

# ML Advanced

Практическое занятие по работе в  
проде: деплой докера в Yandex Cloud



**Проверить, идет ли запись**

# **Меня хорошо видно && слышно?**

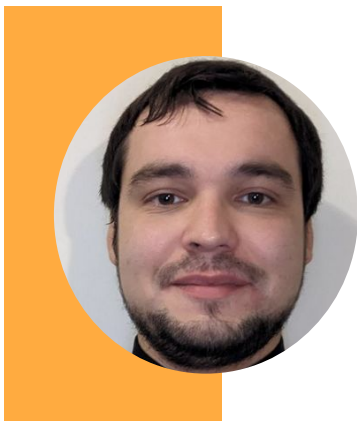


Ставим "+", если все хорошо  
"-", если есть проблемы



Тема вебинара

# Практическое занятие по работе в проде: деплой докера в Yandex Cloud



**Гайнуллин Дмитрий**

Machine Learning Engineer в AIC

- Разработка моделей для распознавания речи
- Прогнозирование ключевых метрик
- Развертывание моделей в продакшене



# Правила вебинара



Активно  
участвуем



Off-topic обсуждаем  
в учебной группе



Задаем вопрос  
в чат или голосом



Вопросы вижу в чате,  
могу ответить не сразу

## Условные обозначения



Индивидуально



Время, необходимое  
на активность



Пишем в чат



Говорим голосом

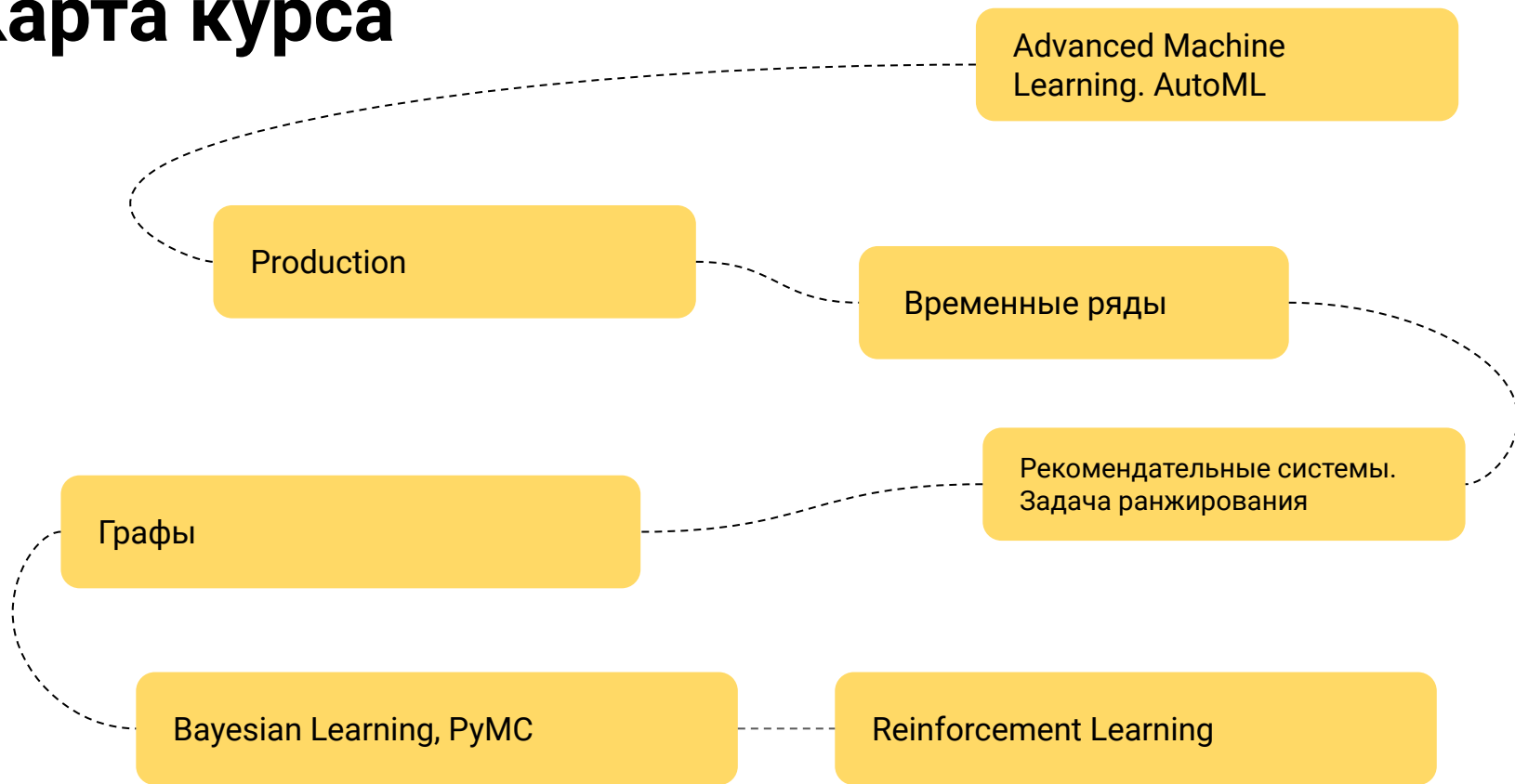


Документ



Ответьте себе или  
задайте вопрос

# Карта курса



# Цели вебинара

1

Просмотреть сервисы  
Yandex.Cloud

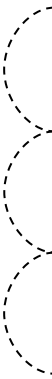
2

Изучить основы виртуализации

3

Практиковаться созданием докер  
образца в Yandex.Cloud

# Маршрут вебинара



Yandex Cloud - Обзор платформы

Виртуализация

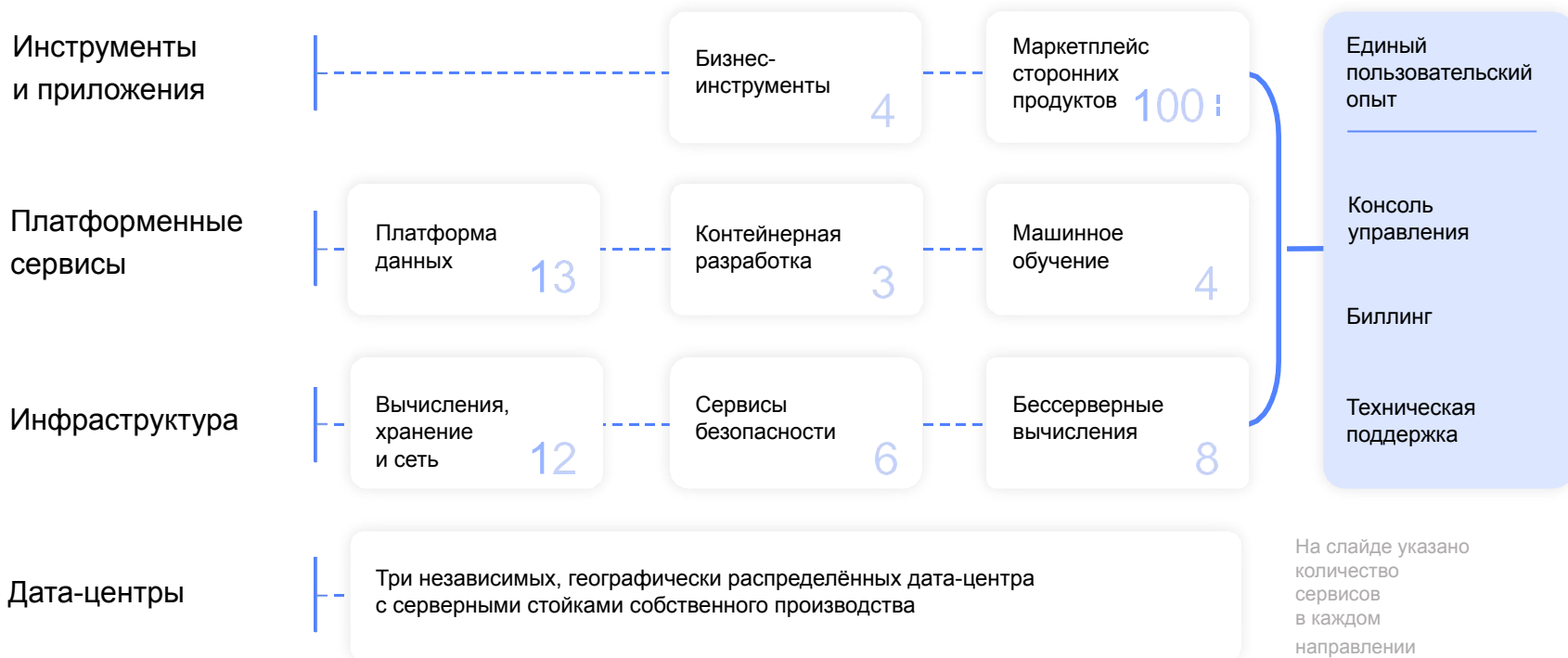
Практика

Рефлексия

