Виртуализация

Виртуализация на уровне железа

Виртуализация

Виртуализация на уровне ядра
(на уровне операционной системы)

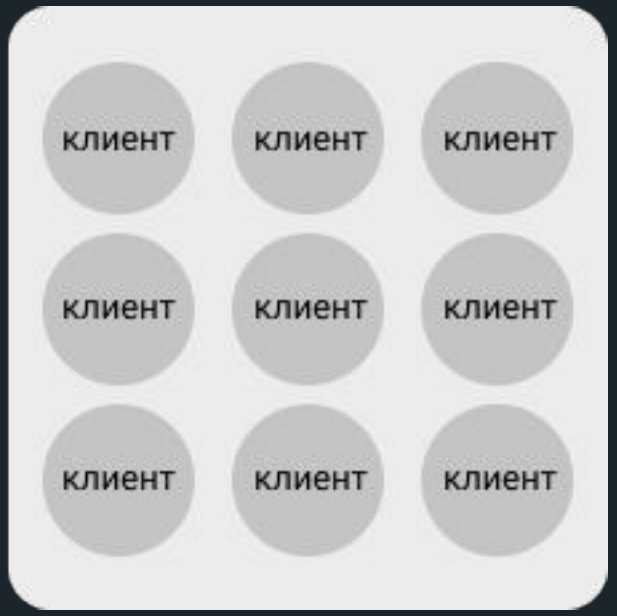
Systemd-nspawn

Виртуализация

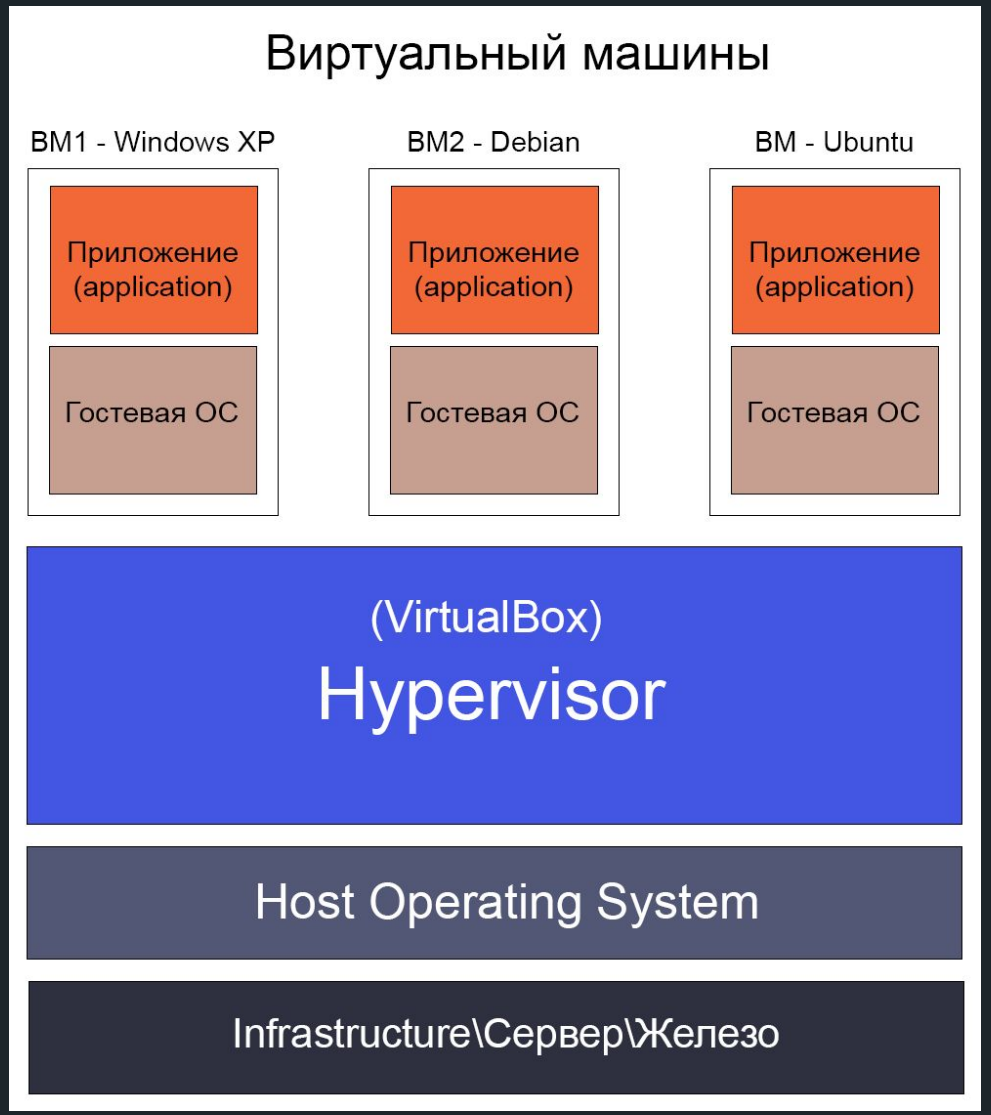
Что контейнеры, что вир. машины - это все виртуализация:

Виртуализация появилась как средство уплотнения окружений на одном и том же железе. Сначала программный продукт выполнялся на железном сервере. Потом, чтобы иметь возможность поселять в одно и то же железо больше клиентов, чтобы максимально полно утилизировать производительные мощности, придумали виртуализацию. Теперь на одном и том же железе можно держать несколько окружений.

Сервер:
Memory - 64GB
CPUs - 32
SSD - 1TB
1ый случай % исп. 5%
2ой случай % исп. 90%



Виртуализация на уровне железа



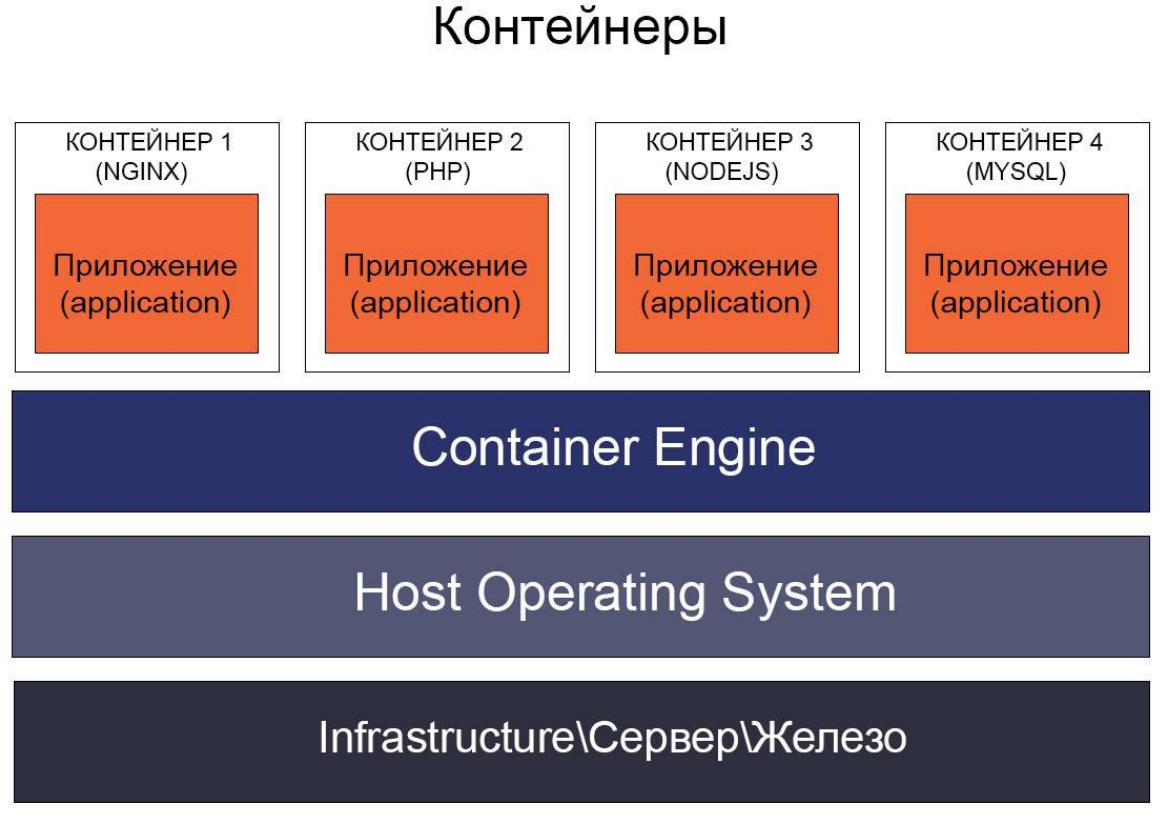
Гипервизор - или монитор виртуальных машин

В VM нельзя иметь ресурсов больше чем на хостовой системе

В VM полностью изолированное ядро, все библиотеки, строго ограниченные ресурсы по процессору и памяти.

