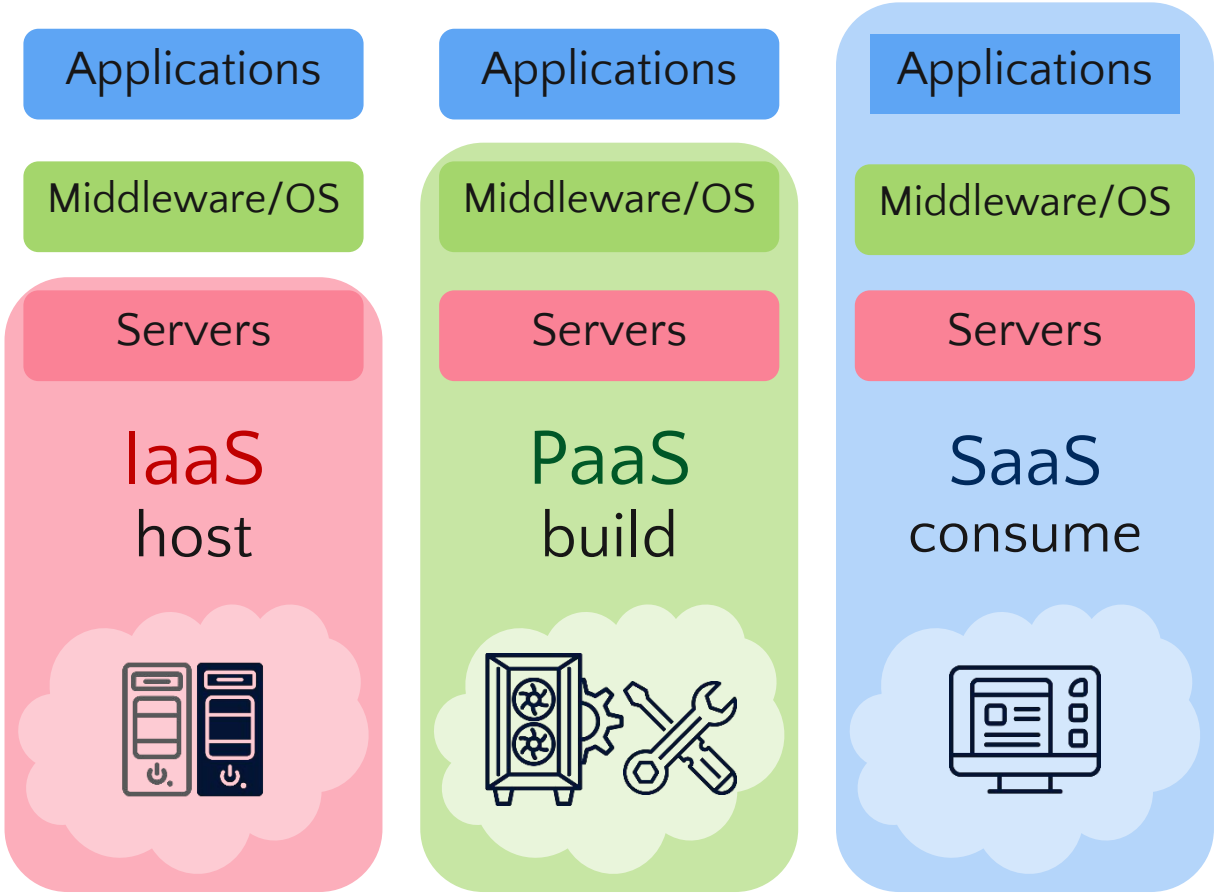
Технология контейнеризации Docker

Ровнягин Михаил Михайлович

15.10.2024

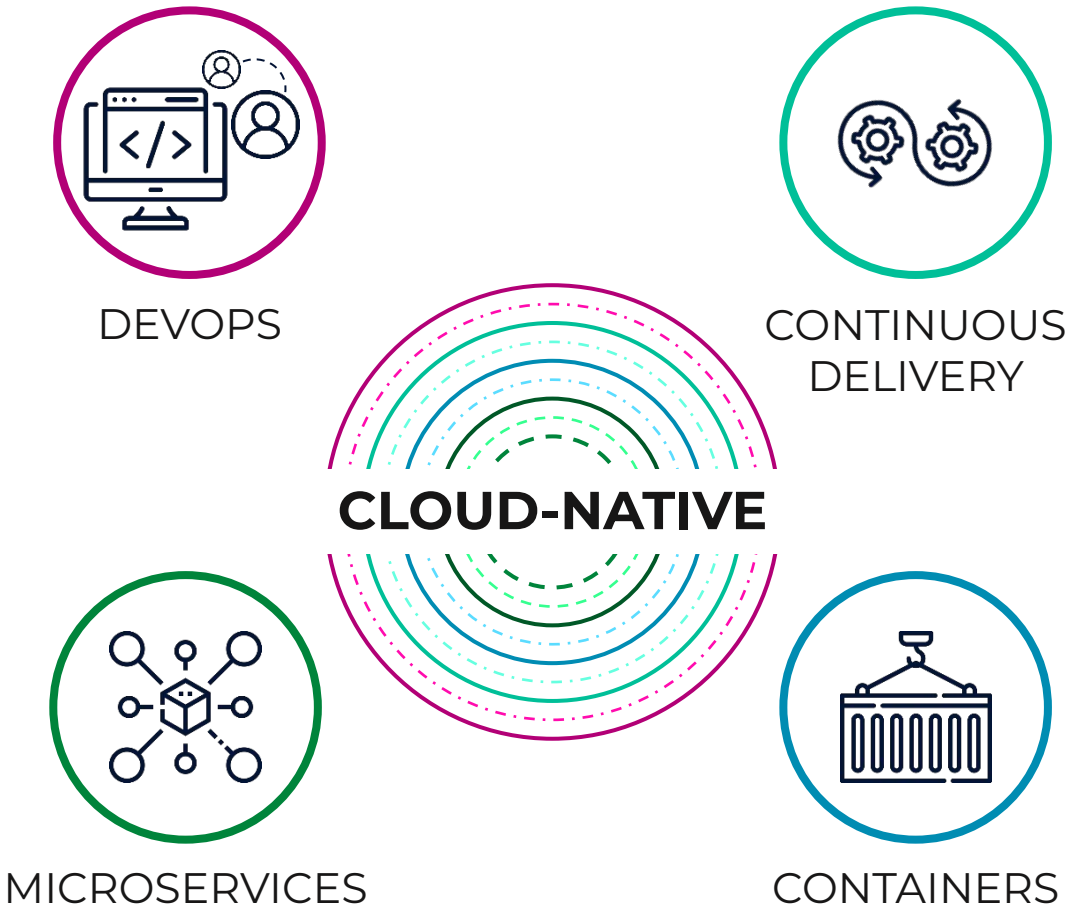
XaaS – концепция облачных вычислений

Все как услуга (XaaS) – это концепция возможности вызова повторно используемых, детализированных программных компонентов по сети



