

The background of the entire image is an aerial photograph of a dense city skyline, likely New York City, with numerous skyscrapers. A semi-transparent blue overlay covers the entire image. In the center, there is a horizontal band with a gradient from teal on the left to dark blue on the right. Overlaid on this band is a white network pattern of dots and lines. The title text is centered within this band.

Работа с системой контейнеризации Docker

Преподаватель



Игнатенко Филипп

Руководитель разработки PaaS-сервисов российской облачной платформы (БАЗИС)

Преподаватель на курсах DevOps, DevSecOps, Docker, Kubernetes, Linux на платформе онлайн-образования «Otus»

Спикер международных конференций



Scan me! Ignatenko Filipp

Маршрут вебинара


Виртуализация/Контейнеризация.



Введение в Docker



**Работа с данными и сетями в
Docker**

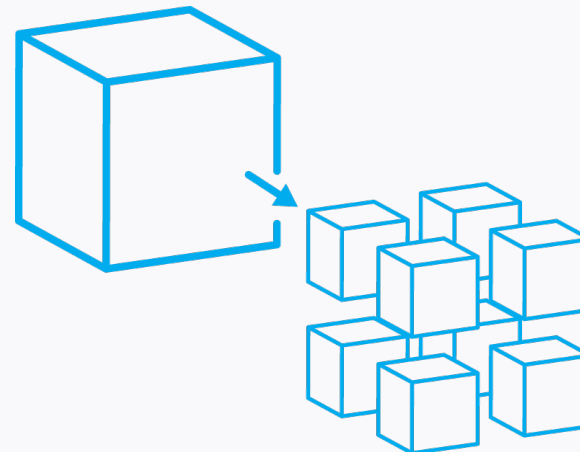


Виртуализация vs контейнеризация

Виртуализация vs контейнеризация

Каким был мир до микросервисов и контейнеров?

До появления микросервисов и контейнеров повсеместно использовались **монолитные приложения** на основе систем **виртуализации**

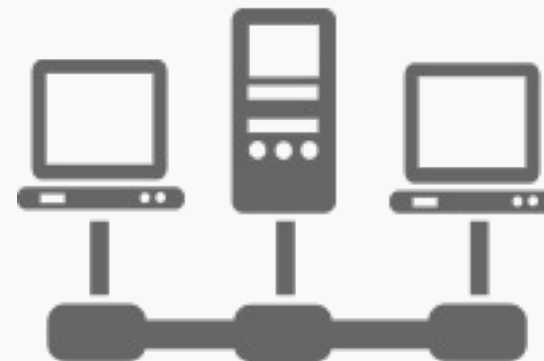


Виртуализация vs контейнеризация

Особенности монолитных приложений

Главные свойства монолитных приложений:

- Много зависимостей
- Долгая разработка (devops еще не существовало)
- Повсеместно используется **виртуализация**



Виртуализация vs контейнеризация

Виртуализация

Виртуализация – это набор вычислительных ресурсов, обеспечивающее логическую изоляцию вычислительных процессов, выполняемых на одном физическом ресурсе



Виртуализация vs контейнеризация

Виртуализация

Особенности применения виртуализации:

- Зачастую представлено одним сервером или несколькими VM
- Полная изоляция окружений
- Выделение ресурсов по запросу

Виртуализация vs контейнеризация

Виртуализация

- Vmware (многими считается флагманским продуктом)
- Xen (была поглощена Citrix)
- KVM (виртуализация в среде Linux, поглощена Red Hat, а затем IBM)
- Hyper-V (сервер виртуализации от Microsoft)

Виртуализация vs контейнеризация

Виртуализация. Гипервизор

Для обеспечения одновременного (параллельного) выполнения нескольких операционных систем на одном и том же хосте используется **гипервизор**



Виртуализация vs контейнеризация

Виртуализация. Гипервизор

Гипервизор – управляет всей виртуализацией

Гипервизор зачастую:

- Большой
- Медленный
- Платный*



Виртуализация vs контейнеризация

Новый подход в виртуализации

Назрела необходимость в *совершенно ином* подходе к построению архитектуры приложений, при котором **ядро операционной системы поддерживает несколько изолированных экземпляров пользовательского пространства*** (namespaces) вместо одного

* пользовательское пространство (namespace) - это функция ядра Linux, позволяющая изолировать и виртуализировать глобальные системные ресурсы множества процессов

Виды пространств имен в Linux: **cgroups**, IPC, Network, Mount, PID, User, UTS

Виртуализация vs контейнеризация

Возникновение контейнеризации

Так возникла новая технология виртуализации – **контейнеризация**, которая использует ядро хостовой системы, оставаясь при не менее функциональной и обеспечивающей необходимую изоляцию



Виртуализация vs контейнеризация

Различия VM от контейнера

- VM требует виртуализации железа под гостевую ОС
- Контейнер использует ядро хостовой системы
- В VM может работать любая ОС, в контейнере – не любая
- VM хорошо изолируется, контейнер - плохо

Важно! в контейнере может быть запущен экземпляр операционной системы только с тем же ядром, что и у хостовой операционной системы (все контейнеры узла используют общее ядро)

Виртуализация vs контейнеризация

Проблемы контейнеризации

Для контейнеров используется виртуализация на уровне ядра, то есть **от гипервизора можно отказаться,**

Однако:

- Контейнер использует ядро хостовой системы, а отсюда проблемы с безопасностью
- Используются большие образы контейнеров
- Нет стандарта упаковки образов и контейнеров
- Все еще множество зависимостей

Виртуализация vs контейнеризация

Как решить проблемы контейнеризации

Какими качествами должна обладать контейнеризация?

- Контейнер следует принципу единственной ответственности
- Все необходимое для работы есть в самом контейнере (в т.ч. все зависимости)
- Образы небольшого размера
- Эфемерность

Выводы: виртуализация vs контейнеризация

От виртуализации к контейнеризации

Проблемы виртуализации породили необходимость контейнеризации, но проблемы контейнеризации ставят перед нами новую задачу:

Необходимо программное обеспечение, способное работать в средах контейнеризации и удовлетворить имеющиеся требования

- Контейнер следует принципу единственной ответственности
- Все необходимое для работы есть в самом контейнере (в т.ч. все зависимости)
- Образы небольшого размера
- Эфемерность

И такой «контейнеризатор приложений» существует!



The background of the slide is a high-angle, blue-tinted aerial photograph of a dense urban skyline, likely New York City. Overlaid on this image is a semi-transparent blue band across the middle, which contains a white network diagram of interconnected nodes and lines. The title text is centered within this band.

Docker и его компоненты

Docker и его компоненты

Ключевые особенности docker

Docker позволил использовать совершенно иной подход к построению архитектуры приложений

- Один процесс – один контейнер (одна функция – одна ответственность)
- Контейнер self-contained (все что необходимо для его работы есть в самом контейнере, в т.ч. все зависимости)
- Образы не занимают много места на диске (напр., alpine)
- Контейнер – это эфемерная сущность

Docker и его компоненты

Установка docker

Docker поддерживает несколько вариантов установки, наиболее распространенный – с помощью репозитория:

```
$ sudo apt-get update  
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Обратим внимание и рассмотрим все составляющие docker

Docker и его компоненты

Из чего состоит docker?

Docker, как технологию контейнеризации с открытым исходным кодом, представляет собой клиент-серверное приложение. Условно его можно разделить на три составляющие:

- Серверную часть в виде демона (dockerd)
- API-интерфейса для взаимодействия с docker
- Command line interface (CLI)

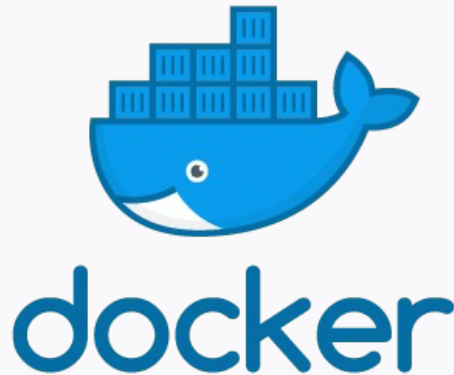
```
$ sudo apt-get update  
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Docker и его компоненты

Компоненты и сущности docker

При работе с docker вы будете оперировать следующими сущностями, обеспечивающими его функциональность:

- docker daemon
- docker cli
- dockerfile
- docker image
- docker container
- docker registry



Рассмотрим их подробнее

Docker и его компоненты

Docker daemon

Docker daemon – способ запуска контейнеров

- серверная часть
- работает на хостовой машине
- скачивает образы и запускает контейнеры
- связывает контейнеры по сети
- собирает логи
- создает образы из dockerfile

- docker daemon 
- docker cli
- dockerfile
- docker image
- docker container
- docker registry

Docker и его компоненты

Docker CLI

Docker CLI – утилита управления docker

- клиентская часть
- утилита для работы с докером
- работает локально или по сети

- docker daemon
- docker cli ←
- dockerfile
- docker image
- docker container
- docker registry

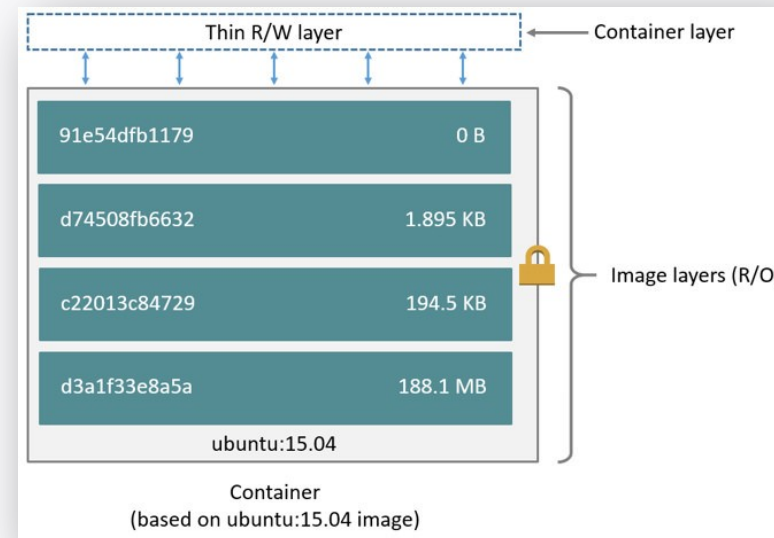
Docker и его компоненты

dockerfile

dockerfile – инструкция по сборке docker-image

- docker daemon
- docker cli
- **dockerfile** ←
- docker image
- docker container
- docker registry