Выделяем ресурсы

Виртуализация на уровне ядра



Container engine — это часть ПО, которое принимает пользовательские запросы, создает\ запускает контейнер.

Существует множество контейнерных движков, включая Docker, RKT, CRI-O и LXD.

Легковесная штука по сравнению с гипервизором практически нет оверхеда

Если в контейнере Linux, то и снаружи тоже Linux, на хост машине

Ограничиваем ресурсы

Отличия

Виртуальная машина:

- Подразумевает виртуализацию железа для запуска гостевой ОС
- Может работать любая ОС
- Хорошо для изоляции

Контейнер:

- Использует ядро хостовой системы
- В контейнере только Linux (не давно и Windows)
- Контейнер для изоляции плохо - т.к можно выбраться на хостовую машину

Рабочее окружение

Для разработки бекенда часто нужные все возможные сервисы, например:

- PHP
- Nginx(Apach) – сервер
- Mysql - бд
- PostgreSQL - бд
- MongoDB - NoSQL бд
- Redis - NoSQL бд
- Memcache - кэширования данных в оперативной памяти
- Nodejs
- RabbitMQ - брокер сообщений
- Mailer – почтовый сервер для тестирования почты
- ….

Рабочее окружение

1. Использование «шпаргалок»

Разработчик под каждый сервис пишет свою инструкции, как установить и как настроить (файлы конфига и т.л)

Все сервисы устанавливаются на компьютере разработчика
Хорошо работает когда разработчик один

Минусы:

- Как поднять такое же окружение другому разработчику?
- Разные ОС
- Если разработка на Linux, то сложно поднять окружение на MacOs, Windows
- Как синхронно обновлять конфигурации сервисов и их настройки
- Разработка на разные устройствах(ноутбук, компьютер)
- Большое время воссоздания рабочего окружения
- Деплой

Рабочее окружение

Пример – установка сервиса MeiliSearch

1. echo "deb [trusted=yes] https://apt.fury.io/meilisearch/ /" | sudo tee /etc/apt/sources.list.d/meilisearch.list
2. sudo apt update
3. sudo apt install -y meilisearch-http

4. sudo nano /etc/systemd/system/meilisearch.service

```
[Unit]
Description=MeiliSearch search engine
After=network.target

[Service]
ExecStart=/usr/bin/meilisearch --http-addr 0.0.0.0:7700 --env production --master-key pwd123
Restart=always

[Install]
WantedBy=multi-user.target
```

5. sudo service meilisearch start
6. sudo systemctl enable meilisearch

Рабочее окружение

2. Разработка через виртуальные машины – виртуализация на уровне железа

Одна ОС, например ubuntu 18.04
Все инструкции и скрипты по установке ПО, запускались в вир. машине

- Настройки вир. машины (в ручную в большинстве случаев)
- Выделить кол. ядер,
- Выделить память,
- Настроить сети,
- Настроить сетевые папки
- И т.д

Рабочее окружение

3. Vagrant

Инструмент который позволяет автоматизировать процесс создания и настройки вир. машин

Файл конфига Vagrantfile – все инструкции по установке образа и настройки виртуальной машины

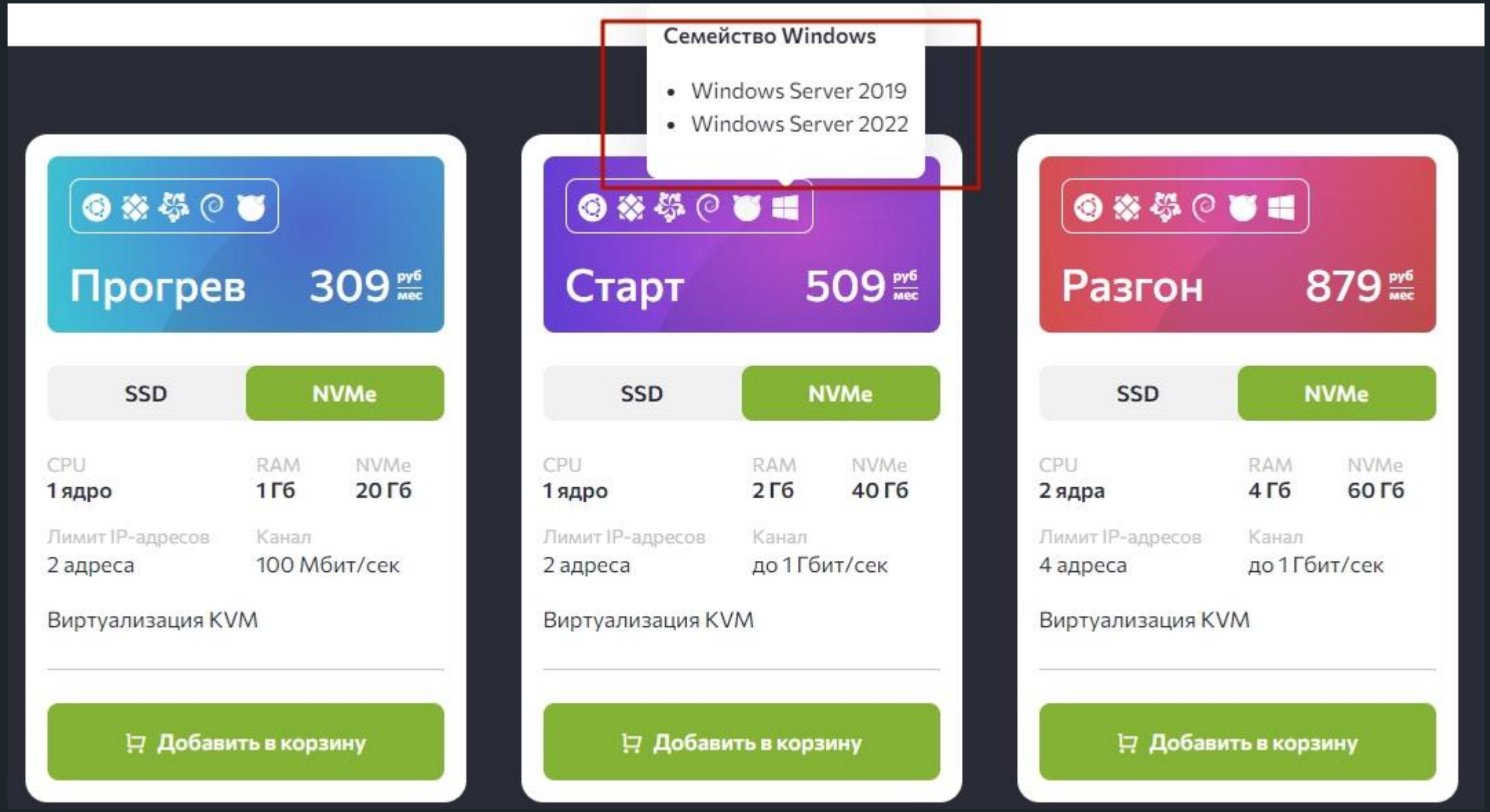
Примеры команд:
vagrant up запускалась вирт. машина - запускались наши скрипты - уст. сервисы и настраивались
vagrant halt - выключал
vagrant reload - перезагрузить машины
vagrant destroy - уничтожение машины
vagrant ssh - консоль вир. машины