- **IaaS** – клиент получает только инфраструктуру
- **PaaS** – инфраструктуру и подготовленное для разработки приложений ПО
- **SaaS** – готовое работающее в облаке приложение

Cloud Native



Подход, позволяющий задействовать все преимущества облачной модели вычислений.

Свод правил и рекомендаций о том как следует разрабатывать приложения.

Подразумевает:

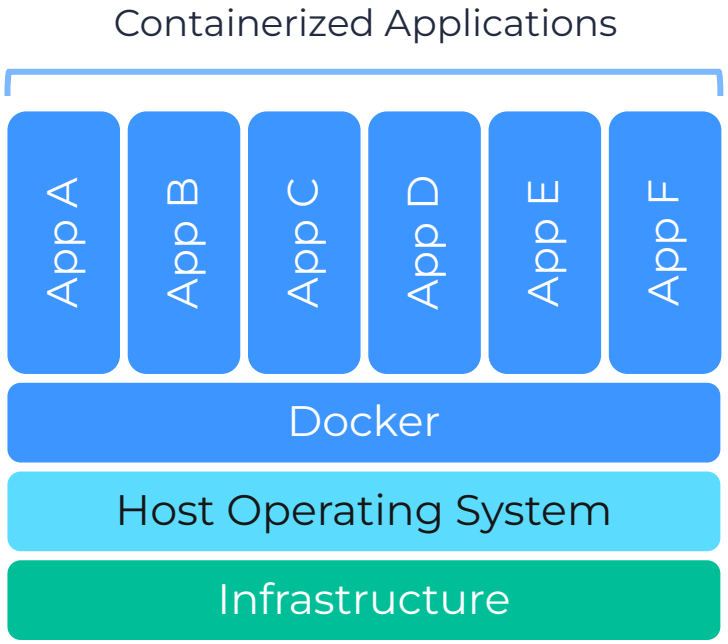
- взаимодействие инженеров разработки и инженеров развертывания
- выстраивание механизма непрерывной доставки обновлений
- проектирование систем на основе микросервисной архитектуры
- контейнеризацию

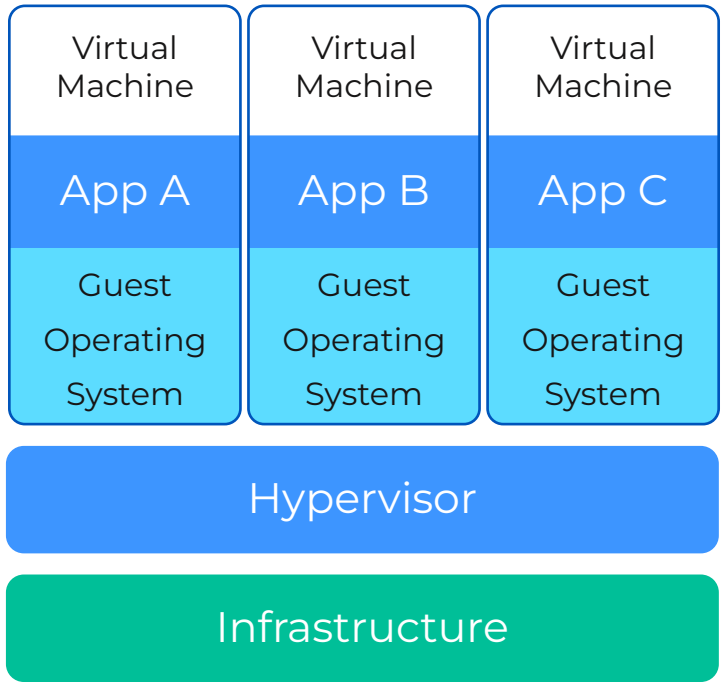
Технология контейнеризации Docker

Docker – это инструмент, предназначенный для упрощения создания, развертывания и запуска приложений.

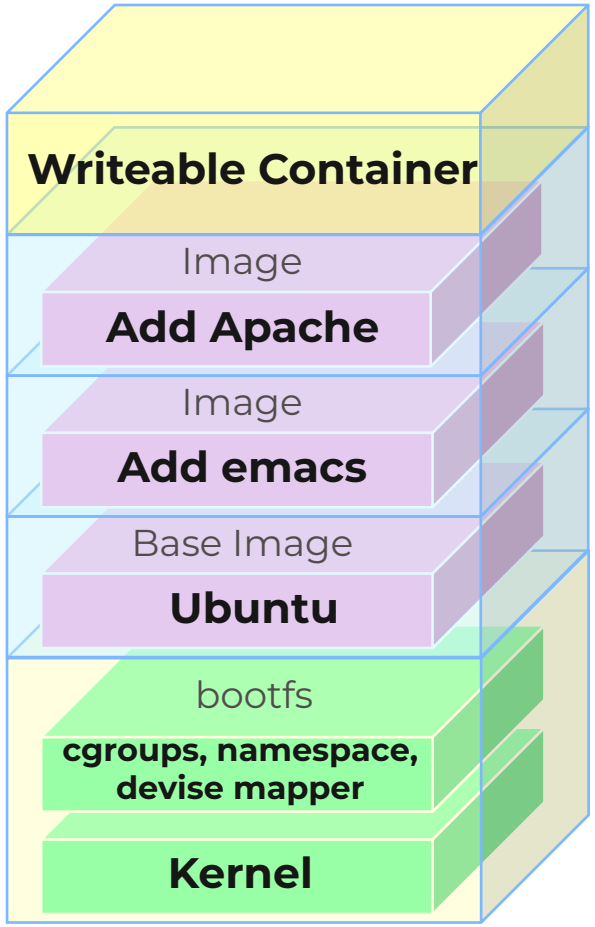
Docker позволяет отделять приложения от остальной инфраструктуры, запаковывая их вместе с зависимостями и запуская их в изолированной среде, называемой контейнером.