- инструкция по созданию образа docker (docker-image)
- практически каждая команда – это новый слой



Docker и его компоненты

Docker image

Docker image – образ, по которому будет запущен контейнер

- упакованное приложение (будущий контейнер)
- хранится в registry (может быть скачан из него) или создается из инструкции Dockerfile
- СОСТОИТ ИЗ СЛОЕВ

- docker daemon
- docker cli
- dockerfile
- docker image ←
- docker container
- docker registry

Docker и его компоненты

Docker container

Docker container – экземпляр приложения в docker

- запускается из образа (напр., командой «docker run»)
- изолирован от других контейнеров
- self-contained (содержит все необходимое для работы)

- docker daemon
- docker cli
- dockerfile
- docker image
- docker container ←
- docker registry

Docker и его компоненты

Docker registry

Docker registry – репозиторий для хранения docker image

- хранит образы
- может быть локальным (harbor) или удаленным (docker-hub)

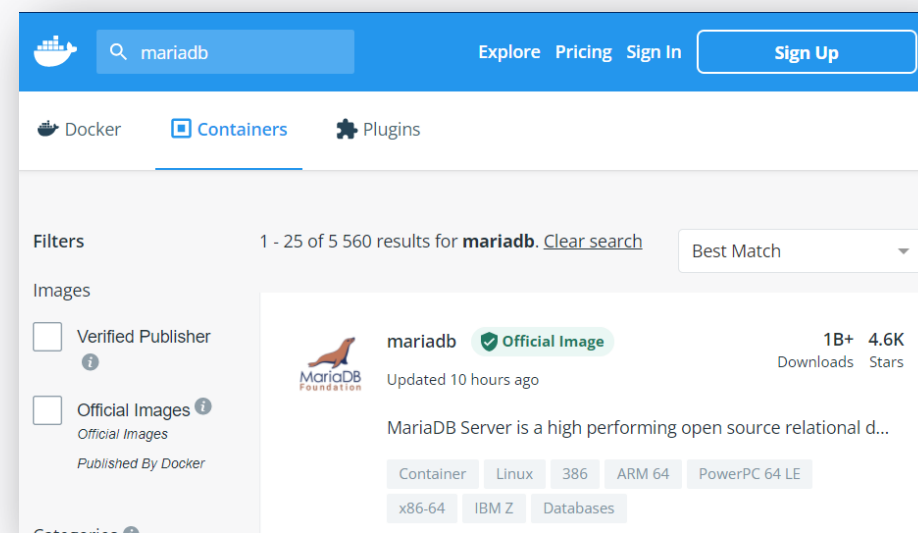
- docker daemon
- docker cli
- dockerfile
- docker image
- docker container
- docker registry ←

Docker и его компоненты

Подробнее о docker-hub

Docker-hub – это официальный репозиторий docker-image, поддерживаемый самим сообществом и предоставляющий возможность скачивать и загружать образы docker

<https://hub.docker.com/>



Итог: Docker и его компоненты

Подведем итоги

Компоненты и сущности docker

- docker daemon - запускает контейнеры
- docker cli - утилита управления docker
- dockerfile - инструкция по сбору образа
- docker image - образ докер для запуска контейнера
- docker container - экземпляр приложения в docker
- docker registry - хранилище образов



Основные команды docker

Основные команды docker

Основные команды

Для взаимодействия с docker используется клиентская часть клиент-серверного приложения docker – **docker CLI**

С помощью утилиты **docker CLI** осуществляется выполнение команд для работы с компонентами и сущностями docker

Основные команды docker

Основные команды

К основным командам docker можно отнести:

`docker pull` – скачивает образ

`docker run` – запускает контейнер на основе образа

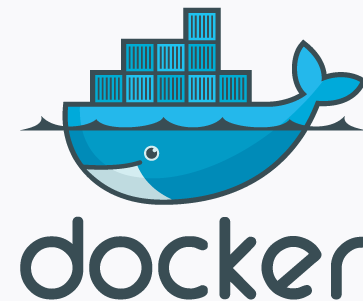
`docker ps` – вызывает список запущенных контейнеров

`docker exec` – позволяет выполнять команды в контейнере

`docker stop` – останавливает контейнер

`docker rm` – удаляет контейнер

`docker rmi` – удаляет образ





LIVE



Закрепление материала

Самостоятельная работа

Для закрепления материала выполните следующее:

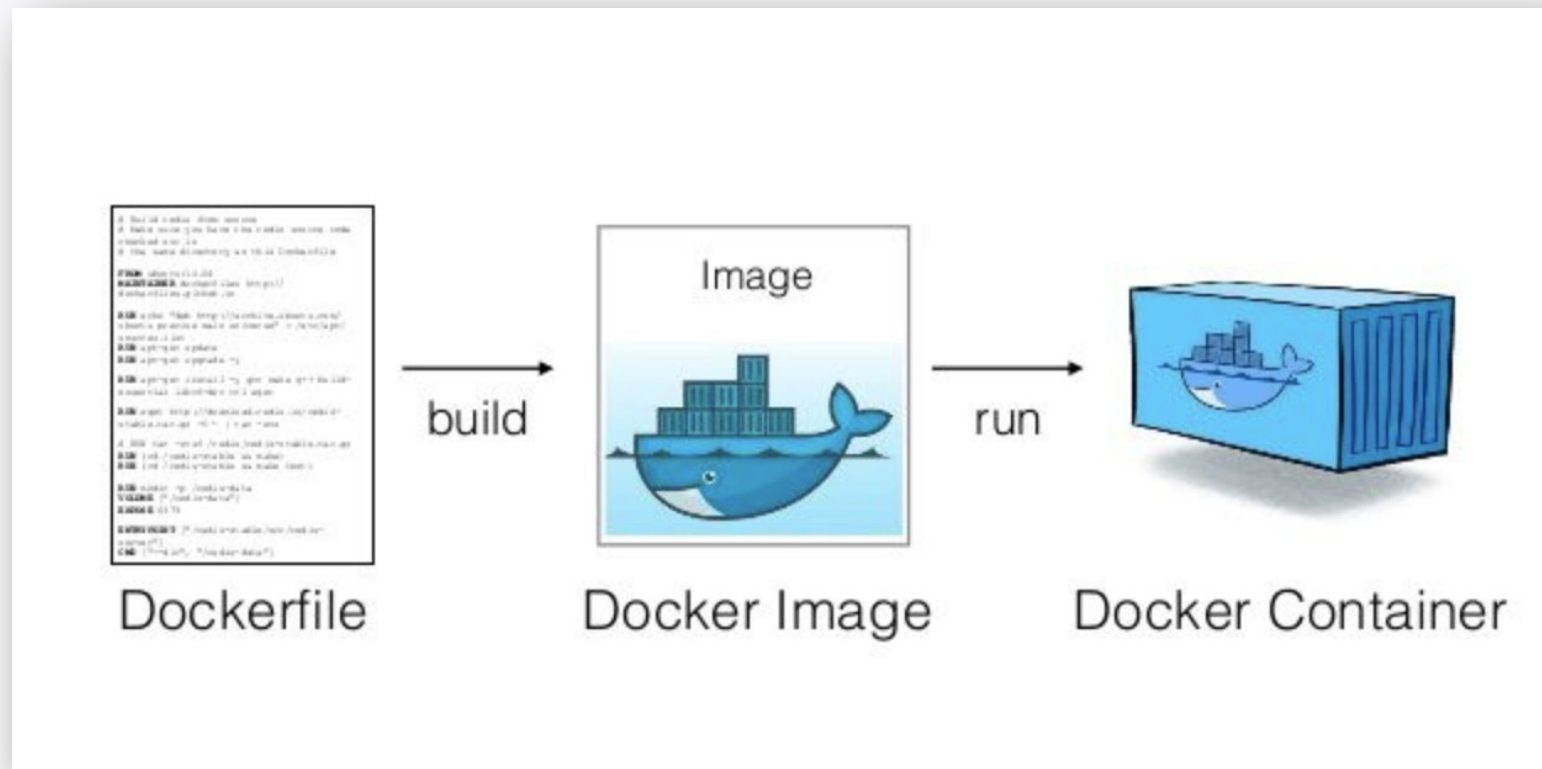
- установите docker
- зарегистрируйтесь на <https://hub.docker.com/>
- выполните основные команды docker, приведенные во время LIVE

The background of the image is an aerial view of a city skyline, likely New York City, with numerous skyscrapers. The image is overlaid with a semi-transparent blue layer that features a network of white lines and dots, resembling a digital or data network. The text "Работа с Dockerfile" is centered in the middle of the image in a large, white, sans-serif font.