# Платформа Yandex Cloud



# Платформа Yandex Cloud — единый хаб новых технологий



# Сценарии использования

Сайт в облаке Интернет-магазин



Хранение и обработка  
персональных  
данных



1С в облаке



Сервисы Microsoft  
в облаке



Автоматизация  
колл-центров



Рекомендательная  
система для ритейла  
и e-commerce



Корпоративное  
хранилище  
данных



Бизнес-аналитика  
и визуализация  
данных



Serverless



Чат-боты  
на Serverless



Distributed Cloud



Миграция в облако



Security Solution  
Library



Solution Library  
for AWS



# Вопросы?



Ставим “+”,  
если вопросы есть

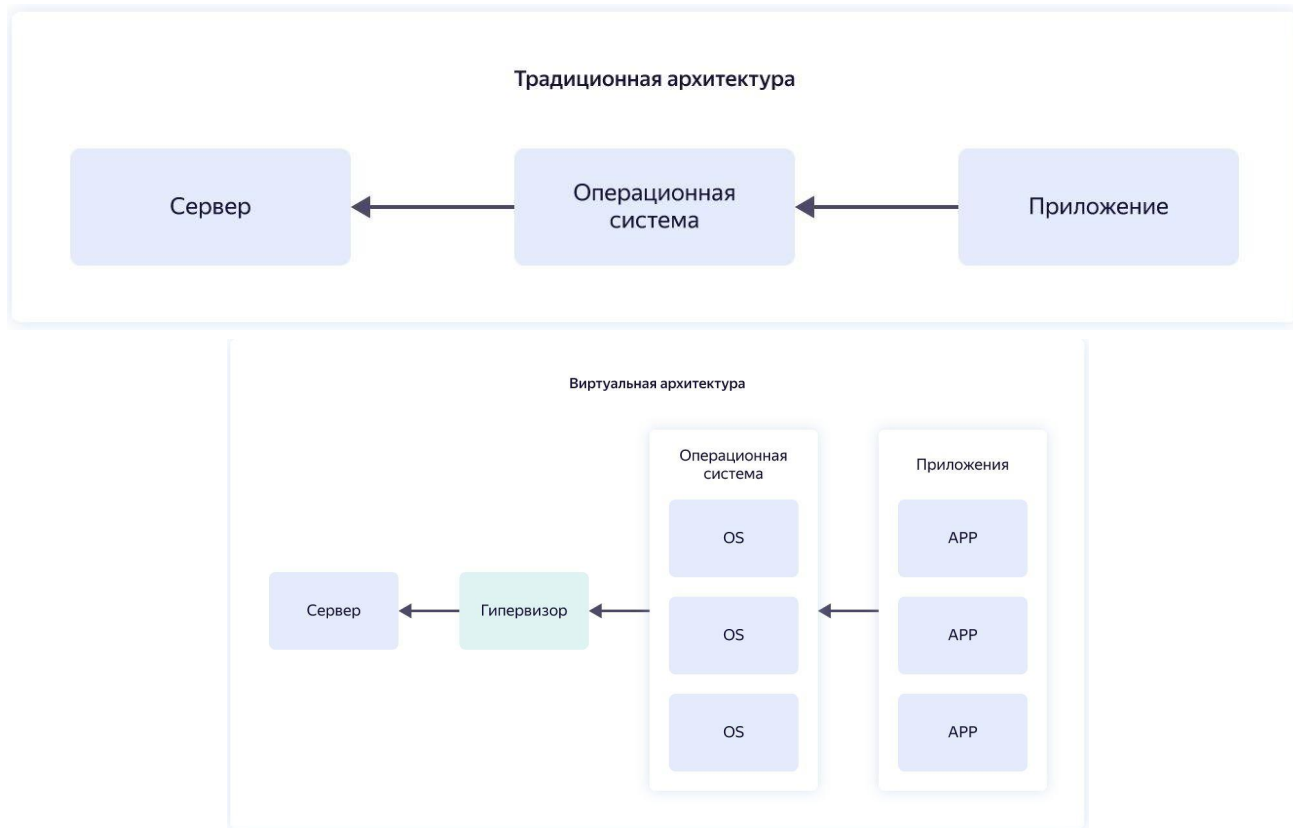


Ставим “-”,  
если вопросов нет