Рабочее окружение

Vagrantfile

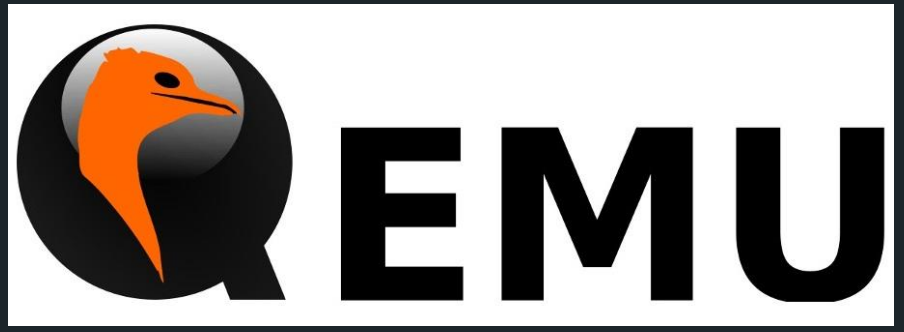
```
# vagrant configurate
Vagrant.configure(2) do |config|
  # select the box
  config.vm.box = 'bento/ubuntu-18.04'

  # should we ask about box updates?
  config.vm.box_check_update = options['box_check_update']

  config.vm.provider 'virtualbox' do |vb|
    # machine cpus count
    vb.cpus = 1
    # machine memory size
    vb.memory = 1024
    # machine name (for VirtualBox UI)
    vb.name = Project1
  end

……….
 # provisioners
  config.vm.provision 'shell', path: './vagrant/provision/once-as-root.sh', args: [options['timezone'], options['ip']]
  config.vm.provision 'shell', path: './vagrant/provision/once-as-vagrant.sh', args: [options['github_token']], privileged: false
  config.vm.provision 'shell', path: './vagrant/provision/always-as-root.sh', run: 'always'

  # post-install message (vagrant console)
  config.vm.post_up_message = "App URL: http://#{domains[:app]}"
```

Пример VDS


Семейство Windows
Windows Server 2019
Windows Server 2022
Прогрев 309 руб мес
SSD NVMe
CPU 1 ядро
RAM 1 Гб
NVMe 20 Гб
Лимит IP-адресов 2 адреса
Канал 100 Мбит/сек
Виртуализация KVM
Добавить в корзину
Старт 509 руб мес
SSD NVMe
CPU 1 ядро
RAM 2 Гб
NVMe 40 Гб
Лимит IP-адресов 2 адреса
Канал до 1 Гбит/сек
Виртуализация KVM
Добавить в корзину
Разгон 879 руб мес
SSD NVMe
CPU 2 ядра
RAM 4 Гб
NVMe 60 Гб
Лимит IP-адресов 4 адреса
Канал до 1 Гбит/сек
Виртуализация KVM
Добавить в корзину

Виртуализация

Виртуализация на уровне железа



Оверхед на гипервизор, большой образ, медленно

Виртуализация

Виртуализация на уровне ядра
(на уровне операционной системы)

Systemd-nspawn

Большие образы, нет стандарта упаковки, проблема зависимости

Пример LXC

LXC — системы контейнеров – виртуализация на уровне ядра

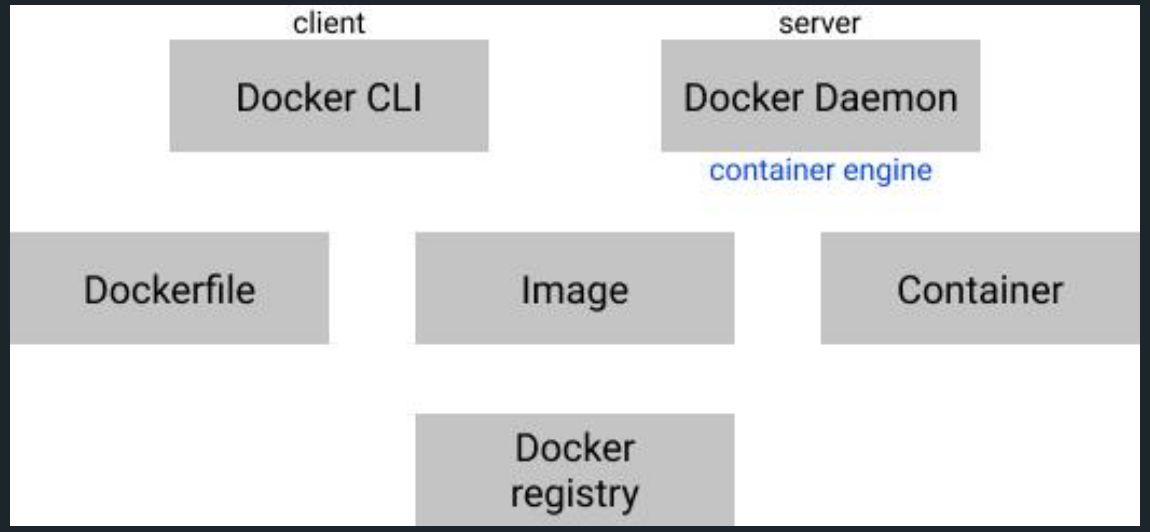
```
lxc launch ubuntu:18.04 web
lxc exec web -- apt update
lxc exec web -- apt install apache2
lxc exec web -- apt install mysql-server
lxc exec web -- apt install php
```

```
lxc launch ubuntu:18.04 db
lxc exec db -- apt install -y mysql-server
```

Переходим к докеру

- Меняет философию работы контейнеров
- 1 процесс - 1 контейнер
- Все зависимости в контейнере
- Чем меньше образ - тем лучше
- Инстансы становятся эфемерными - (недолговечными) - контейнер можно остановить и уничтожить, а затем перестроить
- Стандартизация упаковки приложения, приложение можно упаковать, отправить в репозиторий \ скачать, развернуть
- Гарантирует воспроизводимость, на люб. хостовых машинах
- Минимум или нет оверхеда
- Развертывание контейнеров Docker позволяет быстро и изолированно создавать несколько сред, включая среды, необходимые для вашего конвейера CI/CD, такие как среды сборки и тестирования

Из чего состоит докер



- Docker cli - утилита по упр. докером - исп. для запуска команд – клиент
- Docker daemon - container engine – сервер
- Dockerfile - инструкция как собирать образ
- Image – образы
- Container – контейнер который запускаются на основе образов

Объекты 1ого класса докера

Images (образ) - доступный только для чтения шаблон который содержит набор инструкций для создания контейнера Docker - можно считать как класс