Работа с Dockerfile

Работа с Dockerfile

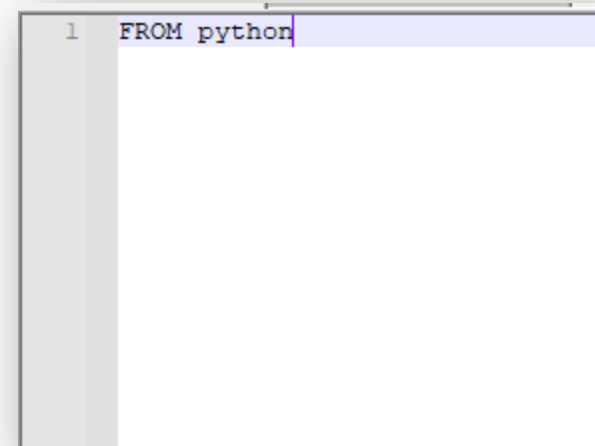
Как устроена работа с Dockerfile



Работа с Dockerfile

Пример Dockerfile

Простейший пример Dockerfile



```
1 FROM python
```

Работа с Dockerfile

Пример Dockerfile

Другой пример Dockerfile

```
1 FROM alpine
2
3 RUN apk add --update-cache
4 RUN apk add python3
5 RUN apk add python3-dev
6 RUN apk add py-pip
7 RUN apk add build-base
8 RUN pip install virtualenv
9 RUN rm -rf /var/cache/apk/*
10
```

Работа с Dockerfile

Пример Dockerfile

Оптимизированный пример Dockerfile

```
1 FROM alpine
2
3 RUN apk add --update-cache \
4     python3 \
5     python3-dev \
6     py-pip \
7     build-base \
8     && pip install virtualenv \
9     && rm -rf /var/cache/apk/*
10
```


Работа с Dockerfile

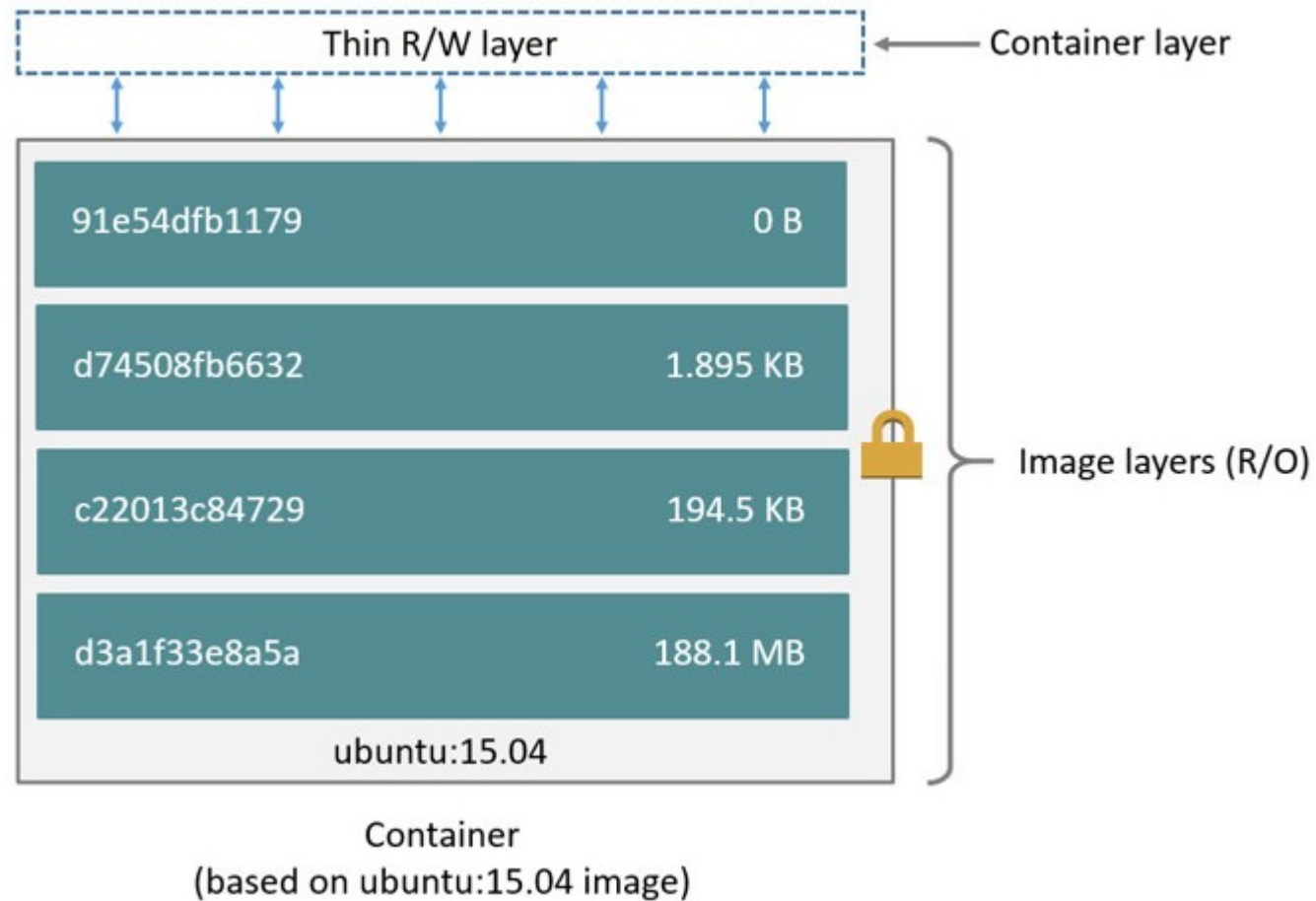
Пример Dockerfile

Визуально сравните два предыдущих примера, в чем разница?

```
1 FROM alpine
2
3 RUN apk add --update-cache
4 RUN apk add python3
5 RUN apk add python3-dev
6 RUN apk add py-pip
7 RUN apk add build-base
8 RUN pip install virtualenv
9 RUN rm -rf /var/cache/apk/*
10
```

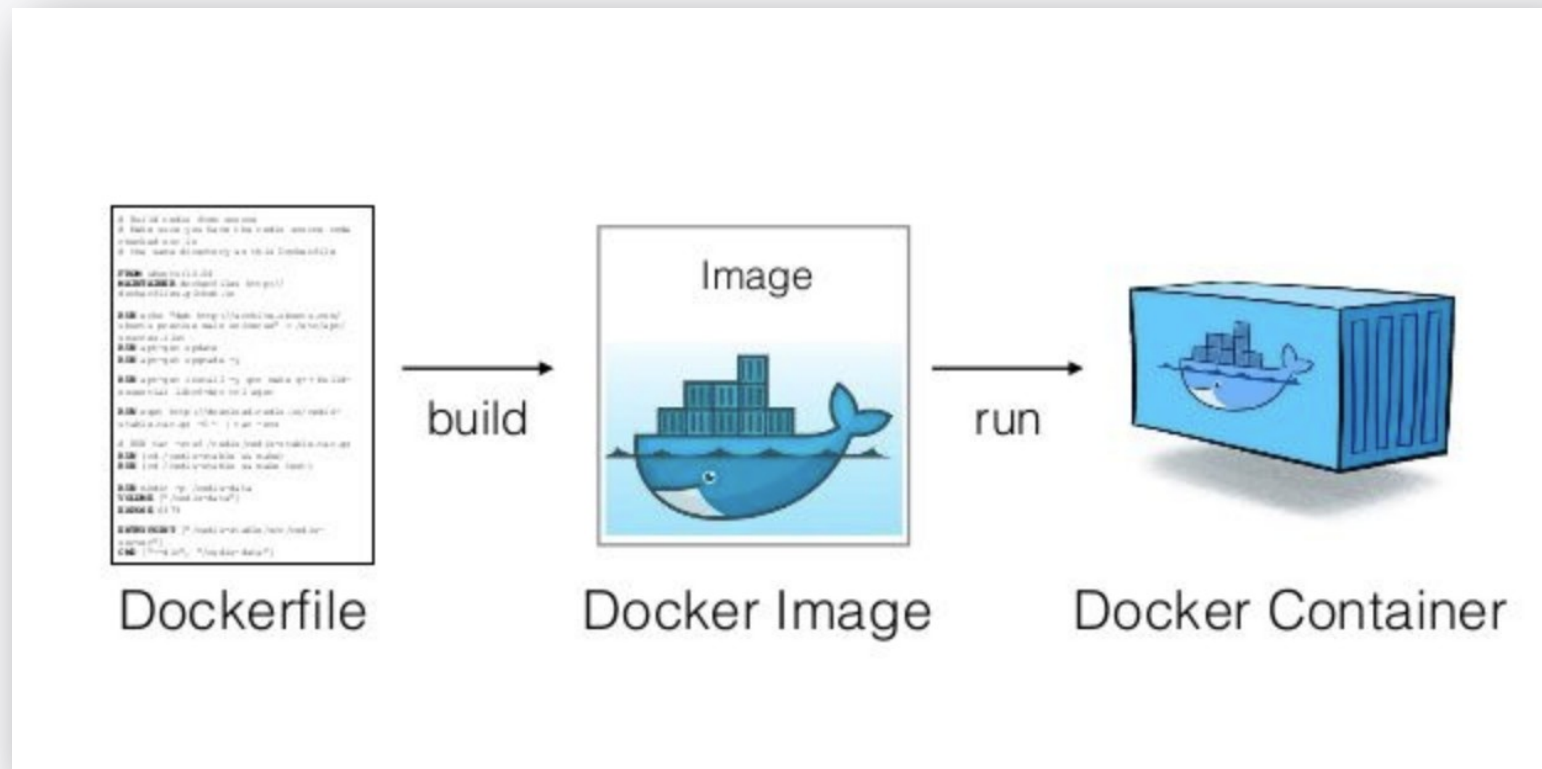
```
1 FROM alpine
2
3 RUN apk add --update-cache \
4     python3 \
5     python3-dev \
6     py-pip \
7     build-base \
8     && pip install virtualenv \
9     && rm -rf /var/cache/apk/*
10
```