Контейнеры легковесны, в отличие от виртуальной машины, потому что они запускаются внутри ядра машины – хоста и не оказывают дополнительной нагрузки на гипервизор.



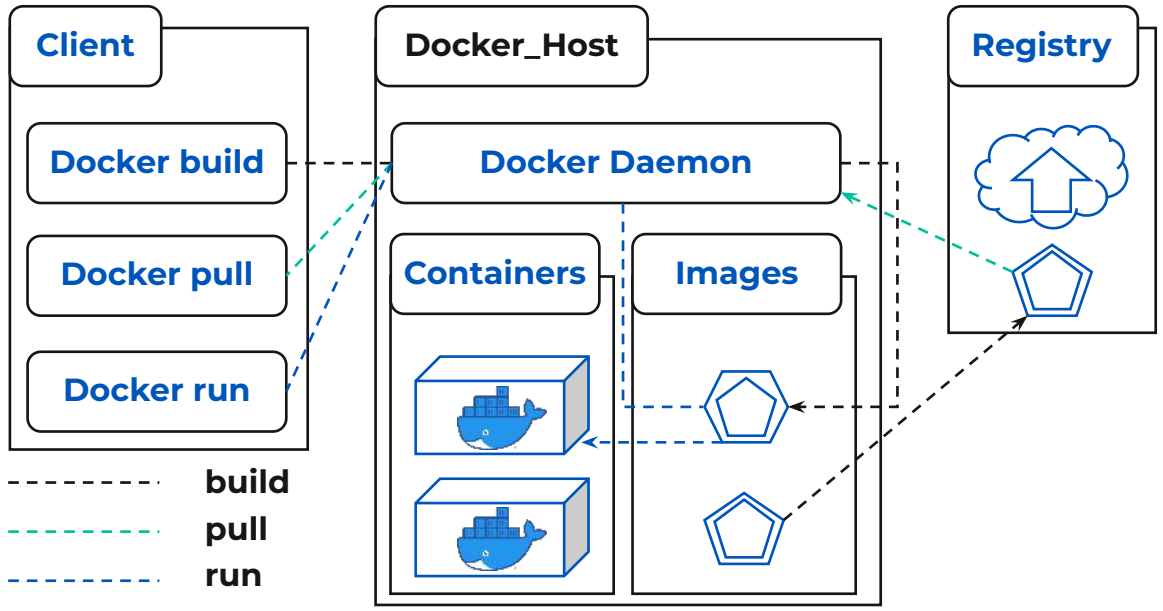


Docker Union File System



- В основе каждого нового образа находится базовый образ. В качестве базового уровня могут использоваться образы Ubuntu, CentOS и т.д.
- Для официальных docker-образов часто используют дистрибутив Alpine – его образ весит меньше 4Мб.
- Шаги описания для создания новых образов называются «инструкциями». Каждая инструкция создает новый образ или уровень. В качестве примера инструкций можно привести:
  - запуск команды (RUN)
  - добавление файла или директории (ADD)
  - создание переменной окружения (ENV)
  - указание входной точки образа (CMD)