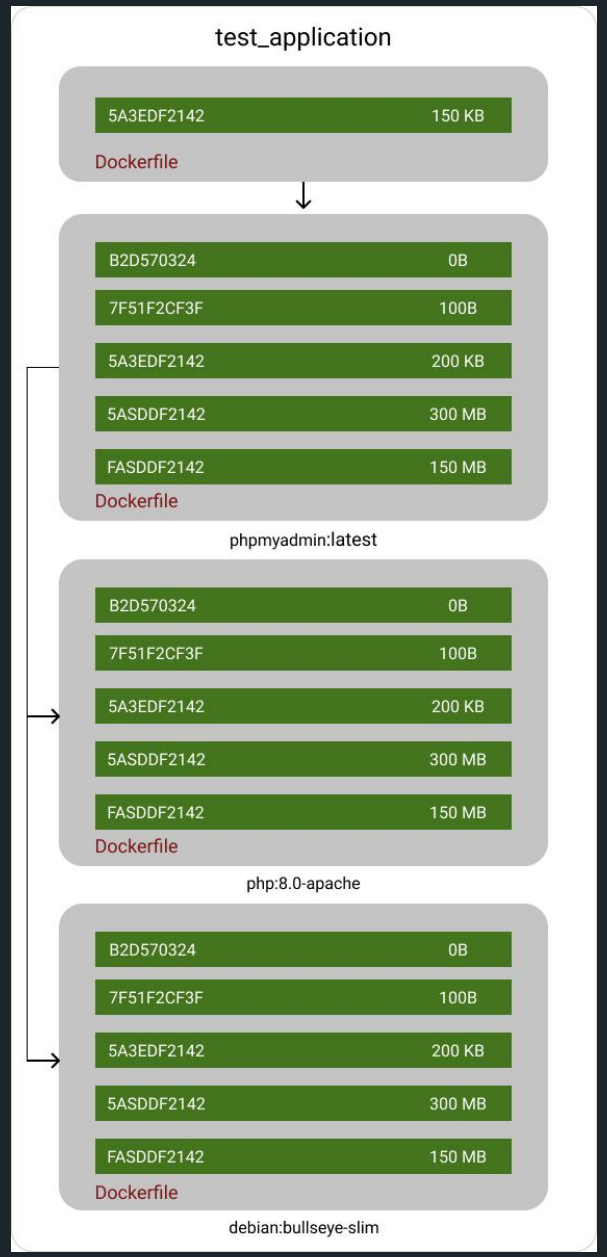
- Упаковка нашего контейнера
- Из них запускается контейнеры
- Хранятся в докер реестрах (registry - docker hub)
- Имееют hash (sha256), имя и таг
- Имеет слоенную архитектуру
- Создаются по инструкциям dockerfile

Объекты 1ого класса докера

Containers - работоспособный экземпляр или инстанс образа из которого он был создан. Когда создается контейнер из образа, оба они становятся зависимыми др от др нельзя удалить образ, пока сущ. др контейнеры

- Запускается из образа
- Изолирован
- Должен содержать в себе все для работы приложения
- 1 процесс - 1 контейнер (но в практике не всегда)

Образ



Образы строятся из множества слоев, все размещаются др. над другом
Вместе они представляют единый объект

- Каждый слой представляет собой отдельную инструкцию из докерафайла образа
- Все слои доступы только для чтения, за иск. последнего
- Каждый слой это набор отличий от слоя который был предыдущим
- Когда создается контейнер он добавляет новый доступный для записи слой поверх всех др. слоев - это слой контейнера

Только инструкции RUN, COPY, ADD создают слои. Другие инструкции не увеличивают размер сборки.

Пример

Dockerfile:

```
FROM phpmyadmin:latest
RUN apt-get -y install curl && rm -rf /var/lib/apt/lists/*

ENTRYPOINT [ "/docker-entrypoint.sh" ]
CMD ["apache2-foreground"]
```

```
docker build -t custom_phpmyadmin .
```

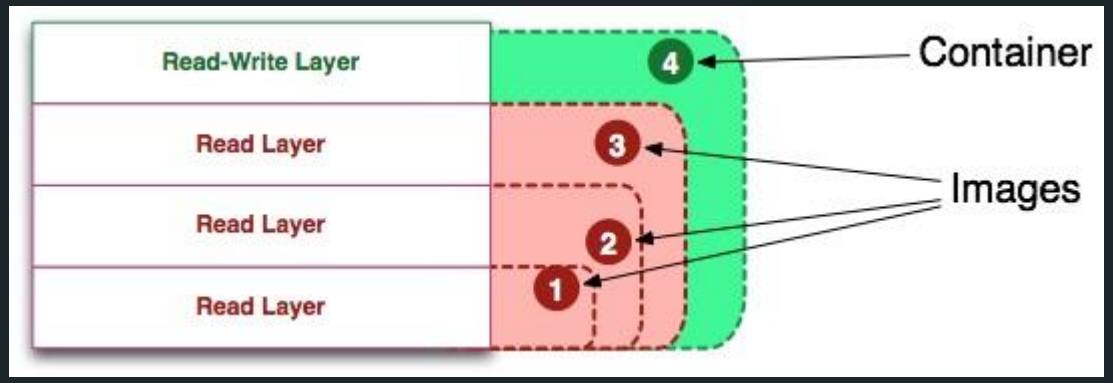
```
docker run --name container_phpmyadmin --rm -p 8080:80 custom_phpmyadmin
```

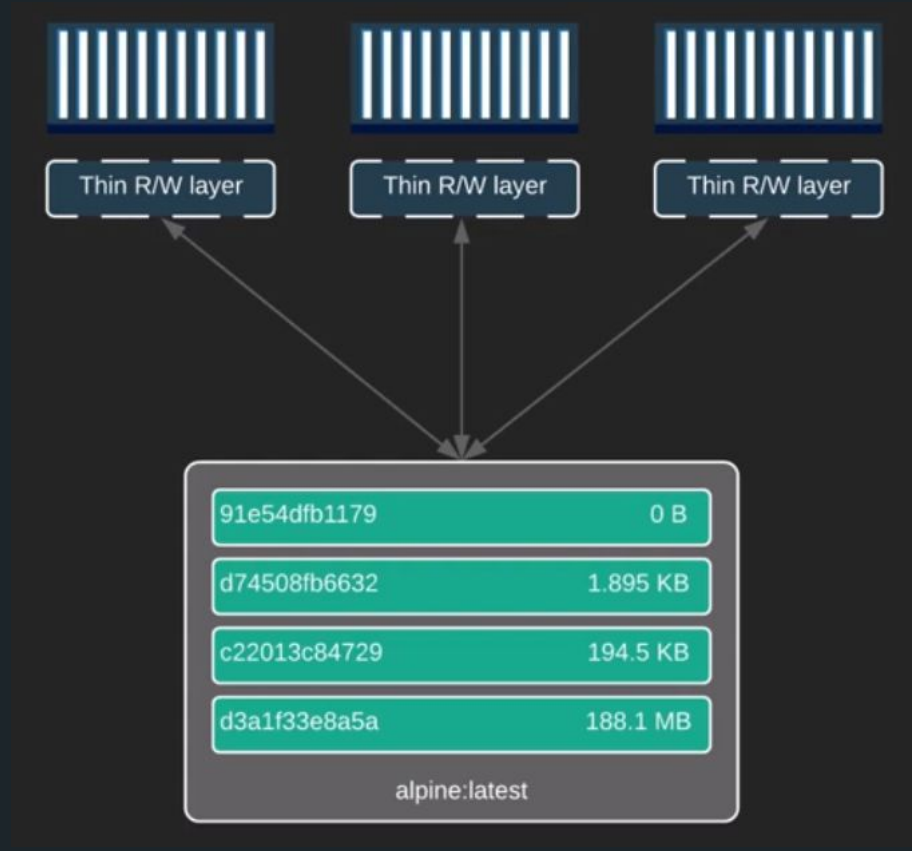
```
docker run --name container_phpmyadmin --rm -p 8081:80 phpmyadmin
```