Слои в Docker image



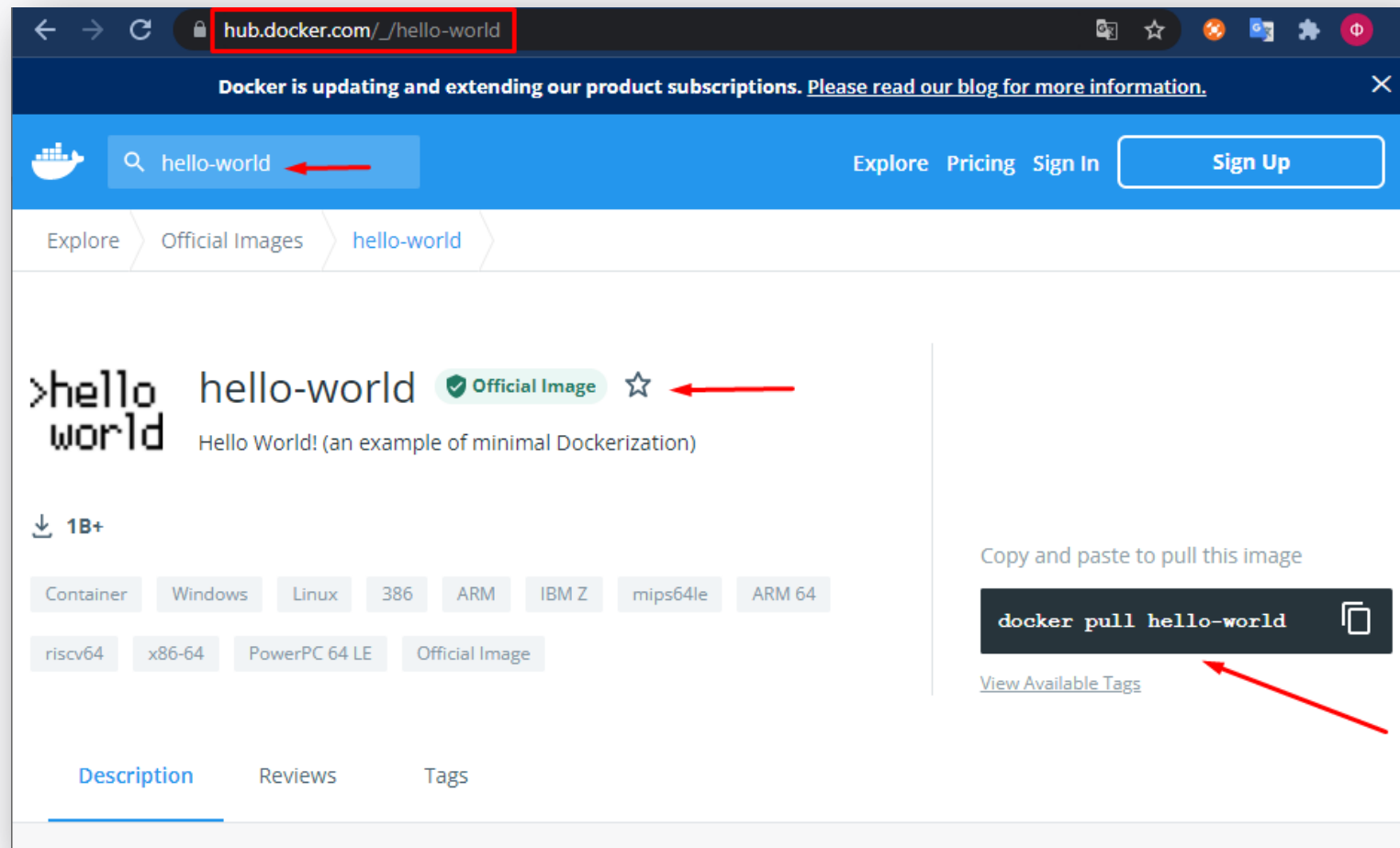
Работа с Docker контейнерами

Как запустить контейнер?



Работа с Docker контейнерами

Официальный репозиторий docker-образов «docker-hub»



Работа с Docker контейнерами

Выполните следующие команды:

```
docker pull hello-world  
docker images  
docker run hello-world
```

Получите следующий результат:

```
Hello from Docker!  
This message shows that your installation appears to be working correctly.  
  
To generate this message, Docker took the following steps:  
1. The Docker client contacted the Docker daemon.  
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.  
   (amd64)  
3. The Docker daemon created a new container from that image which runs the  
   executable that produces the output you are currently reading.  
4. The Docker daemon streamed that output to the Docker client, which sent it  
   to your terminal.
```

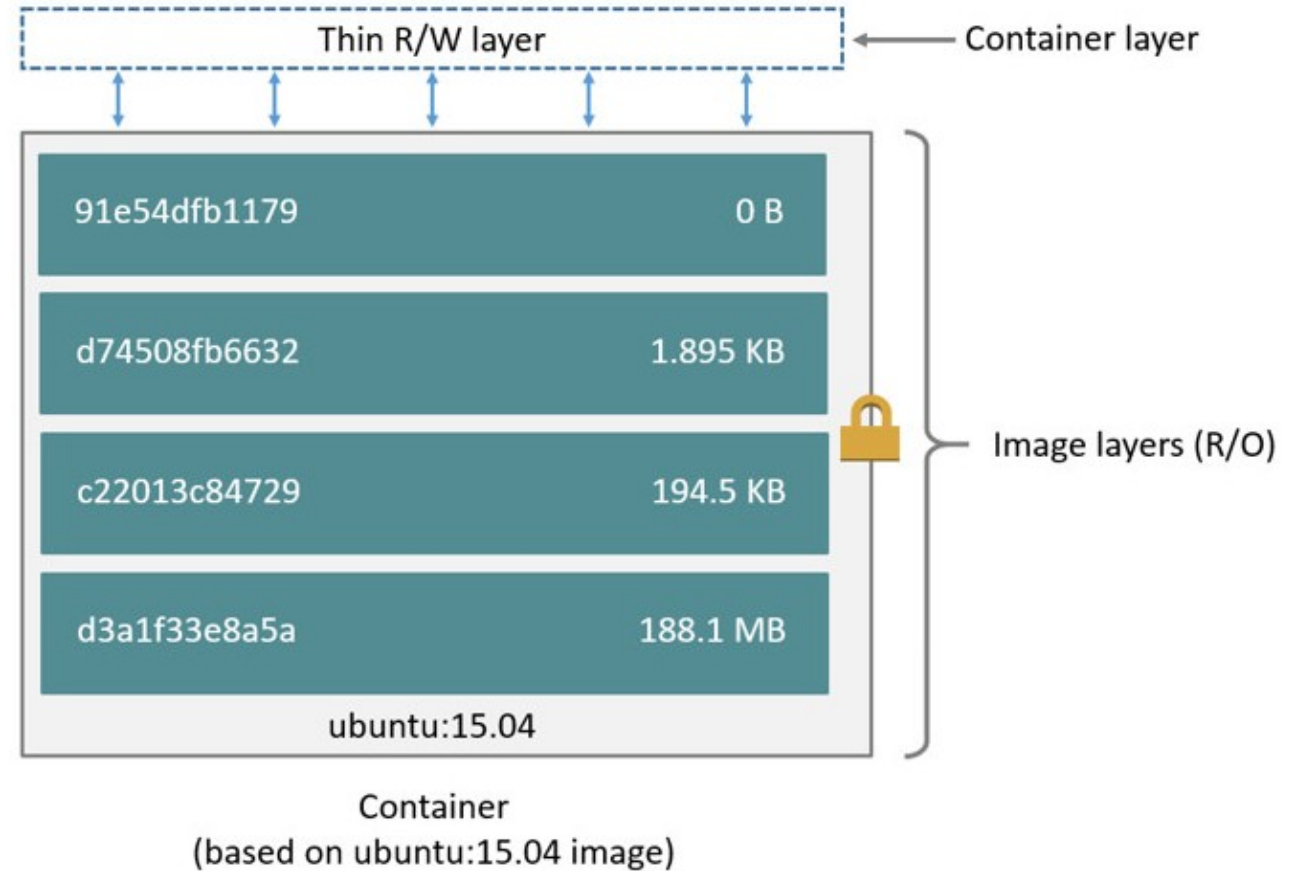

The background of the image is an aerial view of a city skyline, likely New York City, with numerous skyscrapers. The image is overlaid with a semi-transparent blue layer that features a network of white lines and dots, resembling a data or communication network. The text "Работа с данными в Docker" is written in a large, bold, white font across the center of the image.

Работа с данными в Docker

Работа с данными

Storage driver

Docker использует **драйверы хранилища** для хранения слоев изображений и хранения данных в доступном для записи уровне контейнера



Работа с данными

Storage driver types

Storage Drivers	Type
AUFS (Advanced Union Filesystem)	File level
BTRFS	Block level
DeviceMapper	Block level
Overlay	File level
Overlay2	File level
ZFS	Block level

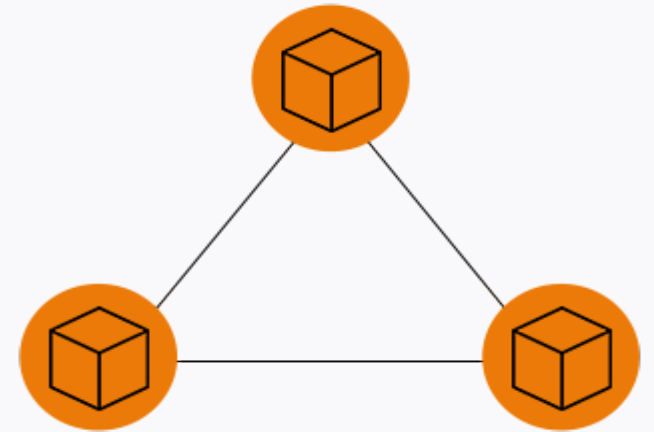
Работа с данными

Block level

Блочное хранилище разделяет данные на блоки и хранит их как отдельные части

Блочное хранилище не зависит от единственного пути к данным - в отличие от файлового хранилища, - его можно получить быстро