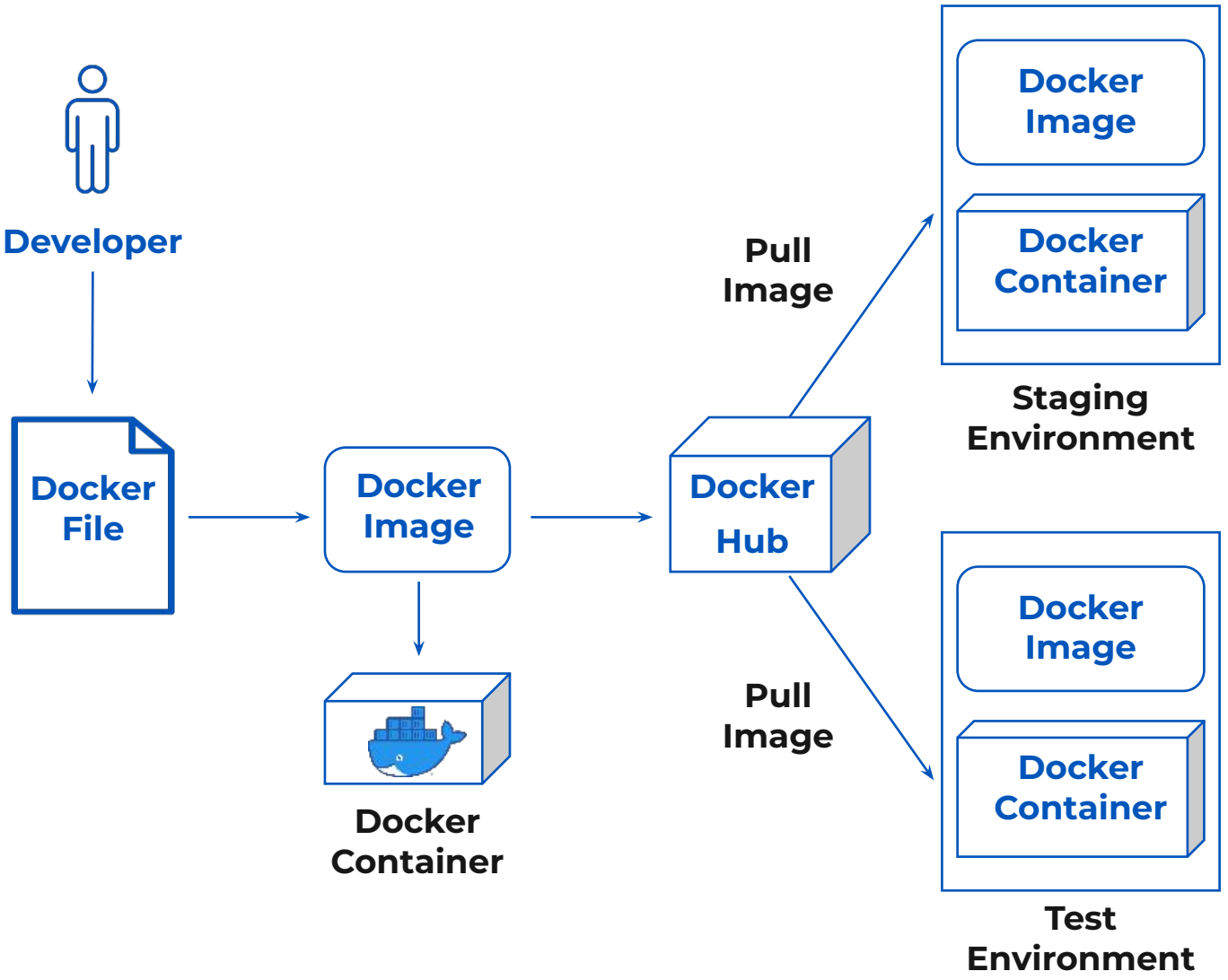
Экосистема Docker



- **Docker-клиент** – программа Docker, получающая команды от пользователя.
  Клиент общается с docker-демоном.
- **Docker-демон** – это фоновый сервис, который отвечает за создание, запуск и уничтожение docker-контейнеров. Клиент и сервер могут работать как на одном узле, так и на разных. Демон и клиент общаются посредством REST API или через Unix-сокет.
- **Docker-образ** – это шаблон «только для чтения». Образ может содержать операционную систему CentOS с установленной JVM и приложением на Java. Образы используются при создании контейнеров.
- **Docker-контейнер** – это исполняемый экземпляр докер образа. Контейнеры содержат весь набор, необходимый для работы приложения, поэтому приложение может быть запущено изолированно.
- **Docker Registry** – специальное хранилище для Docker-образов, позволяющее легко распространять и обновлять образы на серверах в кластере с сохранением их предыдущих версий.

Пример Dockerfile

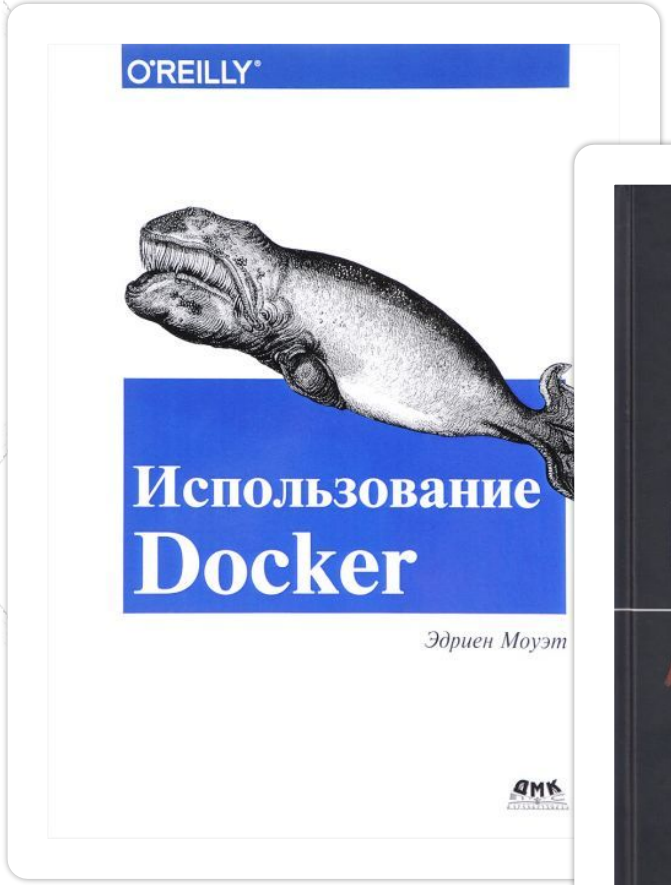
```
# Сборка
FROM eclipse-temurin:17-jdk-alpine as builder
WORKDIR /opt/app
COPY .mvn/ .mvn
COPY mvnw pom.xml ./
RUN ./mvnw dependency:go-offline
COPY ./src ./src
RUN ./mvnw clean install
# Запуск
FROM eclipse-temurin:17-jre-alpine
WORKDIR /opt/app
COPY --from=builder /opt/app/target/*.jar /opt/app/*.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "/opt/app/*.jar"]
```

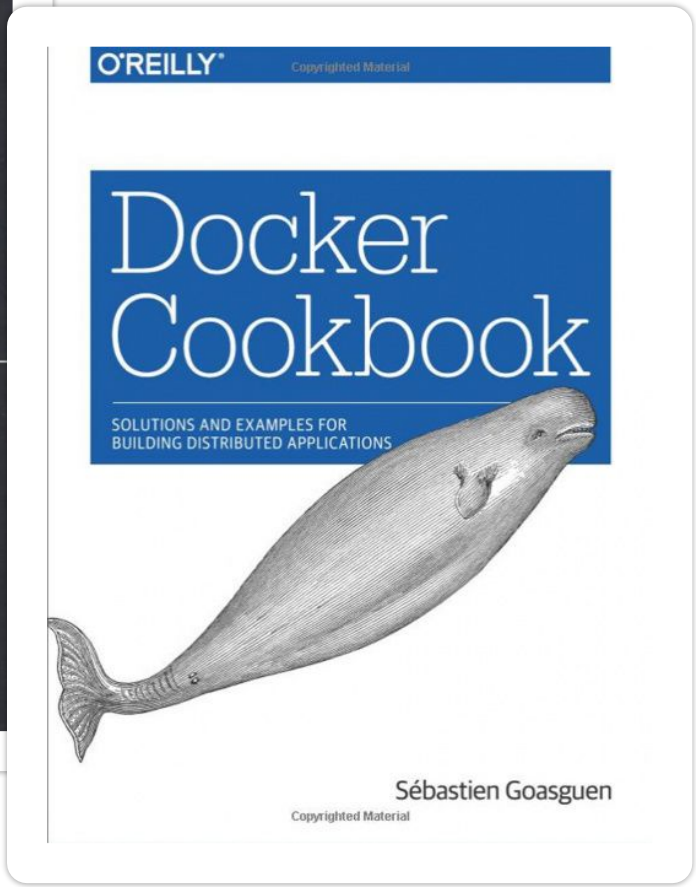


Полезные ссылки

Cgroups

Namespaces





Основные команды Docker

- `docker run` – запускает команду в новом контейнере
- `docker start` – запускает один или несколько остановленных контейнеров
- `docker stop` – останавливает один или несколько работающих контейнеров
- `docker build` – создает образ из файла Docker
- `docker pull` – извлекает образ из реестра
- `docker push` – выталкивает изображение в реестр
- `docker export` – экспортирует файловую систему контейнера как архив tar
- `docker exec` – запускает команду во время выполнения контейнера
- `docker search` – поиск образов в Docker Hub
- `docker commit` – создает новый образ из изменений контейнера