# Виртуализация

# Виртуальная машина

Упрощенно, ВМ — это программа, которая имитирует работу другой программы («компьютер внутри компьютера» или «машина внутри машины»). Поскольку имитируемая машина вне среды хост-платформы не существует, она получила название «виртуальной».



# Виртуализация

Виртуализация на уровне железа



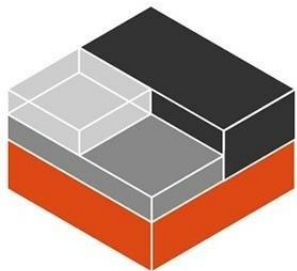
# Виртуализация

Виртуализация на уровне ядра  
(на уровне операционной системы)



OpenVZ

Systemd-nspawn



LXC

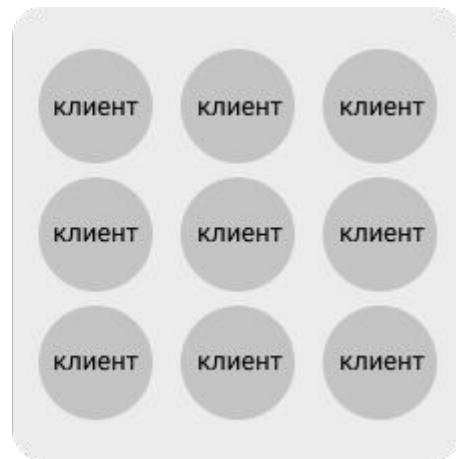
# Виртуализация

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

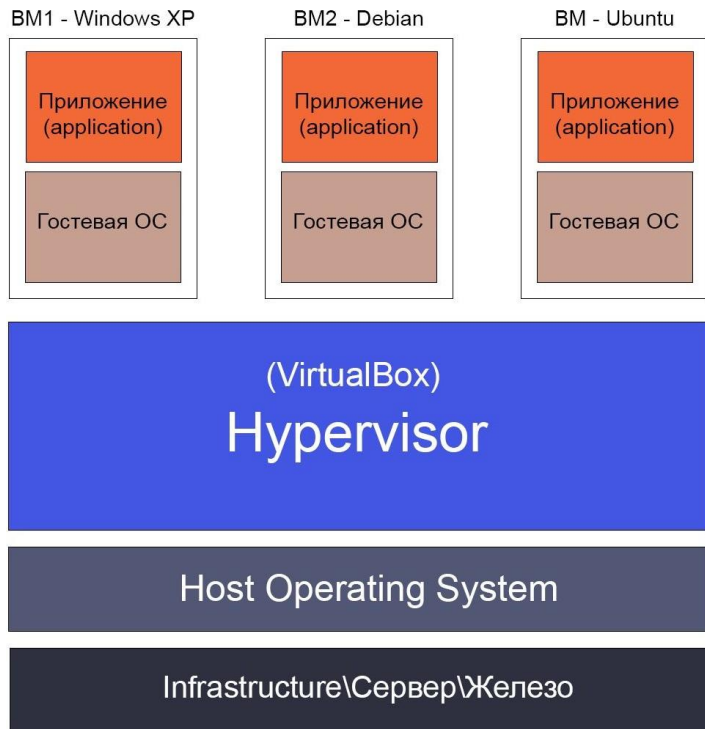
1ый случай % исп. 5%  
2ой случай % исп. 90%