Пример

```
[+] Building 30.0s (6/6) FINISHED
 => [internal] load build definition from Dockerfile                                                                0.0s
 => => transferring dockerfile: 149B                                                                                0.0s
 => [internal] load .dockerignore                                                                                   0.0s
 => => transferring context: 2B                                                                                     0.0s
 => [internal] load metadata for docker.io/library/phpmyadmin:latest                                                4.1s
 => [1/2] FROM docker.io/library/phpmyadmin:latest@sha256:ade62a876bec910e79ddb025cbcf54b70e7eadfcd8c12a77cb6e1c   24.2s
 => => resolve docker.io/library/phpmyadmin:latest@sha256:ade62a876bec910e79ddb025cbcf54b70e7eadfcd8c12a77cb6e1c2   0.0s
 => => sha256:8dace8be58b78f878ae28b9e5016eeb2fc5ffa0f26cf61e12f6a7f6a0735048a 4.08kB / 4.08kB                      0.0s
 => => sha256:e1bd55b3ae5fafd56f7dc3bbd88b11372bda66b4043b9075a626d392863203a5 91.60MB / 91.60MB                   15.3s
 => => sha256:47e86af584f1e1432f2e919354a76a112af1f3146d6b2342e32a496eca57f70a 226B / 226B                          0.8s
 => => sha256:1f3a70af964a1258ce68bbae397b3464a892ea2210425230d68ccdfa5e62ab90 269B / 269B                          0.7s
 => => sha256:ade62a876bec910e79ddb025cbcf54b70e7eadfcd8c12a77cb6e1c264a9e584f 1.86kB / 1.86kB                      0.0s
 => => sha256:88a6fdf429bf879515ad6d2471faa0b8a49fccdf65d0f59e2c0a3052a4f0f23e 20.30kB / 20.30kB                    0.0s
 => => sha256:0f5086159710b3ea409b4fa418d18041a81acedf8b2e62b356b16538ac746e49 19.25MB / 19.25MB                    3.8s
 => => extracting sha256:47e86af584f1e1432f2e919354a76a112af1f3146d6b2342e32a496eca57f70a                           0.0s
 => => sha256:7d9c764dc190cebd90a1661358ab11342866a47bfc7bf5898df1cf82751a6872 475B / 475B                          1.8s
 => => sha256:ec2bb7a6eead8d367aaf40f9a05e9a63913789cb1a551360a57f142680939a67 511B / 511B                          2.2s
 => => sha256:eb95760e67143cc36d681a827e056338555f9036c381aadac6d5b39d45893ced 11.11MB / 11.11MB                   12.2s
 => => sha256:f0f290ddb769f5d0b2ed2a97ac199c203c73a1f2f872f8c6dd4f1029928f0a3b 490B / 490B                          4.0s
 => => sha256:fba78b297299b8c82db43c25f9900ac214775835f28e56926150cac786ace918 10.77MB / 10.77MB                    6.3s
 => => sha256:3e42caf34750f5a8ec95329729ccaa06968de89807df97db580c731f17e22e5c 2.46kB / 2.46kB                      6.6s
 => => sha256:6abe055037f8df8124c50b80d1191e7ba81dd971df326771c715159e417b988e 245B / 245B                          6.9s
 => => sha256:052796ec377ff2e74c2711f5b783a72c1b5bc336bccc02661eacd1d1650348f3 893B / 893B                          7.1s
 => => sha256:d883d22f0a9dc67a560fc808f7d521947b0526fdbb81e347ddd35c73b126d2f3 3.31MB / 3.31MB                      8.0s
 => => sha256:f6addc6c5d35fc51fed4e0609c382387fdefe37d6802809fb69959253303d53d 545B / 545B                          8.4s
 => => sha256:74938794d4da2bd6072c27c197de503b174ac18426d24580f311c7c4f03672fb 12.43MB / 12.43MB                   11.8s
 => => sha256:38f2ecbe2869704793b41c227365589ad3c9defb441e01d4fbd5a4b173332280 1.53kB / 1.53kB                      12.1s
 => => sha256:5df91171f5ec295c48df5ec0b6cca4a647fa05ec3dc0a08e7ee3c4bbd0a5cdc1 772B / 772B                          12.5s
 => => extracting sha256:e1bd55b3ae5fafd56f7dc3bbd88b11372bda66b4043b9075a626d392863203a5                           4.2s
 => => extracting sha256:1f3a70af964a1258ce68bbae397b3464a892ea2210425230d68ccdfa5e62ab90                           0.0s
 => => extracting sha256:0f5086159710b3ea409b4fa418d18041a81acedf8b2e62b356b16538ac746e49                           0.8s
 => => extracting sha256:7d9c764dc190cebd90a1661358ab11342866a47bfc7bf5898df1cf82751a6872                           0.0s
 => => extracting sha256:ec2bb7a6eead8d367aaf40f9a05e9a63913789cb1a551360a57f142680939a67                           0.0s
 => => extracting sha256:eb95760e67143cc36d681a827e056338555f9036c381aadac6d5b39d45893ced                           0.1s
 => => extracting sha256:f0f290ddb769f5d0b2ed2a97ac199c203c73a1f2f872f8c6dd4f1029928f0a3b                           0.0s
 => => extracting sha256:fba78b297299b8c82db43c25f9900ac214775835f28e56926150cac786ace918                           0.5s
 => => extracting sha256:3e42caf34750f5a8ec95329729ccaa06968de89807df97db580c731f17e22e5c                           0.0s
 => => extracting sha256:6abe055037f8df8124c50b80d1191e7ba81dd971df326771c715159e417b988e                           0.0s
 => => extracting sha256:052796ec377ff2e74c2711f5b783a72c1b5bc336bccc02661eacd1d1650348f3                           0.0s
 => => extracting sha256:d883d22f0a9dc67a560fc808f7d521947b0526fdbb81e347ddd35c73b126d2f3                           0.2s
 => => extracting sha256:f6addc6c5d35fc51fed4e0609c382387fdefe37d6802809fb69959253303d53d                           0.0s
 => => extracting sha256:74938794d4da2bd6072c27c197de503b174ac18426d24580f311c7c4f03672fb                           1.0s
 => => extracting sha256:38f2ecbe2869704793b41c227365589ad3c9defb441e01d4fbd5a4b173332280                           0.0s
 => => extracting sha256:5df91171f5ec295c48df5ec0b6cca4a647fa05ec3dc0a08e7ee3c4bbd0a5cdc1                           0.0s
```