Пример Dockerfile


```
FROM ubuntu:22.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

- `FROM` создает слой из образа ubuntu:22.04
- `COPY` добавляет файлы из текущей директории в директорию `/app` контейнера
- `RUN` запускает команду `make` внутри контейнера
- `CMD` определяет команду, которая будет запущена внутри контейнера

Docker build

- `docker build [OPTIONS] PATH | URL | -`

Команда сборки docker создает образы Docker из Dockerfile и «контекста».

- Контекст сборки – это набор файлов, расположенных в указанном PATH или URL. Процесс сборки может ссылаться на любой из файлов в контексте. Например, ваша сборка может использовать инструкцию COPY для ссылки на файл в контексте.

- Параметр URL может относиться к трем видам ресурсов:
  - репозитории Git,
  - предварительно упакованные тарбол-контексты,
  - простые текстовые файлы.

Пример Docker build

```
$ docker build .
Uploading context 10240 bytes
Step 1/3 : FROM busybox
Pulling repository busybox
 ---> e9aa60c60128MB/2.284 MB (100%) endpoint:
https://cdn-registry-1.docker.io/v1/
Step 2/3 : RUN ls -lh /
 ---> Running in 9c9e81692ae9
total 24
drwxr-xr-x    2 root     root        4.0K Mar 12  2013 bin
drwxr-xr-x    5 root     root        4.0K Oct 19 00:19 dev
drwxr-xr-x    2 root     root        4.0K Oct 19 00:19 etc
drwxr-xr-x    2 root     root        4.0K Nov 15 23:34 lib
lrwxrwxrwx    1 root     root           3 Mar 12  2013
lib64 -> lib
dr-xr-xr-x  116 root     root           0 Nov 15 23:34
proc
lrwxrwxrwx    1 root     root           3 Mar 12  2013
sbin -> bin
dr-xr-xr-x   13 root     root           0 Nov 15 23:34 sys
drwxr-xr-x    2 root     root        4.0K Mar 12  2013 tmp
drwxr-xr-x    2 root     root        4.0K Nov 15 23:34 usr
 ---> b35f4035db3f
Step 3/3 : CMD echo Hello world
 ---> Running in 02071fceb21b
 ---> f52f38b7823e
Successfully built f52f38b7823e
Removing intermediate container 9c9e81692ae9
Removing intermediate container 02071fceb21b
```

Docker run

- `docker run [OPTIONS] IMAGE [COMMAND] [ARG... ]`

- Команда `docker run` сначала создает доступный для записи слой-контейнер над указанным образом, а затем запускает его, используя указанную команду.
  То есть, запуск Docker эквивалентен API /containers/create then /containers/(id)/start.
  Остановленный контейнер может быть перезапущен с сохранением всех предыдущих изменений с помощью запуска Docker. Используйте `docker ps -a` для просмотра списка всех контейнеров.

- Команда `docker run` может использоваться в сочетании с `docker commit` для изменения команды, выполняемой контейнером. В справочнике по запуску Docker содержится дополнительная подробная информация о запуске Docker.

Пример Docker run


```
$ docker run --name test -it debian
root@d6c0fe130dba:/# exit 13
$ echo $?
13
$ docker ps -a | grep test
d6c0fe130dba debian:7 "/bin/bash" 26 seconds ago Exited (13) 17
seconds ago test
```

Установка Docker

```
[root@localhost hpc]# yum install -y yum-utils
[root@localhost hpc] # yum-config-manager --add-repo
https://download.docker.com/linux/rhel/docker-ce.repo
[root@localhost hpc]# yum install sudo yum install docker-ce
docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
-y
[root@localhost hpc]# systemctl start docker
[root@localhost hpc]# systemctl enable docker
[root@localhost hpc]# reboot
```

Пример docker-compose.yml