# Виртуализация на уровне железа

## Виртуальные машины



Гипервизор - или монитор виртуальных машин

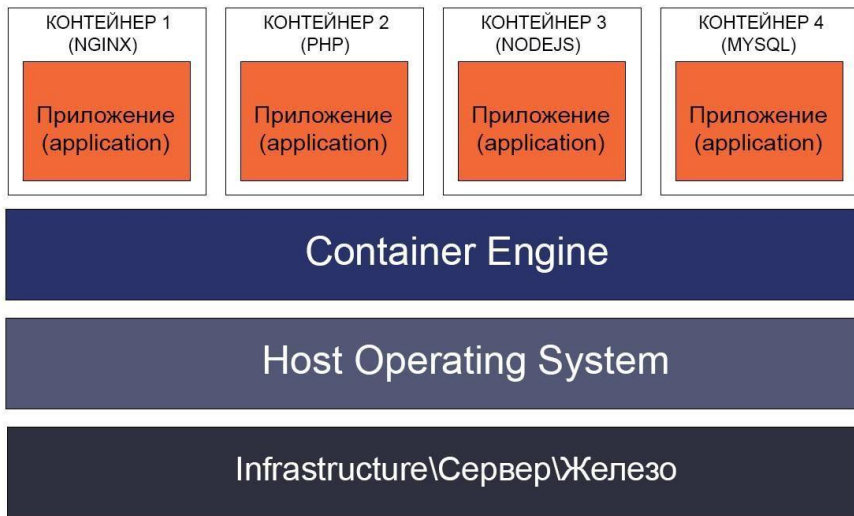
В VM нельзя иметь ресурсов больше чем на хостовой системе

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

Выделяем ресурсы

# Виртуализация на уровне ядра

## Контейнеры



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:

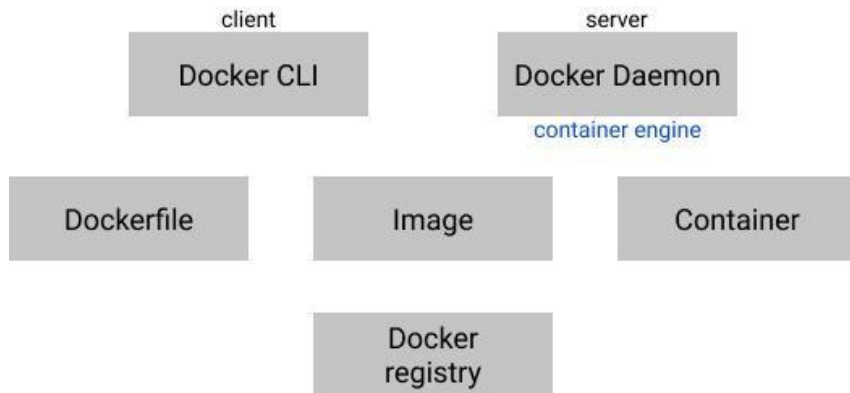
- Использует ядро хостовой системы
- В контейнере только Linux (не давно и Windows)
- Контейнер для изоляции плохо - т.к можно выбраться на хостовую машину

# Рабочее окружение

Для разработки бекенда часто нужны 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
- ....

# 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

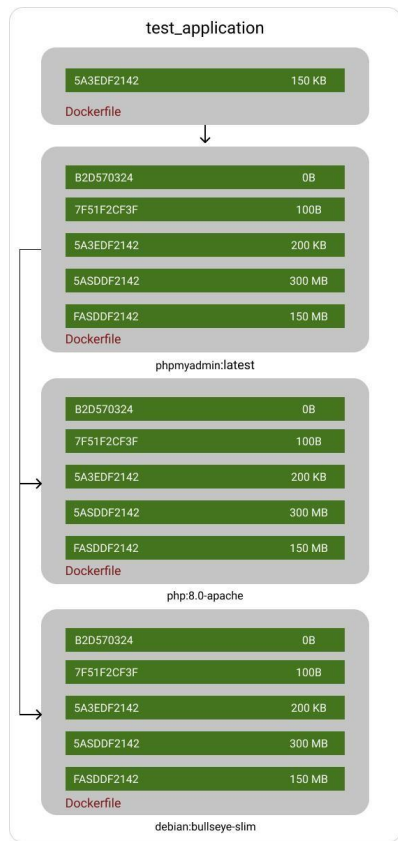
- Упаковка нашего контейнера
- Из них запускается контейнеры
- Хранятся в докер реестрах (registry - docker hub)
- Имеют hash (sha256), имя и tag
- Имеет слоенную архитектуру
- Создаются по инструкциям dockerfile

# Объекты первого класса докера

Containers - работоспособный экземпляр или инстанс образа из которого он был создан. Когда создается контейнер из образа, оба они становятся зависимыми друг от друга нельзя удалить образ, пока существуют 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 друг над другом