Пример блочных storage drivers: DeviceMapper, ZFS



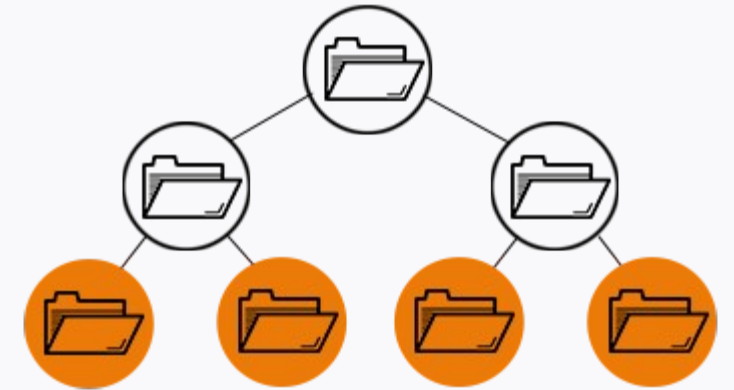
Работа с данными

File level

Файловое хранилище, также называемое хранилищем на уровне - это именно то, что вы думаете: данные хранятся в виде единого фрагмента информации внутри папки

Это самая старая и наиболее широко используемая система хранения данных для систем хранения с прямым подключением к сети, и вы, вероятно, используете ее на протяжении десятилетий

Пример файловых storage drivers: AUFS, Overlay, Overlay2



Работа с данными

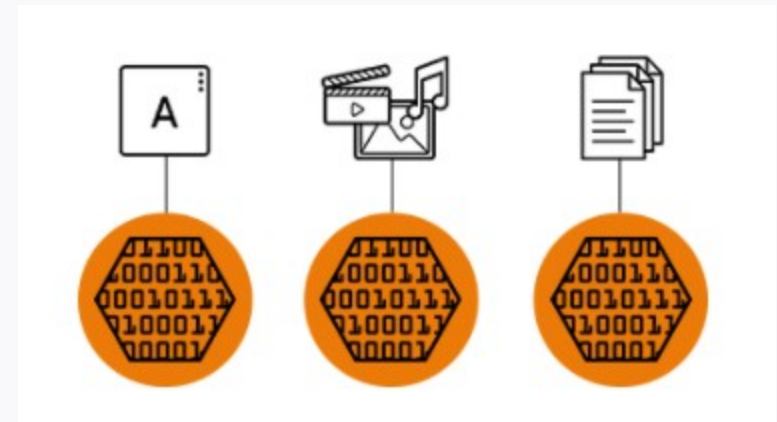
Object level

Объектное хранилище, также известное как объектно-ориентированное хранилище, представляет собой плоскую структуру, в которой файлы разбиваются на части и распределяются по оборудованию.

Для объектного хранилища требуется простой интерфейс прикладного программирования (API)

Объекты не могут быть изменены

Примеры ООХ: Сeph, OpenIO



Работа с данными

Форматы хранилищ

- Файловое хранилище организует и представляет данные в виде иерархии файлов в папках
- Блочное хранилище разбивает данные на произвольно организованные тома одинакового размера
- Объектное хранилище управляет данными и связывает их со связанными метаданными (не рассматривается в Docker)



Работа с данными

overlay2	overlay2 является предпочтительным драйвером хранилища для всех поддерживаемых в настоящее время дистрибутивов Linux и не требует дополнительной настройки
overlay	Устаревший overlay драйвер использовался для ядер, которые не поддерживали функцию «multiple-lowerdir»
Btrfs, ZFS	В btrfs и zfs драйверах имеются дополнительные опции, такие как создание «снапшотов», но требуют большего обслуживания и настройки
aufs	Aufs Драйвер запоминающего устройства был предпочтительным драйвером для хранения Docker 18.06 и старше, при работе на Ubuntu 14.04 на ядре 3.13, которая не имела никакой поддержки overlay2. Однако в текущих версиях Ubuntu и Debian теперь есть поддержка overlay2, которая теперь является рекомендуемым драйвером
devicemapper	Был для производственных сред, потому что loopback-lvm, но имел очень низкую производительность. Devicemapper был рекомендованным драйвером хранилища для CentOS и RHEL, поскольку их версия ядра не поддерживала overlay2. Однако в текущих версиях CentOS и RHEL теперь есть поддержка overlay2

Работа с данными

```
C:\Windows\system32>docker info
Client:
 Context:    default
 Debug Mode: false
 Plugins:
  buildx: Build with BuildKit (Docker Inc., v0.6.1-docker)
  compose: Docker Compose (Docker Inc., v2.0.0-rc.2)
  scan: Docker Scan (Docker Inc., v0.8.0)

Server:
 Containers: 5
  Running: 2
  Paused: 0
  Stopped: 3
 Images: 5
 Server Version: 20.10.8
 Storage Driver: overlay2
 Backing Filesystem: extfs
```

docker info

Storage Driver: overlay2
Backing Filesystem: extfs

Работа с данными

Backing Filesystem

Драйверы хранилища требуют, чтобы вы использовали определенный формат для резервной файловой системы

Backing Filesystem – это файловая система, в которой расположен `/var/lib/docker/`

Storage driver	Supported backing filesystems
<code>overlay2</code> , <code>overlay</code>	<code>xfs</code> with <code>ftype=1</code> , <code>ext4</code>
<code>fuse-overlayfs</code>	any filesystem
<code>aufs</code>	<code>xfs</code> , <code>ext4</code>
<code>devicemapper</code>	<code>direct-lvm</code>
<code>btrfs</code>	<code>btrfs</code>
<code>zfs</code>	<code>zfs</code>
<code>vfs</code>	any filesystem

Какие данные стоит хранить постоянно?

Варианты правильных ответов:

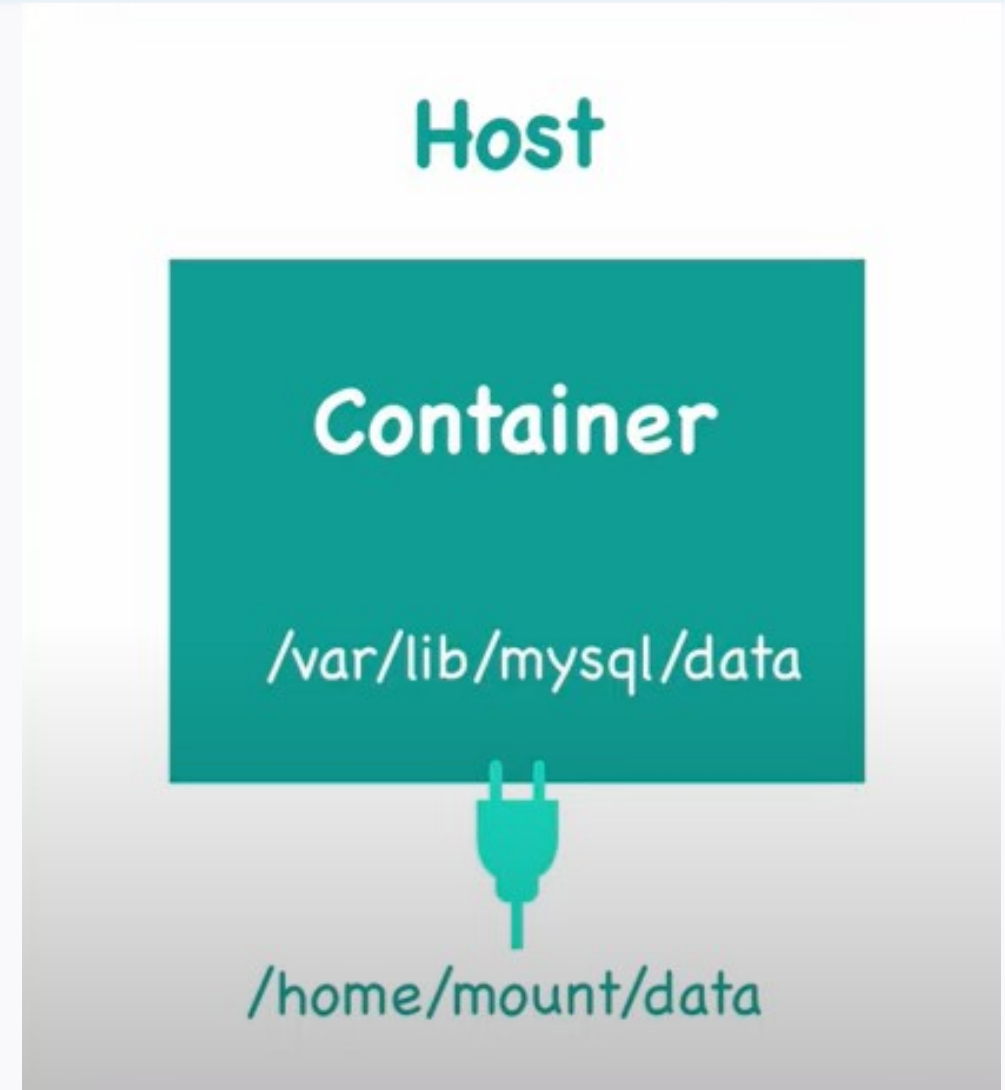
- Конфигурационные файлы
- Скачиваемые для использования плагины
- Базы данных
- Настройки пользователей
- Артефакты работы приложения
- Скрипты, которые вы используете

Работа с данными

Docker volumes

Используются для:

- данных с интенсивной записью
- данных, которые должны сохраняться по истечении срока жизни контейнера
- данных, которые должны совместно использоваться контейнерами