```
services:
  elasticsearch:
    image: elasticsearch:7.16.1
    container_name: es
    environment:
      discovery.type: single-node
      ES_JAVA_OPTS: "-Xms512m -Xmx512m"
    ports:
      - "9200:9200"
      - "9300:9300"
    healthcheck:
      test: ["CMD-SHELL", "curl --silent --fail localhost:9200/_cluster/health || exit 1"]
      interval: 10s
      timeout: 10s
      retries: 3
    networks:
      - elastic
kibana:
    image: kibana:7.16.1
    container_name: kib
    ports:
      - "5601:5601"
    depends_on:
      - elasticsearch
    networks:
      - elastic
networks:
  elastic:
    driver: bridge
```

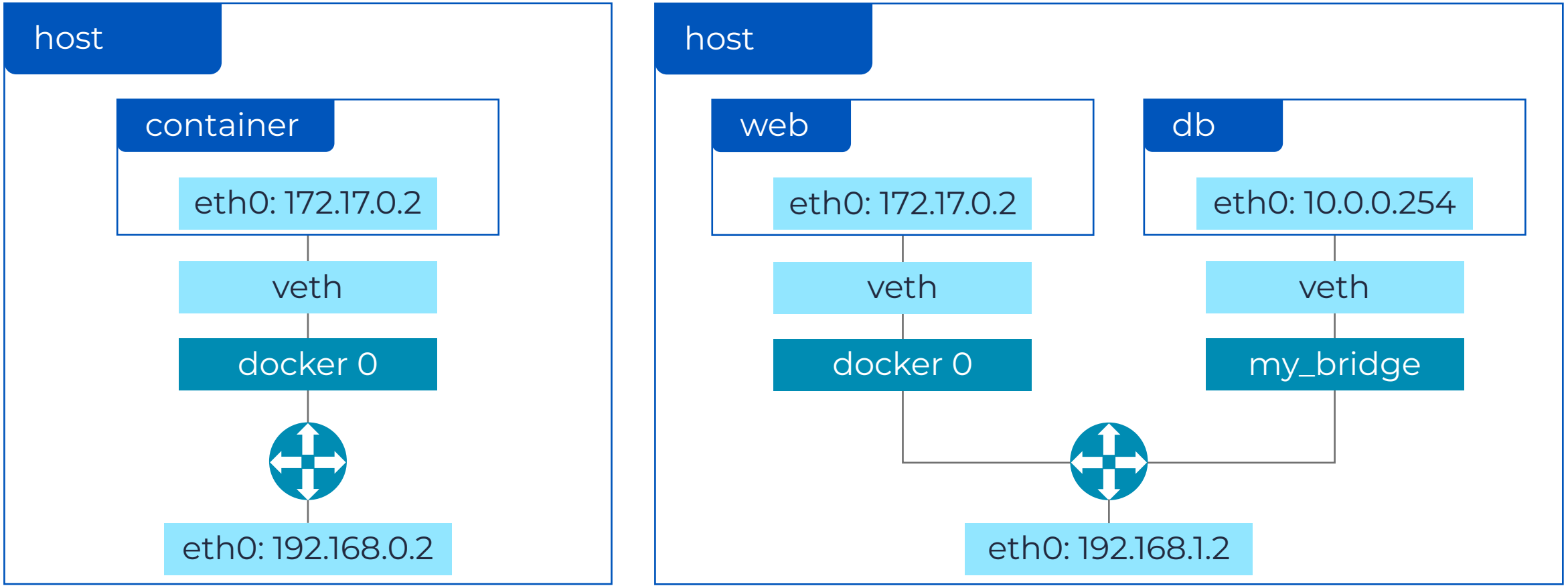
Копирование файлов в контейнер

```
[hpc@localhost ~]$ docker ps

CONTAINER ID   IMAGE                          COMMAND              CREATED                 STATUS     PORTS     NAMES

2521bba9c548   carlochess/openmpi:latest "/usr/sbin/sshd -D" 21 minutes ago  Up 21 minutes   22/tcp
test_mpi_node.1.xtbm3cx96l10ic7dzk6ouqi7p

e6a0ba9d78d7   carlochess/openmpi:latest "/usr/sbin/sshd -D" 21 minutes ago  Up 20 minutes   22/tcp
test_mpi_head.1.uh3nw3epr0vcvzqothtu8gabt