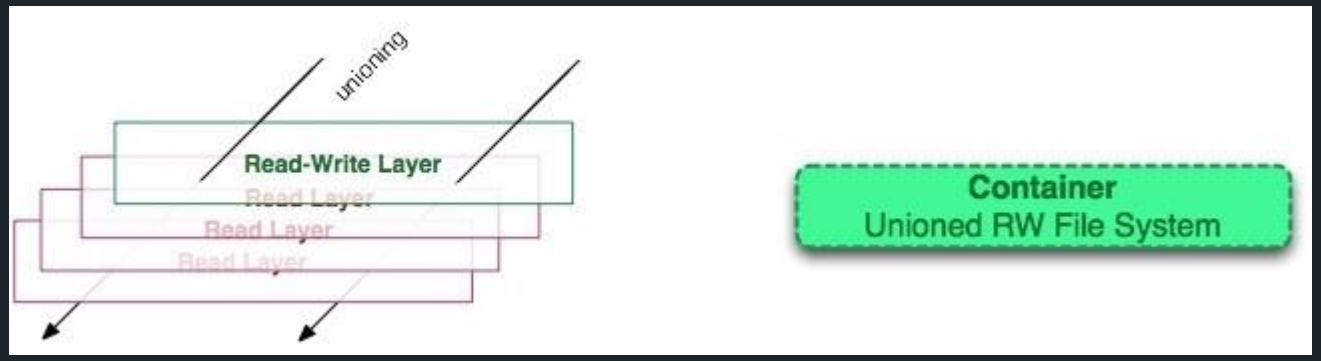
Слой контейнера



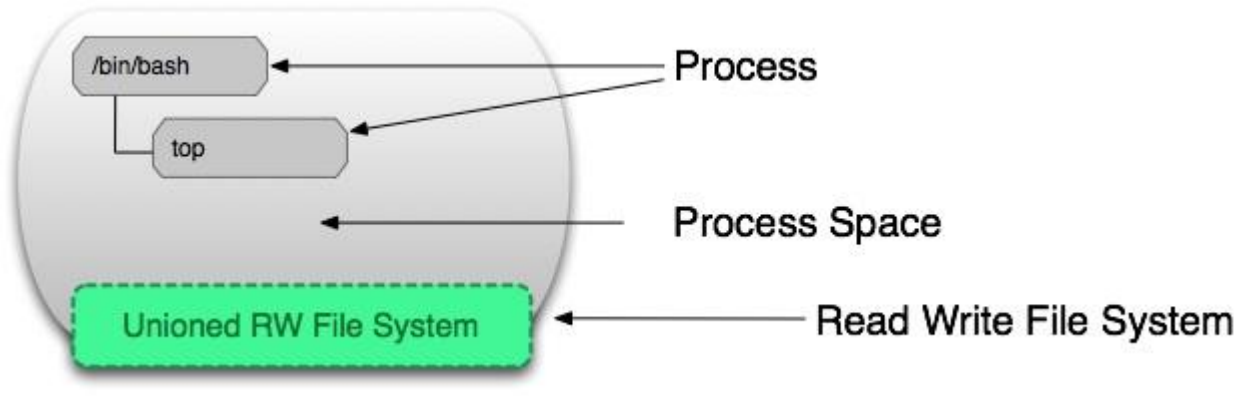


Containers - работоспособный экземпляр или инстанс образа из которого он был создан.

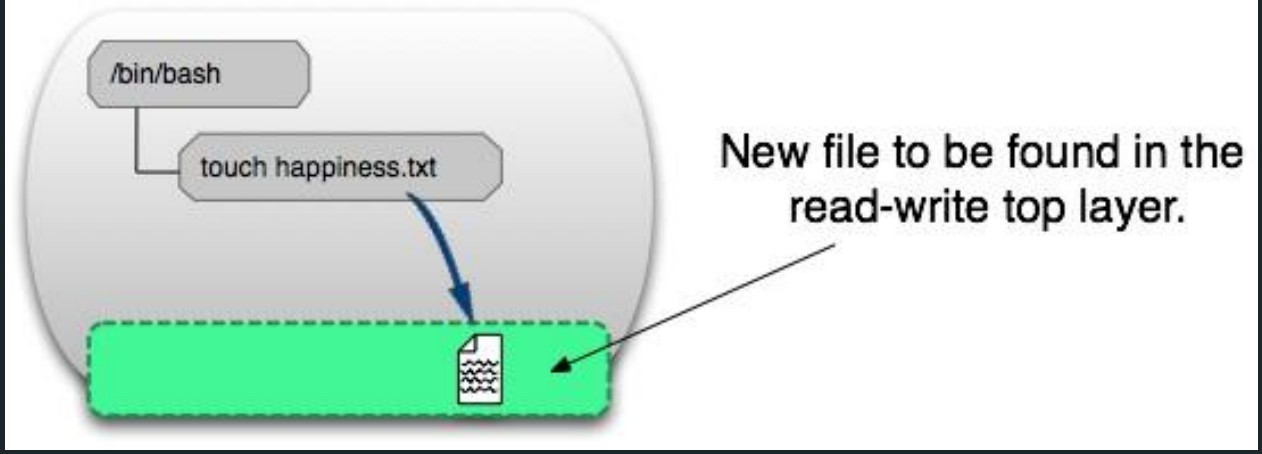
Контейнер определяет лишь слой для записи\чтения наверху образа. Но не понятно запущен он или нет.



Определение запущенного контейнера



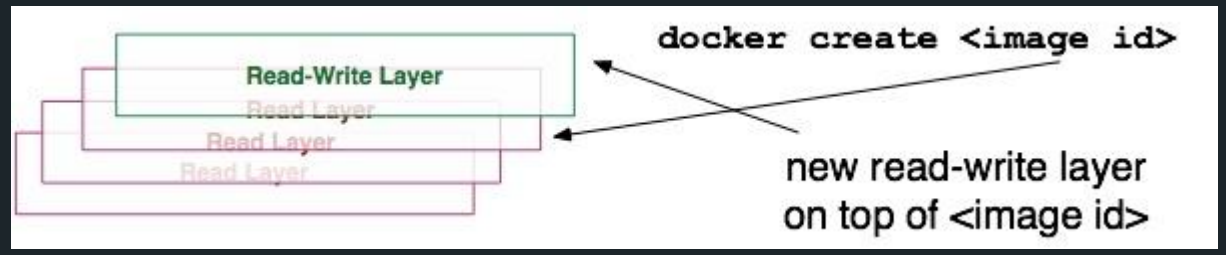
Запущенный контейнер — это «общий вид» контейнера для чтения-записи его изолированного пространства процессов.



Процессы в пространстве контейнера могут изменять, удалять или создавать файлы, которые сохраняются в верхнем слое для записи.

Команды контейнера

docker create <image-id> (docker container create) - добавляет слой для записи наверх стека слоев, найденного по <image-id> (ex. phpmyadmin). Команда не запускает контейнер.



До

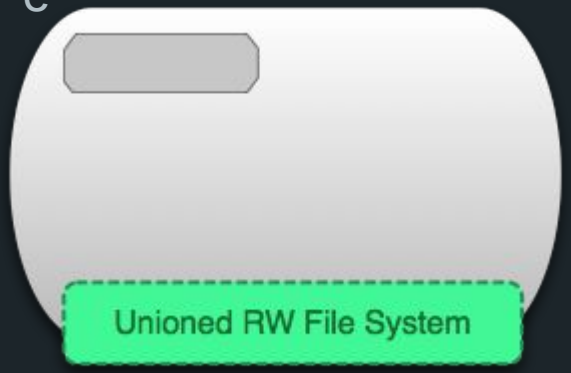
Посл
е

Команды контейнера

docker start <container-id> (docker container start) - создает пространство процессов вокруг слоев контейнера. Может быть только одно пространство процессов на один контейнер.

До

После

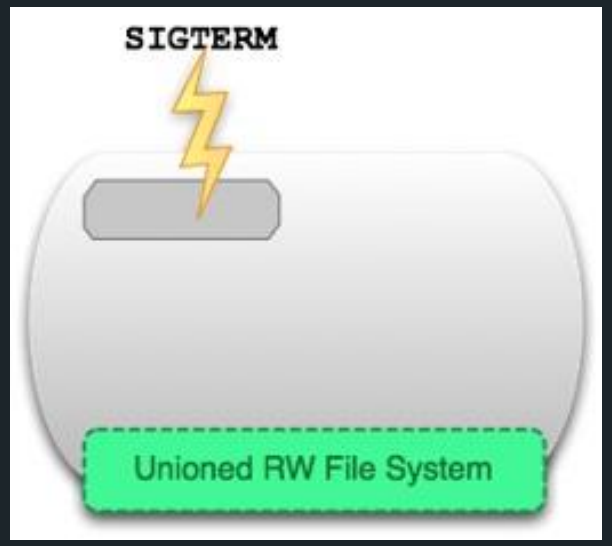


docker run === docker container run
-docker container create
-docker container start

Команды контейнера

docker stop(kill) <container-id> - Команда 'docker stop' посылает сигнал SIGTERM запущенному контейнеру, что мягко останавливает все процессы в пространстве процессов контейнера. В результате мы получаем не запущенный контейнер.
SIGTERM — сигнал, для запроса завершения процесса.