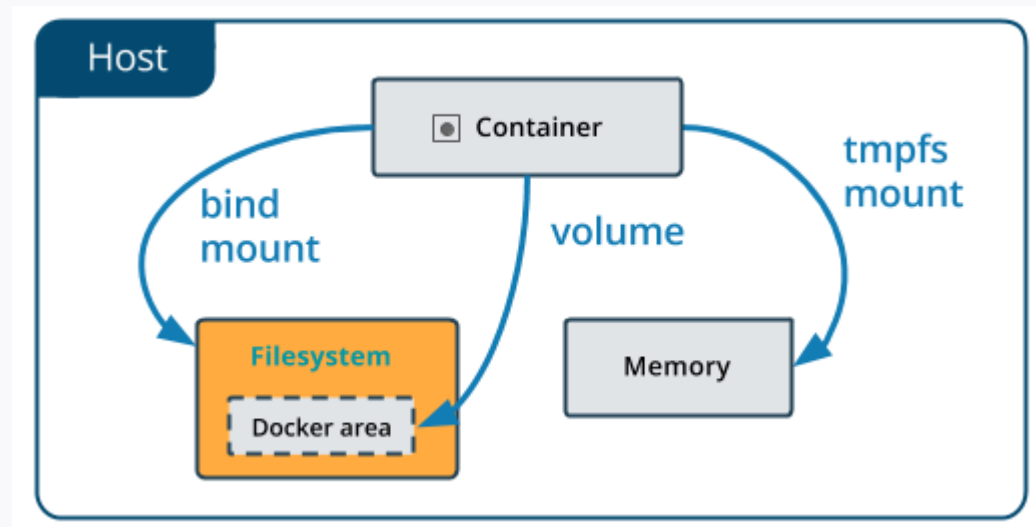


Работа с данными

Docker volumes

тома находятся по умолчанию в `/var/lib/docker/volumes/` (в linux)

Важно! Другие программы не должны получать к ним доступ напрямую, только через контейнер



Варианты использования Docker Volumes

Host volumes

```
docker run  
-v /home/mount/data:/var/lib/mysql/data
```

Варианты использования Docker Volumes

Anonymous volumes

```
docker run  
  -v /var/lib/mysql/data
```

Варианты использования Docker Volumes

Named volumes

```
docker run  
  -v name:/var/lib/mysql/data
```


Варианты использования Docker Volumes:

- Host volumes
- Anonymous volumes
- Named volumes

Работа с данными

Как это происходит в Docker-Compose

```
1 version: '3'
2 services:
3   nginx:
4     image: nginx:1.17.8
5     ports:
6       - "8080:80"
7     volumes:
8       - ./code:/code
9       - ./docker/nginx/site.conf://c/docker-volumes/nginx"
10  php:
11    build:
12      context: docker/php-fpm
13    volumes:
14      - ./code:/code
15      - ./docker/php-fpm/php.ini://c/docker-volumes/php/php.ini"
16    environment:
17      XDEBUG_CONFIG: "remote_host=192.168.220.1 remote_enable=1"
18  db:
19    image: mysql:8.0
20    environment:
21      MYSQL_DATABASE: 'base'
22    volumes:
23      - ./docker/db://c/docker-volumes/mysql"
24    ports:
25      - "3306:3306"
```

Работа с данными

Как это происходит в Docker-Compose

```
1 version: '3'
2 services:
3   nginx:
4     image: nginx:1.17.8
5     ports:
6       - "8080:80"
7     volumes: ← host volumes
8       - ./code:/code
9       - ./docker/nginx/site.conf:/c/docker-volumes/nginx"
10  php:
11    build:
12      context: docker/php-fpm
13    volumes: ← host volumes
14      - ./code:/code
15      - ./docker/php-fpm/php.ini:/c/docker-volumes/php/php.ini"
16    environment:
17      XDEBUG_CONFIG: "remote_host=192.168.220.1 remote_enable=1"
18  db:
19    image: mysql:8.0
20    environment:
21      MYSQL_DATABASE: 'base'
22    volumes: ← host volumes
23      - ./docker/db:/c/docker-volumes/mysql"
24    ports:
25      - "3306:3306"
```




Работа с сетью в Docker

Работа с сетью

Что делает Docker для обеспечения работы сети?

- Настройка правил iptables (linux)
- Управление правилами маршрутизации (win)
- Формирование пакетов и их инкапсуляция
- Шифрование
- Безопасность

Network Drivers

Docker использует специальные сетевые драйверы (**Network Drivers**).

Сетевых драйверов несколько, все они обеспечивают основные сетевые функции

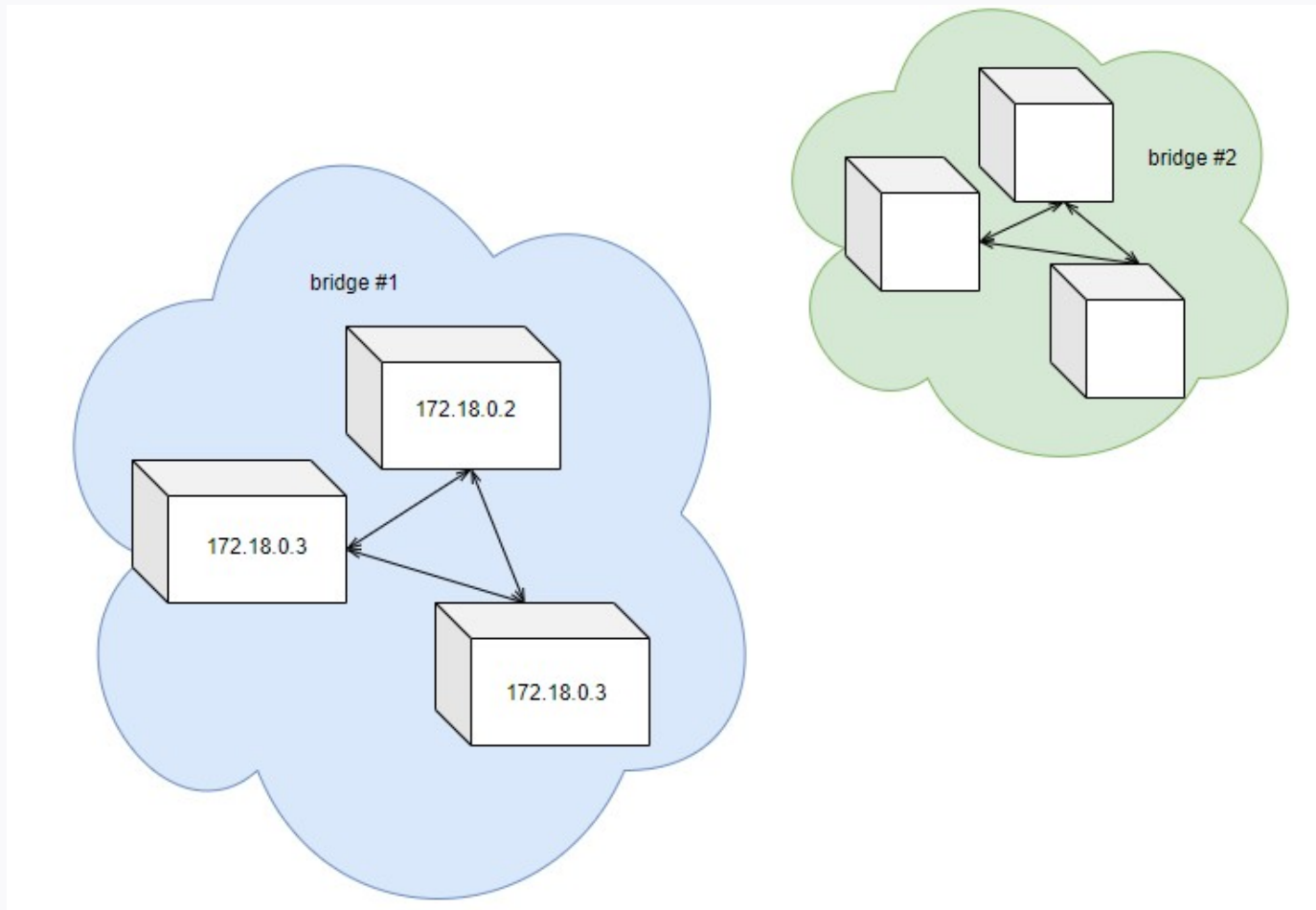
Network Drivers

- Bridge
- Host
- Overlay
- Macvlan
- None
- Network plugins*

Network Drivers

Bridge network driver (by default)

- Контейнеры, объединенные в такую сеть (bridge) могут общаться друг с другом, но не могут напрямую обратиться к контейнерам из другой bridge-сети
- Является сетевым драйвером по умолчанию



Bridge network driver

Network Drivers

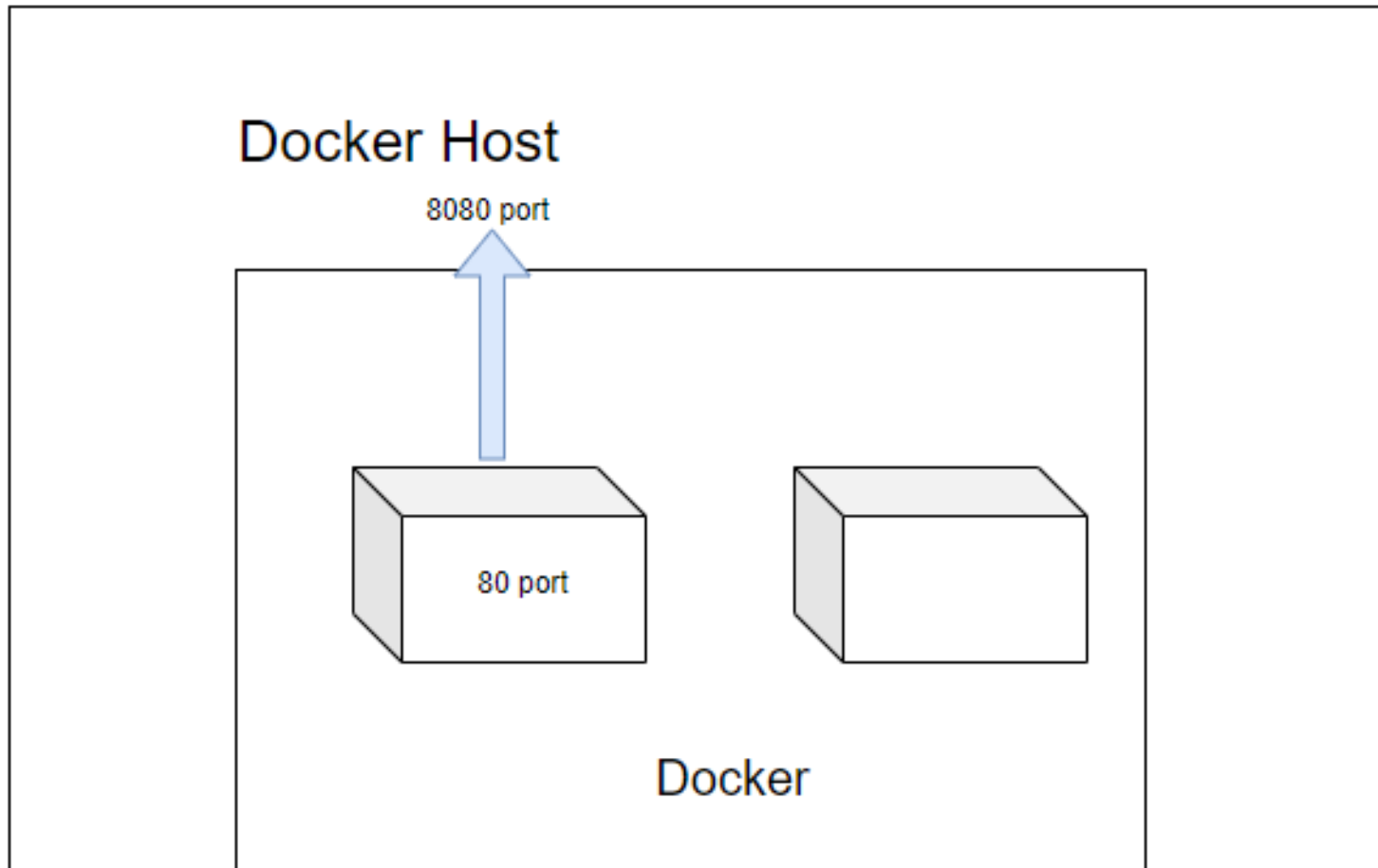
- Bridge ←
- Host
- Overlay
- Macvlan
- None
- Network plugins*