[hpc@localhost ~]$ docker cp MPI_Hello.c e6a0ba9d78d7:/root/MPI_Hello.c

[hpc@localhost ~]$ docker exec -it e6a0ba9d78d7 /bin/bash

root@e6a0ba9d78d7:~# ls

MPI_Hello.c
```

Сеть Docker

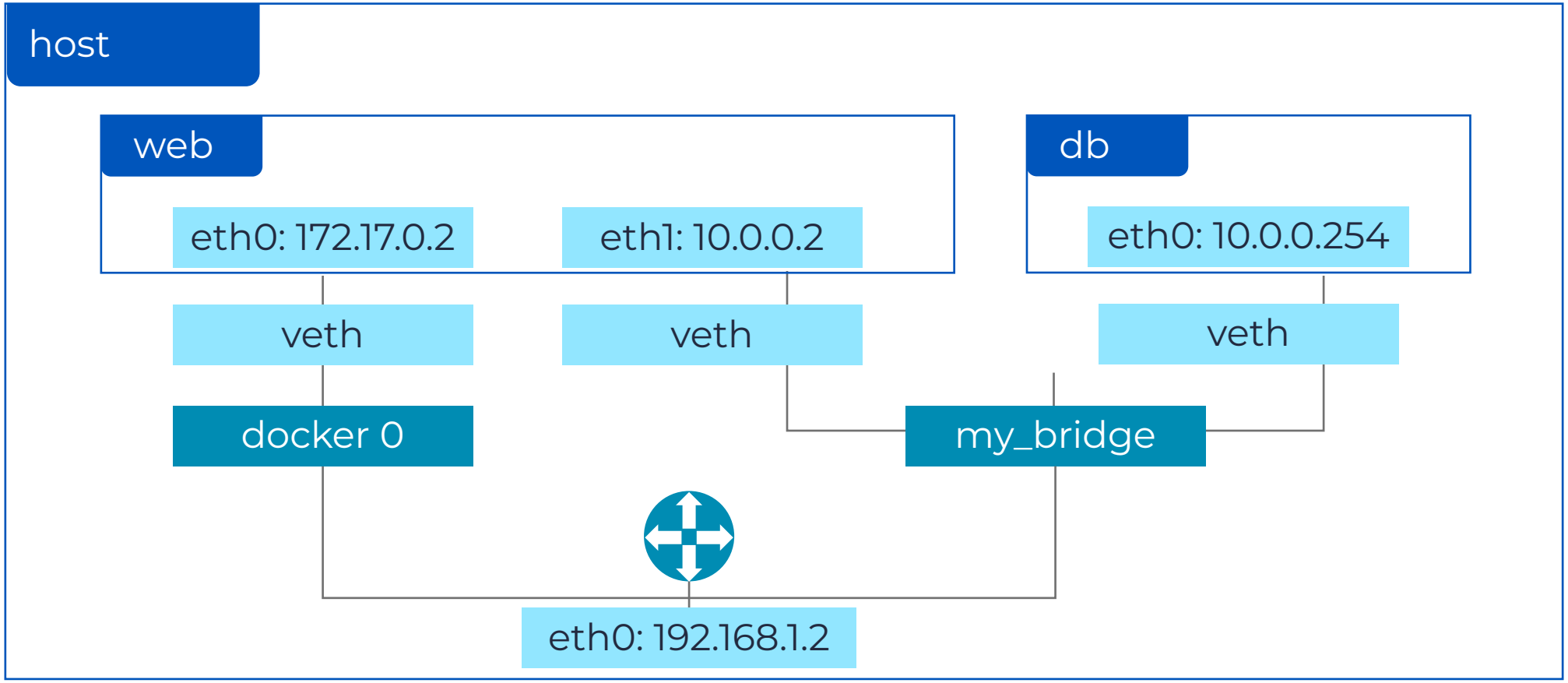
- **bridge:** сетевой драйвер по умолчанию. Если вы не указываете драйвер, создается сеть этого типа. Мостовые сети обычно используются, когда ваши приложения работают в автономных контейнерах, которые должны взаимодействовать.
- **host:** удаляет сетевую изоляцию между контейнером и Docker-хостом, используется непосредственно сеть хоста. Режим host доступен только с Docker 17.06 и выше.
- **overlay:** оверлейные сети соединяют несколько демонов Docker вместе и позволяют сервисам Swarm связываться друг с другом. Вы также можете использовать оверлейные сети для облегчения связи между сервисом Swarm и автономным контейнером или между двумя автономными контейнерами на разных демонах Docker. Эта стратегия устраняет необходимость выполнять маршрутизацию на уровне ОС между этими контейнерами.
- **macvlan:** сети Macvlan позволяют назначать MAC-адрес контейнеру, что делает его физическим устройством в вашей сети. Демон Docker направляет трафик в контейнеры по их MAC-адресам. Использование драйвера macvlan иногда является лучшим выбором при работе с устаревшими приложениями, которые ожидают прямого подключения к физической сети, а не маршрутизации через сетевой стек хоста Docker.
- **none:** для контейнера отключаются все сети. Обычно используется вместе с пользовательским сетевым драйвером.