До:

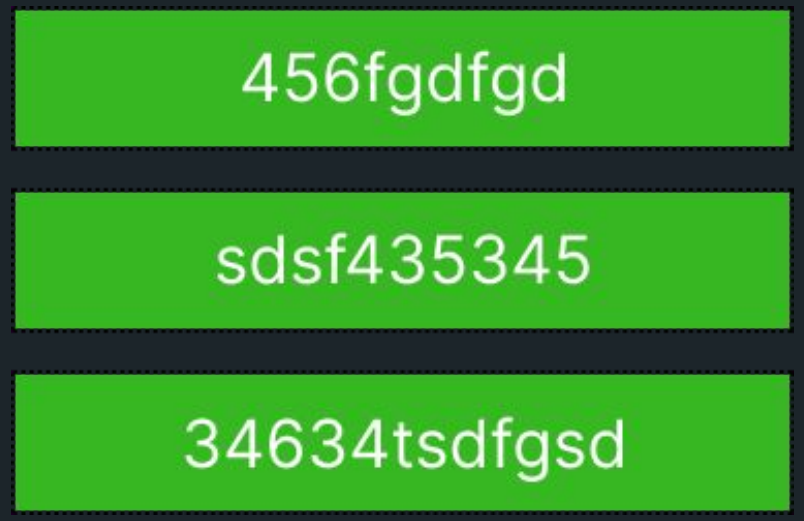


После
.

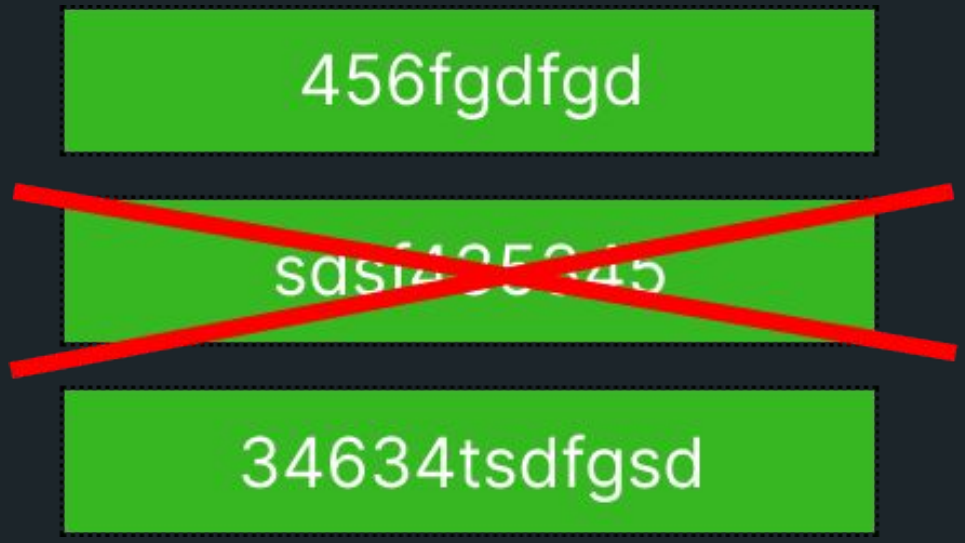
Команды контейнера

docker rm <container-id> - Команда 'docker rm' удаляет слой для записи, который определяет контейнер на хост-системе. Должна быть запущена на остановленном контейнерах. Удаляет файлы.

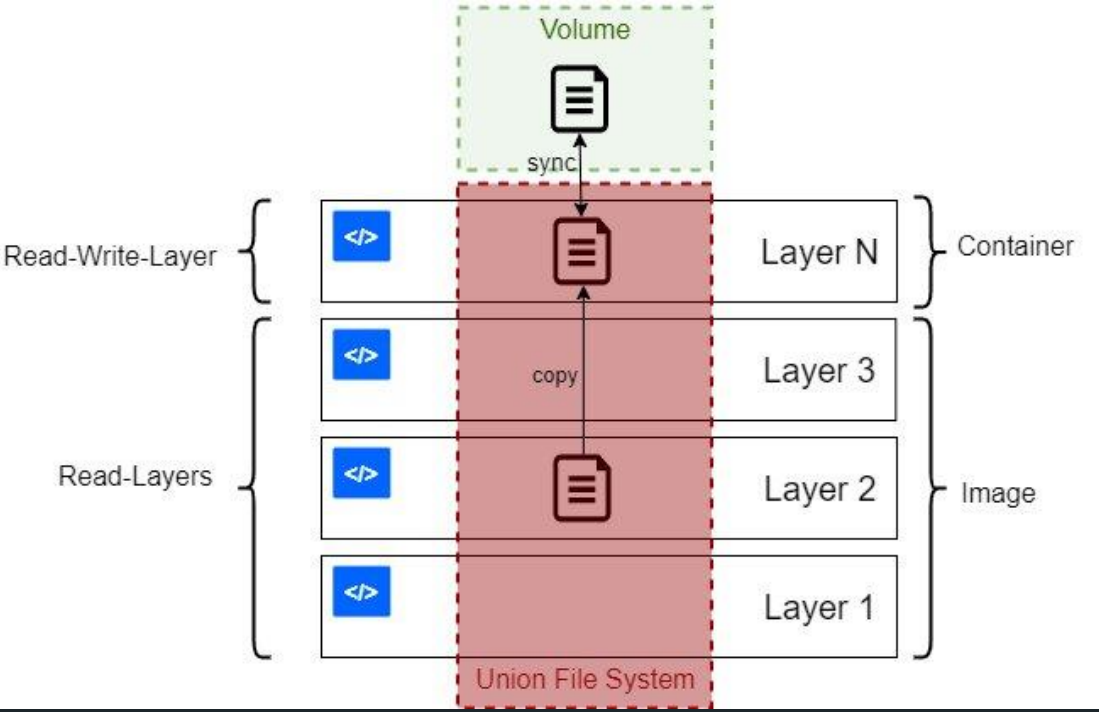
До: docker rm sdsf435345



После:



Docker volumes



В Docker есть несколько способов хранения данных. Наиболее распространенные:
-тома хранения данных (docker volumes),
-монтирование каталогов с хоста (bind mount)

Тома — рекомендуемый разработчиками Docker способ хранения данных.

В Linux тома находятся по умолчанию в /var/lib/docker/volumes/.

Window - \\wsl$\docker-desktop-data\version-pack-data\community\docker\volumes

Том (docker volumes)

Один том может быть примонтирован одновременно в несколько контейнеров. Когда никто не использует том, он не удаляется, а продолжает существовать.

Для чего стоит использовать тома в Docker:

- шаринг данных между несколькими запущенными контейнерами,
- решение проблемы привязки к ОС хоста,
- удалённое хранение данных,
- бэкап или миграция данных на другой хост с Docker (для этого надо остановить все контейнеры и скопировать содержимое из каталога тома в нужное место).

```
docker volume create my-storage
docker run -v my-storage:/var/lib/mysql mysql:8
(or --volume)
```

Монтирование каталога с хоста (bind mount)

Используется, когда нужно пробросить в контейнер конфигурационные файлы с хоста. Другое очевидное применение — в разработке. Код находится на хосте (вашем ноутбуке), но исполняется в контейнере.

```
docker run -v /user/project/database:/var/lib/mysql mysql:8
```

Команды - Volume

Вывод списка всех томов на хосте:
docker volume ls

Создание тома:
docker volume create <NAME>

Инспектирование тома:
docker volume inspect <NAME>

Удаление тома:
docker volume rm <NAME>

Удаление всех неиспользуемых томов:
docker volume prune

Когда использовать тома, а когда монтирование с хоста

Volume	**Bind mount**
Просто расшарить данные между контейнерами.	Пробросить конфигурацию с хоста в контейнер.
У хоста нет нужной структуры каталогов.	Расшарить исходники и/или уже собранные приложения.
Данные лучше хранить не локально (а в облаке, например).	Есть стабильная структура каталогов и файлов, которую нужно расшарить между контейнерами.

Docker network

Подключает контейнер к сети. Подключить контейнер можно по имени или по ID. После подключения контейнер может взаимодействовать с другими контейнерами в той же сети.