Network Drivers

Host network driver

- Драйвер удаляет сетевую изоляцию между контейнером и хостом Docker, и напрямую использует сеть хоста

Работа с сетью



Host network driver

Network Drivers

- Bridge
- Host ←
- Overlay
- Macvlan
- None
- Network plugins*

Работа с сетью

Use host networking

If you use the `host` network mode for a container, that container's network stack is not isolated from the Docker host (the container shares the host's networking namespace), and the container does not get its own IP-address allocated. For instance, if you run a container which binds to port 80 and you use `host` networking, the container's application is available on port 80 on the host's IP address.

📌 **Note:** Given that the container does not have its own IP-address when using `host` mode networking, port-mapping does not take effect, and the `-p`, `--publish`, `-P`, and `--publish-all` option are ignored, producing a warning instead:

```
WARNING: Published ports are discarded when using host network mode
```

Host mode networking can be useful to optimize performance, and in situations where a container needs to handle a large range of ports, as it does not require network address translation (NAT), and no "userland-proxy" is created for each port.

The host networking driver only works on Linux hosts, and is not supported on Docker Desktop for Mac, Docker Desktop for Windows, or Docker EE for Windows Server.

You can also use a `host` network for a swarm service, by passing `--network host` to the `docker service create` command. In this case, control traffic (traffic related to managing the swarm and the service) is still sent across an overlay network, but the individual swarm service containers send data using the Docker daemon's host network and ports. This creates some extra limitations. For instance, if a service container binds to port 80, only one service container can run on a given swarm node.

Host network driver

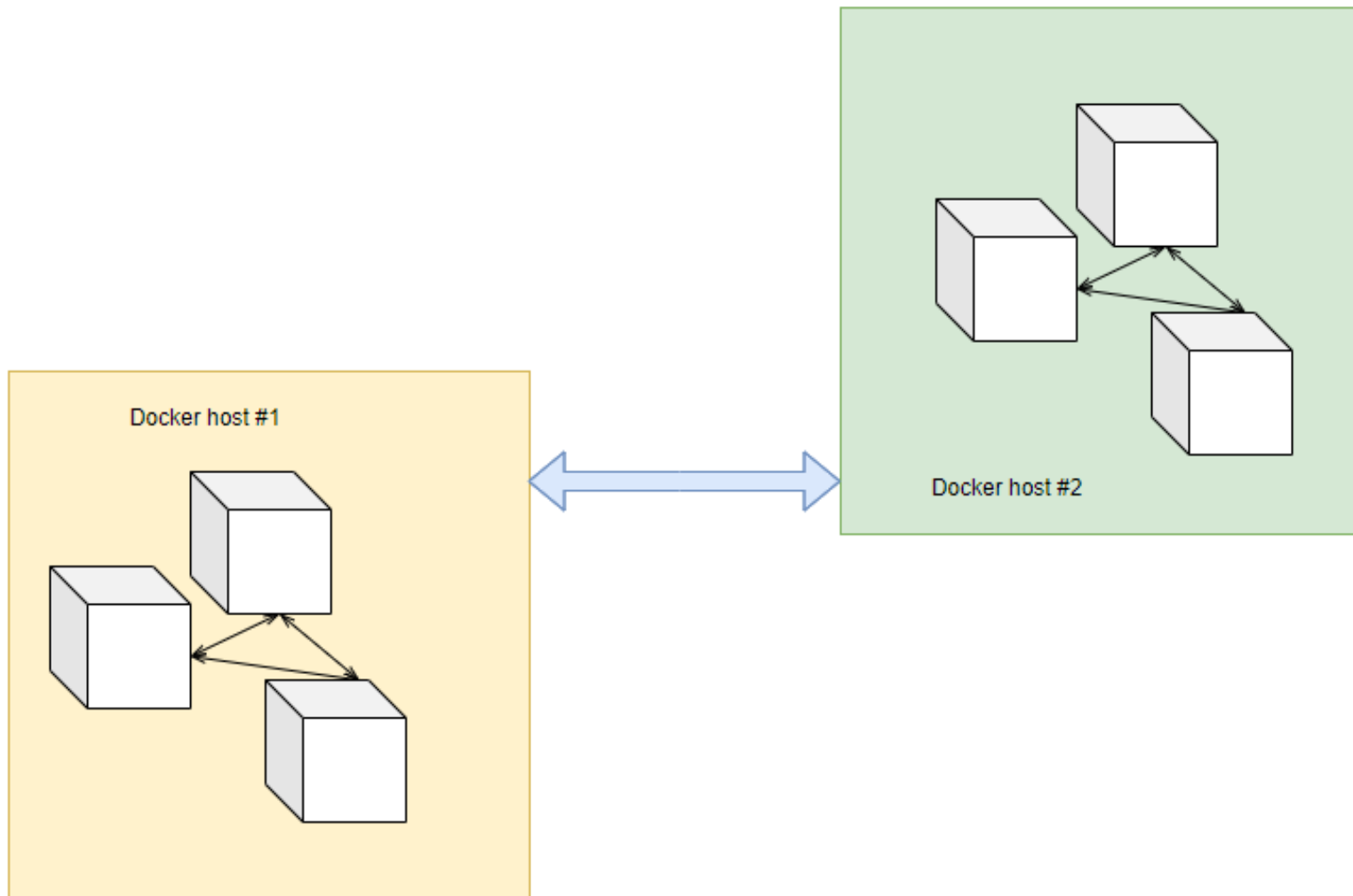
Работает только на Linux-хостах!

Network Drivers

Overlay network driver

- Объединяет сети контейнеров на разных Docker хостах

Работа с сетью



Overlay network driver

Network Drivers

- Bridge
- Host
- Overlay ←
- Macvlan
- None
- Network plugins*

Работа с сетью

Encrypt traffic on an overlay network

All swarm service management traffic is encrypted by default, using the [AES algorithm](#) in GCM mode. Manager nodes in the swarm rotate the key used to encrypt gossip data every 12 hours.

To encrypt application data as well, add `--opt encrypted` when creating the overlay network. This enables IPSEC encryption at the level of the vxlan. This encryption imposes a non-negligible performance penalty, so you should test this option before using it in production.

When you enable overlay encryption, Docker creates IPSEC tunnels between all the nodes where tasks are scheduled for services attached to the overlay network. These tunnels also use the AES algorithm in GCM mode and manager nodes automatically rotate the keys every 12 hours.

✖ Do not attach Windows nodes to encrypted overlay networks.

Overlay network encryption is not supported on Windows. If a Windows node attempts to connect to an encrypted overlay network, no error is detected but the node cannot communicate.