Вместе они представляют единый объект

- Каждый слой представляет собой отдельную инструкцию из докерафайла образа
- Все слои доступны 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" ]
```

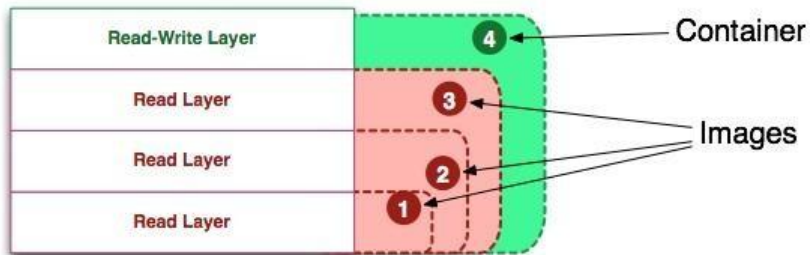
```
CMD ["apache2-foreground"]
```

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

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

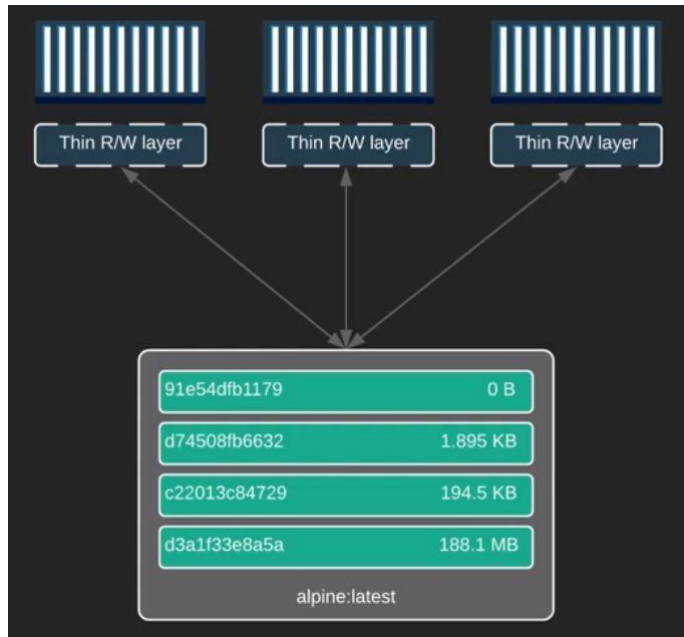
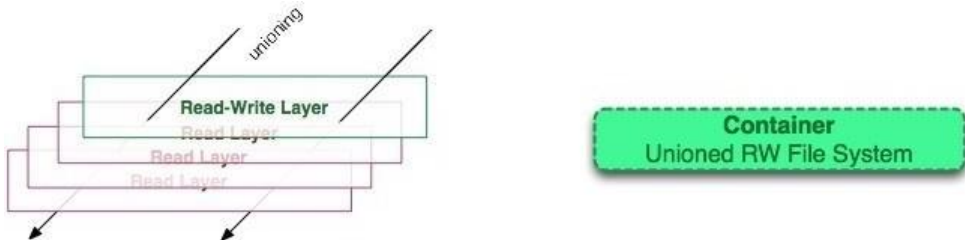
```
docker run --name container_phpmyadmin --rm -p 8081:80 phpmyadmin
```

# Слой контейнера

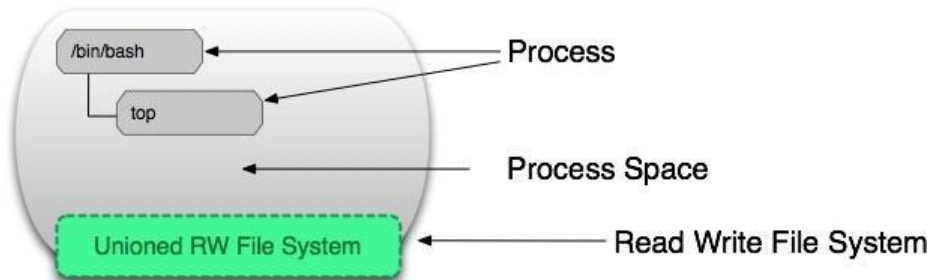


Containers - работоспособный экземпляр или инстанс образа из которого он был создан.

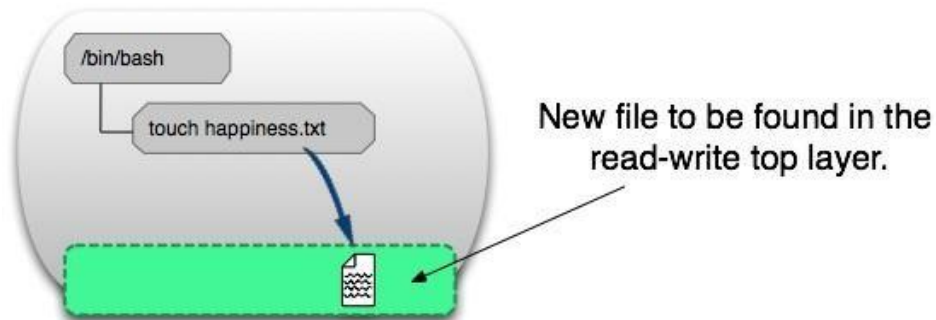
Контейнер определяет лишь слой для записи\чтения наверху образа. Но не понятно запущен он или нет.



# Определение запущенного контейнера



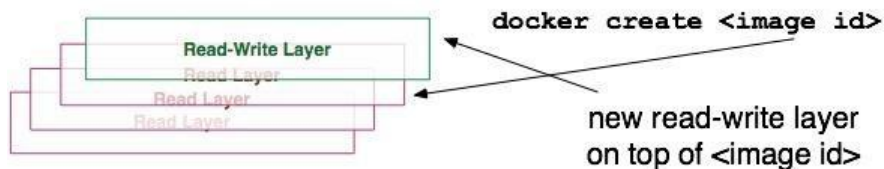
Запущенный контейнер — это «общий вид» контейнера для чтения-записи его изолированного пространства процессов.



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.



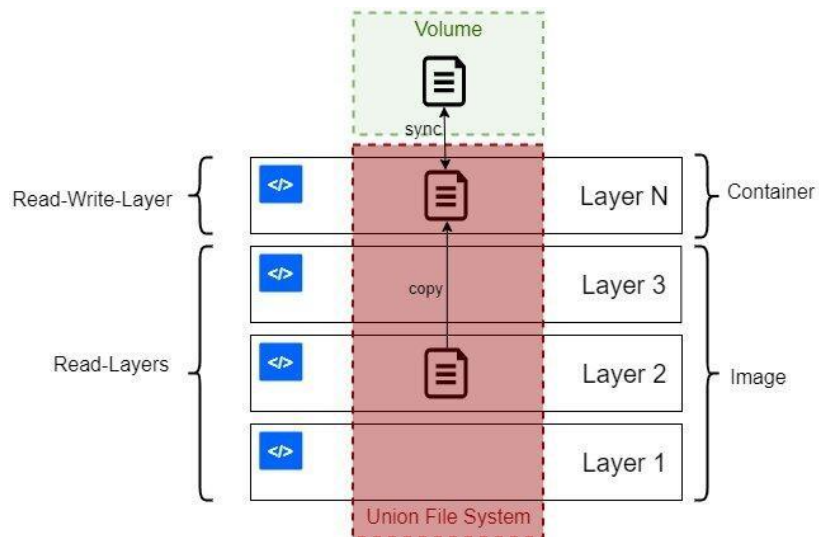
До



После



# Docker volumes



В Docker есть несколько способов хранения данных. Наиболее распространенные:

- тома хранения данных (docker volumes),
- монтирование каталогов с хоста (bind mount)

Томы — рекомендуемый разработчиками Docker способ хранения данных.

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

Windows -

`\\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 выполняется 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

<u>Volume</u>	<u>Bind mount</u>
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.

# 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"](http://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"](http://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)
```

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 описания и запуска приложений, 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 друг к другу 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/>

# Вопросы?



Ставим "+",  
если вопросы есть



Ставим "-",  
если вопросов нет



# Практика



# Рефлексия

# Цели вебинара

## Проверка достижения целей

1. Просмотреть сервисы Yandex.Cloud
2. Изучить основы виртуализации
3. Практиковаться созданием докер образа в Yandex.Cloud

# Рефлексия



С какими впечатлениями уходите с вебинара?



Как будете применять на практике то, что узнали на вебинаре?

**Заполните, пожалуйста,  
опрос о занятии  
по ссылке в чате**