Список сетей
docker network ls

Создание сети
docker network create <NAME>

Инспектирование сети
docker network inspect <NAME>

Удаление сети
docker network rm <NAME>

Примеры команд

Создание томов и сети

```
docker volume create database_storage
docker volume create mongodb_storage
docker network create network_1
```

Создание контейнера phpmyadmin

```
docker run -d --rm -p 8081:80 -e PMA_HOST=test-mysql --name test-phpmyadmin --net network_1
phpmyadmin:latest
```

Создание контейнера mysql

```
docker run -d --rm --name test-mysql -p 3307:3306 -e MYSQL_ROOT_PASSWORD=secret -e
MYSQL_DATABASE=testdb -e MYSQL_USER=user -e MYSQL_PASSWORD=dbsecret -v
database_storage:/var/lib/mysql --net network_1 mysql:5.7
```

Создание контейнера mondgodb

```
docker run –d --rm --name test-mongodb -p 27017:27017 -e MONGO_INITDB_ROOT_USERNAME=mongo_user
-e MONGO_INITDB_ROOT_PASSWORD=mongo_pass -v mongodb_storage:/data/db --net network_1 mongo
```

Docker run

docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]

[OPTIONS]:
--rm - автоматически удалять контейнер при его выключении
--name, задать имя контейнера
--volume , -v – привязать том
--detach,-d - запустить контейнер в фоновом режиме и распечать идентификатор контейнера
--env , -e – установить переменные окружения
--publish , -p – открыть порт контейнера на хосте
--tty,-t - Выделить псевдо-TTY, устройство которое имеет функции физического терминала
--interactive,-i - Держать STDIN (стандартный поток ввода) открытым, даже если он не подключен
--net - Подключить контейнер к сети

IMAGE[:TAG|@DIGEST]:
mysql , mysql:8 или mysql:latest или getmeili/meilisearch или ubuntu@sha256:82becede498899e,

[COMMAND]: и [ARG...]
Это команды, инструкции который будет выполнять контейнер, не обязательный параметр
Т.к команды могут быть встроены уже в Dockerfile

Пример Dockerfile

Часть файла Dockerfile

```
……….
VOLUME /var/lib/mysql

COPY docker-entrypoint.sh /usr/local/bin/
RUN ln -s usr/local/bin/docker-entrypoint.sh /entrypoint
ENTRYPOINT ["docker-entrypoint.sh"]

EXPOSE 3306 33060
CMD ["mysqld"]
```

Инструкция Dockerfile

Инструкция WORKDIR устанавливает рабочий каталог для любых RUN, CMD, ENTRYPOINT, COPY и ADD инструкции

Пример:

```
FROM php:7.4-cli
WORKDIR app
COPY my_application.php /app/my_application.php
CMD ["php", "my_application.php"]
```

Инструкция Dockerfile

Инструкция RUN выполнит любые команды в новом слое поверх текущего образа и фиксирует результаты.

Инструкция COPY - копирует новые файлы или каталоги <src> и добавляет их в файловую систему контейнера по пути <dest>

Инструкция ADD - копирует новые файлы, каталоги или URL-адреса удаленных файлов <src> и добавляет их в файловую систему образа по пути <dest>.

Инструкция EXPOSE информирует Docker о том, что контейнер прослушивает указанные сетевые порты во время выполнения. Вы можете указать, прослушивает ли порт TCP или UDP, и по умолчанию используется TCP, если протокол не указан.

Инструкцию VOLUME следует использовать для предоставления доступа к любой области хранения базы данных, хранилищу конфигурации или файлам/папкам, созданным вашим докер контейнером.

CMD и Entrypoint

CMD
Инструкция CMD позволяет вам установить команду по умолчанию, которая будет выполняться только тогда, когда вы запускаете контейнер без указания команды.
Если контейнер Docker запускается с командой, команда по умолчанию будет игнорироваться.
Если Dockerfile содержит более одной инструкции CMD, все инструкции CMD, кроме последней, игнорируются.
CMD — это инструкция, которую лучше всего использовать, если вам нужна команда по умолчанию, которую пользователи могут легко переопределить.

ENTRYPOINT
Инструкция ENTRYPOINT позволяет настроить контейнер, который будет работать как исполняемый файл. Он похож на CMD, потому что также позволяет указать команду с параметрами. Разница заключается в том, что команда ENTRYPOINT и параметры не игнорируются, когда контейнер Docker запускается с параметрами командной строки.

CMD и Entrypoint

Dockerfile:
```
FROM alpine
ENTRYPOINT ["ping"]
CMD ["www.google.com"]
```

```
docker build -t ping-service .
docker run --rm ping-service

docker run --rm ping-service ya.ru
```

Dockerfile:
```
FROM alpine
CMD ["ping", "www.google.com"]
```

```
docker build -t ping-service2 .
docker run --rm ping-service2

docker run --rm ping-service2 ping ya.ru
```

CMD и Entrypoint

CMD или ENTRYPOINT установлены в родительском образе, ни один из них не установлен в дочернем образе - докер сохранит родительский.

CMD и ENTRYPOINT устанавливаются в родительском образе, как и в дочернем — docker переопределит оба

ENTRYPOINT установлен в родительском образе, CMD представлен в дочернем образе - докер сохранит оба

CMD установлен в родительском образе, ENTRYPOINT представлен в дочернем образе - докер сохранит ENTRYPOINT, но CMD будет сброшен

Команды

Список запущенных контейнеров
docker ps
docker ps -a (все контейнеры в том числе и остановленные)

Список запущенных контейнеров
docker ps
docker ps -a (все контейнеры в том числе и остановленные)

Создать контейнер и присоединиться к нему:
docker run -it busybox

Создать контейнер и запустить его в фоне:
docker run -d nginx

Создать контейнер с именем и запустить его в фоне:
docker run -d -name container_alpine alpine

(ps. docker run === docker container run)

Команды

Выполнить команду в контейнере:
docker container exec -it container_alpine ping ya.ru

Отобразить информацию, собранную запущенным контейнером (логирование)
docker container logs container_alpine

Инспектирование процессов в контейнере
docker container top container_alpine

Отображение статистики использования ресурсов контейнеров в реальном времени.
docker stats container_alpine

Команды

Остановка контейнера
docker stop container_alpine
docker kill container_alpine

Удаление контейнера
docker rm container_alpine

Остановить все Docker контейнеры.
docker stop $(docker ps -a -q)
docker kill $(docker ps -q)

Удалить все Docker контейнеры
docker rm $(docker ps -a -q)

Удаление всех неиспользуемых контейнеров, сетей, образов и томов
docker system prune

Docker Compose

Инструмент для описание и запусков приложения которые состоят из нескольких приложений

Docker Compose утилита позволяет запускать проект состоящий из нескольких контейнеров:

- Позволяет запустить и настроить многоконтейнерные приложения
- Все описывается в docker-compose.yml
- Создает свою сеть проекту
- Дает возможность обращаться контейнерам др к др по именам

Docker Compose

docker-compose build собрать проект

docker-compose up -d запустить проект (запускать в режиме демона)

docker-compose down остановить проект

docker-compose logs -f [service name] посмотреть логи сервиса

docker-compose ps вывести список контейнеров

docker-compose exec [service name] [command] выполнить команду

docker-compose images список образов

Docker Compose

RESTART:

on-failure:
on-failure[:max-retries]
Перезапустите контейнер, если он завершает работу из-за ошибки, которая проявляется как ненулевой код выхода. При необходимости ограничьте количество попыток демона Docker перезапустить контейнер с помощью :max-retries параметра.

always
контейнер будет перезапускаться всегда, даже если был остановлен
Всегда перезапускайте контейнер, если он останавливается. Если он остановлен вручную, он перезапускается только при перезапуске демона Docker или при ручном перезапуске самого контейнера.

unless-stopped
Аналогичен always, за исключением того, что когда контейнер останавливается (вручную или иным образом), он не перезапускается даже после перезапуска демона Docker.

Для ознакомления

https://www.youtube.com/watch?v=QF4ZF857m44

https://12factor.net/ru/

https://habr.com/ru/company/southbridge/blog/530226/

https://habr.com/ru/company/southbridge/blog/329138/

https://habr.com/ru/post/349802/