Overlay network driver

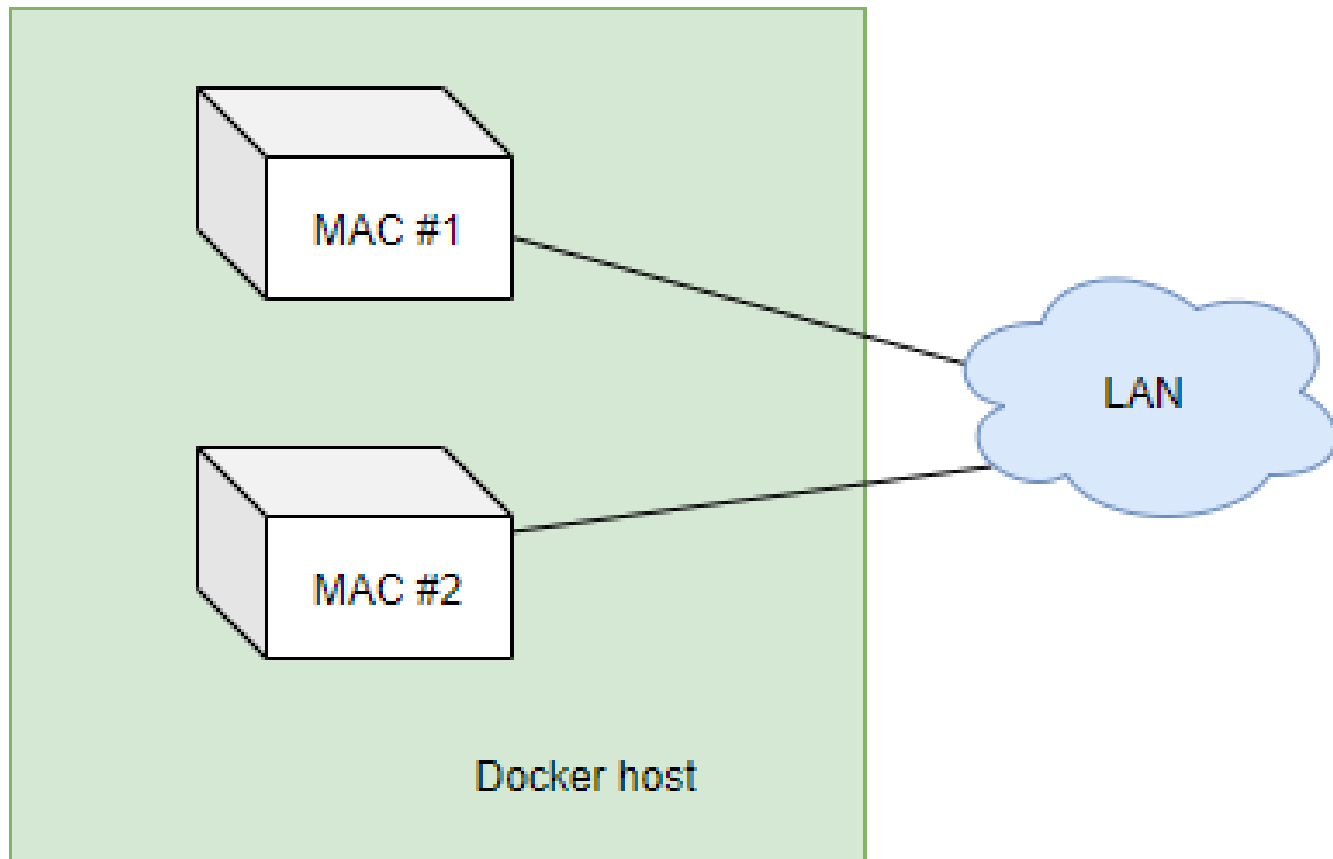
Не поддерживает шифрование на Windows хостах!

Network Drivers

Macvlan network driver

- Присваивает каждому контейнеру физический MAC-адрес, позволяющий обращаться к нему как к настоящему устройству

Работа с сетью



Macvlan network driver

Network Drivers

- Bridge
- Host
- Overlay
- Macvlan ←
- None
- Network plugins*

Работа с сетью

Networking using a macvlan network

Estimated reading time: 5 minutes

This series of tutorials deals with networking standalone containers which connect to `macvlan` networks. In this type of network, the Docker host accepts requests for multiple MAC addresses at its IP address, and routes those requests to the appropriate container. For other networking topics, see the [overview](#).

Goal

The goal of these tutorials is to set up a bridged `macvlan` network and attach a container to it, then set up an 802.1q trunked `macvlan` network and attach a container to it.

Prerequisites

- Most cloud providers block `macvlan` networking. You may need physical access to your networking equipment.
- The `macvlan` networking driver only works on Linux hosts, and is not supported on Docker Desktop for Mac, Docker Desktop for Windows, or Docker EE for Windows Server.
- You need at least version 3.9 of the Linux kernel, and version 4.0 or higher is recommended.
- The examples assume your ethernet interface is `eth0`. If your device has a different name, use that instead.

Macvlan network driver

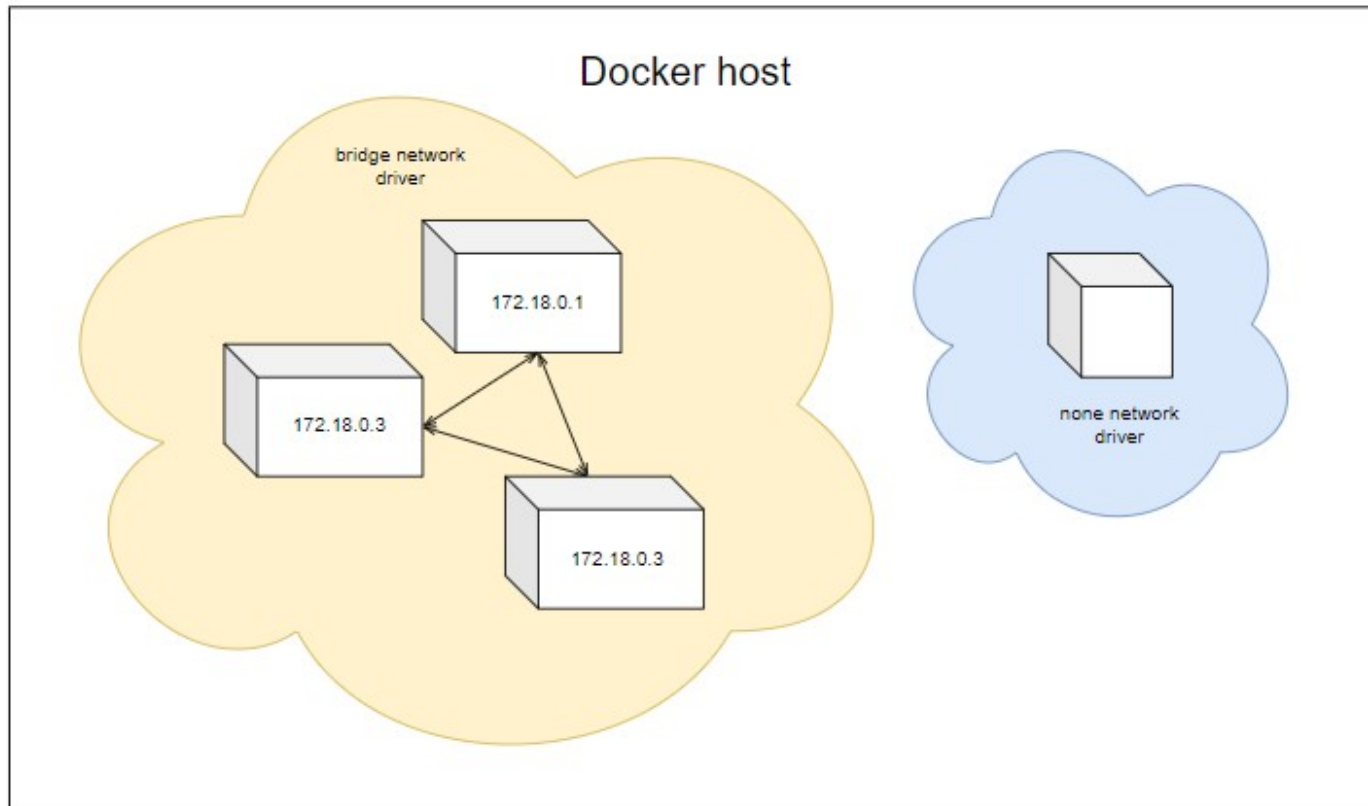
Работает только на Linux-хостах!

Network Drivers

None network driver

- Отключает все сети для контейнера, к которому применен

Работа с сетью



None network driver

Network Drivers

- Bridge
- Host
- Overlay
- Macvlan
- None ←
- Network plugins*

Network Drivers

Подключаемые сторонние сетевые плагины*

- Расширяют применимый спектр сетевых технологий, таких как VXLAN, IPVLAN, MACVLAN другие

Работа с сетью

Плагин	Описание
Contiv Networking	<ul style="list-style-type: none">- Открытый исходный код- Обеспечивает инфраструктуры и политики безопасности для развертывания мультитенантных микросервис- Интеграция с физической сетью для неконтэйнерных рабочих нагрузок
Kuryr Network	<ul style="list-style-type: none">- Разработан в рамках проекта OpenStack Kuryr- Реализует API удаленного драйвера сети Docker (libnetwork) с помощью Neutron (сетевой службы OpenStack)- Включает драйвер IPAM.
Weave Network	<ul style="list-style-type: none">- Создает виртуальную сеть, которая соединяет ваши контейнеры Docker на нескольких хостах или в облаках- Обеспечивает автоматическое обнаружение приложений- Сети Weave устойчивы, допускают разделение, безопасны и работают в частично связанных сетях и других неблагоприятных средах- Относительно легко настраивается

Network plugins

Network Drivers

- Bridge
- Host
- Overlay
- Macvlan
- None
- Network plugins* ←

Резюмируем: Network Drivers

- **Bridge** – объединяет контейнеры в bridge-сеть, является сетевым драйвером **по умолчанию**
- **Host** – использует сеть хоста (только Linux!)
- **Overlay** – объединяет сети на разных хостах (не поддерживает шифрование на Windows хостах!)
- **Macvlan** – присваивает контейнеру MAC-адрес (только Linux!)
- **None** – отключает все сети
- Network plugins* - сторонние сетевые плагины

Работа с данными

```
C:\docker>docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
f68738c7934c        bridge             bridge             local
f92a82d70427        host              host              local
d90284917c5c        none             null              local
```

docker network ls

- Выполните команду в консоли
- Найдите доступные сети bridge, host, none
- Нашли ли вы какие-то еще доступные сети?

The image features a high-angle, aerial view of a dense urban skyline, likely New York City, with numerous skyscrapers and buildings. The entire image is overlaid with a semi-transparent blue filter. A prominent network pattern, consisting of white dots connected by thin white lines, is visible across the entire frame, particularly concentrated in the central area where the text is located. The word "LIVE" is written in large, bold, white, sans-serif capital letters, centered horizontally and vertically.

LIVE

Спасибо за внимание!



Игнатенко Филипп

Руководитель разработки PaaS-сервисов российской облачной платформы (БАЗИС)

Преподаватель на курсах DevOps, DevSecOps, Docker, Kubernetes, Linux на платформе онлайн-образования «Otus»

Спикер международных конференций



Scan me! Ignatenko Filipp