Bridge docker network



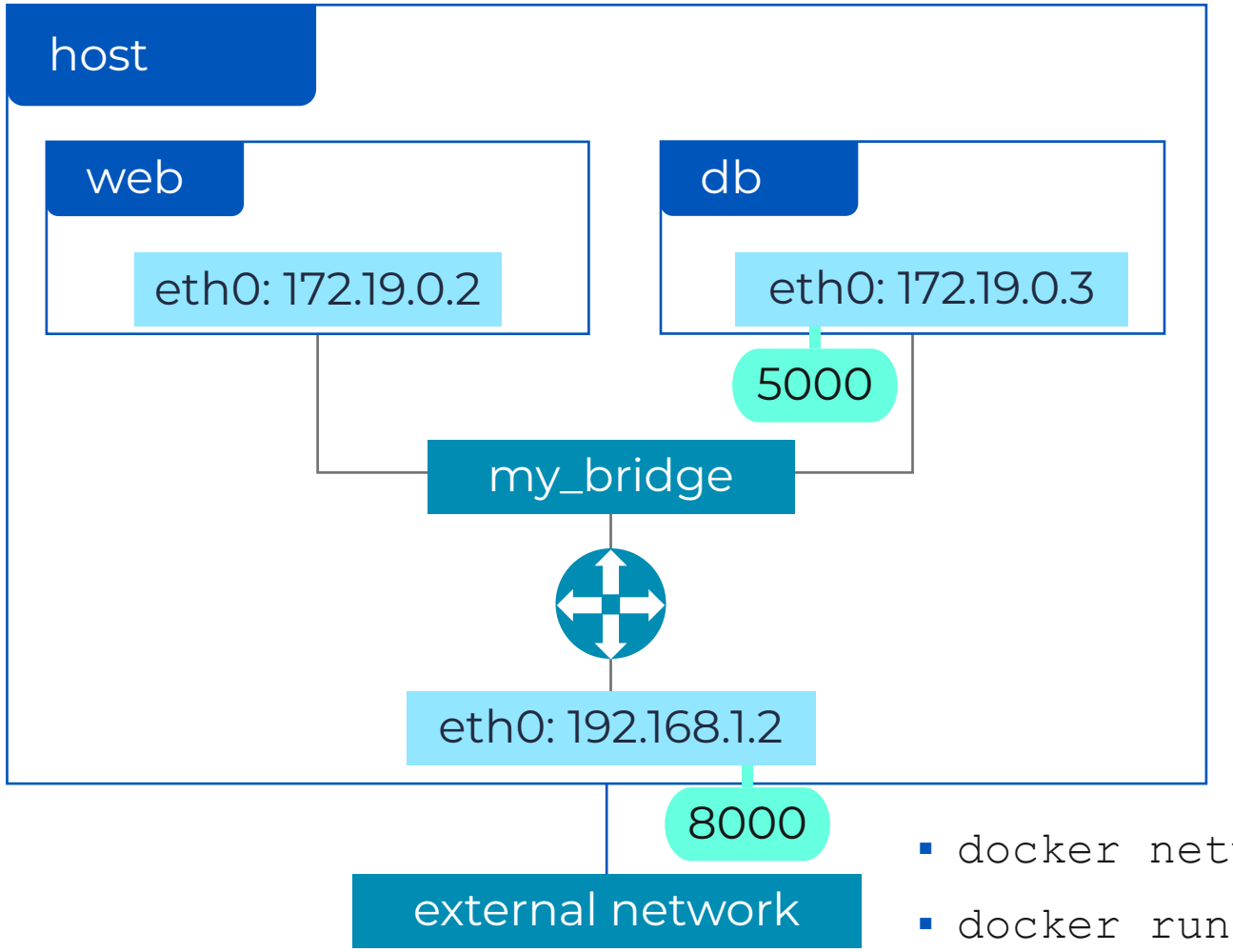
- `docker run -d --name=networktest ubuntu`
- `docker network create -d bridge my_bridge`
- `docker run -d --net=my_bridge --name db training/postgres`

Bridge подключение



- Сеть Docker позволяет подключить контейнер к любому количеству сетей. Вы также можете прикрепить уже работающий контейнер.
- docker network connect my_bridge web

Отображение портов

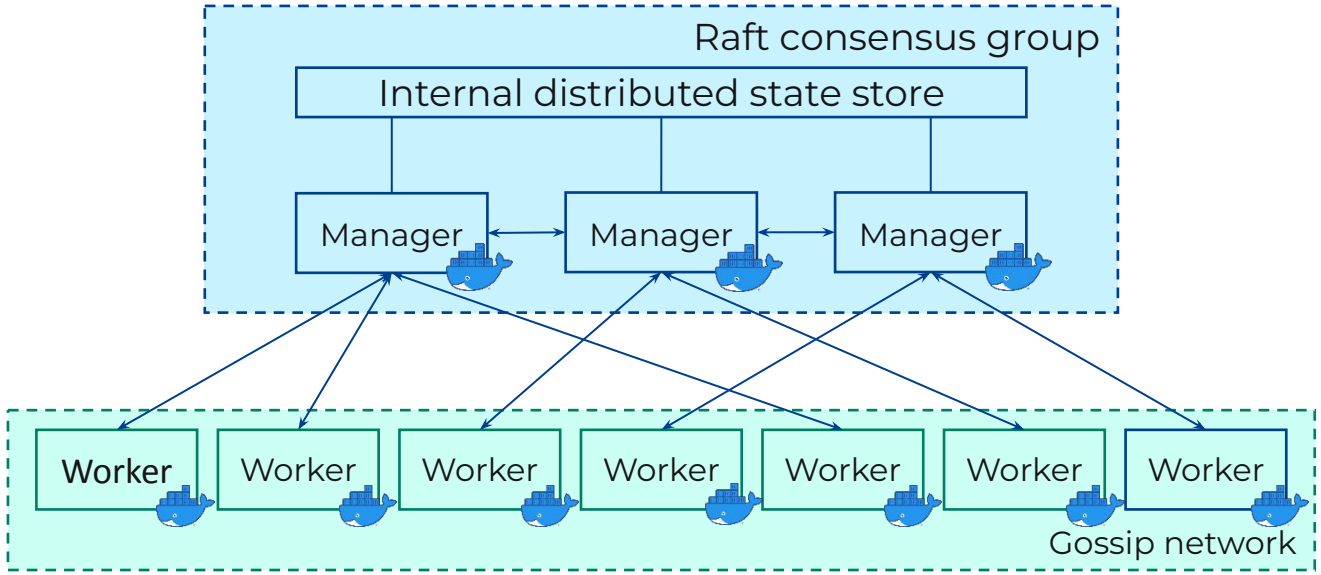


- `docker network create -d bridge mybridge`
- `docker run -d --net mybridge --name db redis`
- `docker run -d --net mybridge -e DB=db -p 8000:5000 --name web chrch/web`

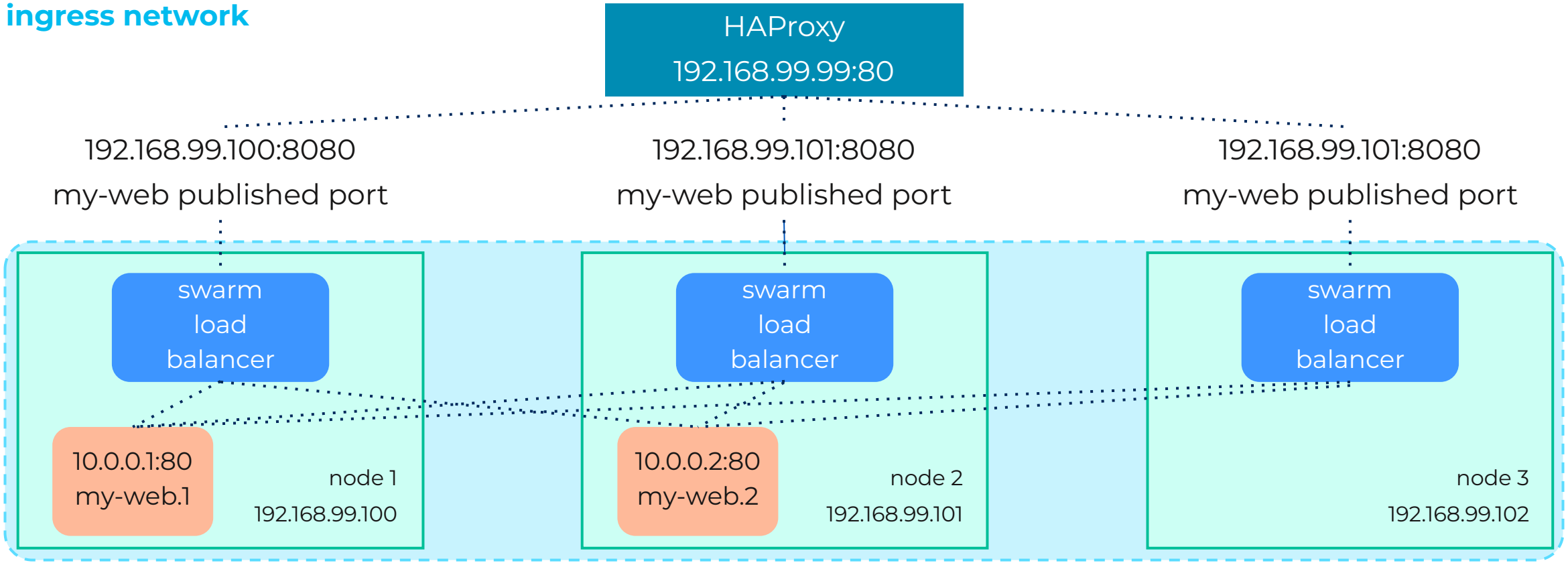
Docker Swarm

Текущие версии Docker включают режим ”роя” для управления кластером Docker Engines.

```
docker swarm init --advertise-addr <MANAGER-IP>

docker swarm join \ --token
SWMTKN-1-49nj1cmql0jkz5s954yi3oex3nedyz0fb0xx14ie
39trti4wxv-8vxv8rssmk743ojnwacrr2e7c \
192.168.99.100:2377

docker node ls


ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS

9j68exjopxe7wfl6yuxml7a7j worker1 Ready Active

dxn1zf6l61qsb1josjja83ngz * manager1 Ready Active
Leader
```



Ingress load balancing



- Все узлы участвуют во входной сети маршрутизации (ingress **routing mesh**).
- Сеть маршрутизации позволяет каждому узлу в swarm-кластере принимать соединения через опубликованные порты для любой службы, работающей в swarm-кластере, даже если на узле не выполняется никаких задач.

Технология контейнеризации Docker