Промышленное машинное обучение на Spark

Лекция 1: Docker

13 января 2024 г.

Содержание курса

1. > *Оборачиваем модель в сервис. Docker. Kubernetes.*
2. Оборачиваем модель в сервис. Requests. Flask. REST API.
3. Погружение в среду Spark. Распределённые вычисления. MapReduce.
4. Погружение в среду Spark. Распределённое хранение данных. Форматы.
5. ML блок. Генерация признаков. Spark feature engineering.
6. ML блок. Распределённое обучение моделей. Spark ML
7. Рекомендательные системы в Spark. Spark NLP.
8. Обработка потоковых данных. Spark Streaming.

Правила игры

В курсе будет 4 домашние работы:

- За каждую домашнюю работу можно набрать 20 баллов;
- На выполнение каждого домашнего задания даётся 2 недели;
- Если сдать после 2х недель дз принимается со штрафом;
- Если сдать после 4х недель дз не принимается;

Квизы перед занятиями по предыдущему материалу:

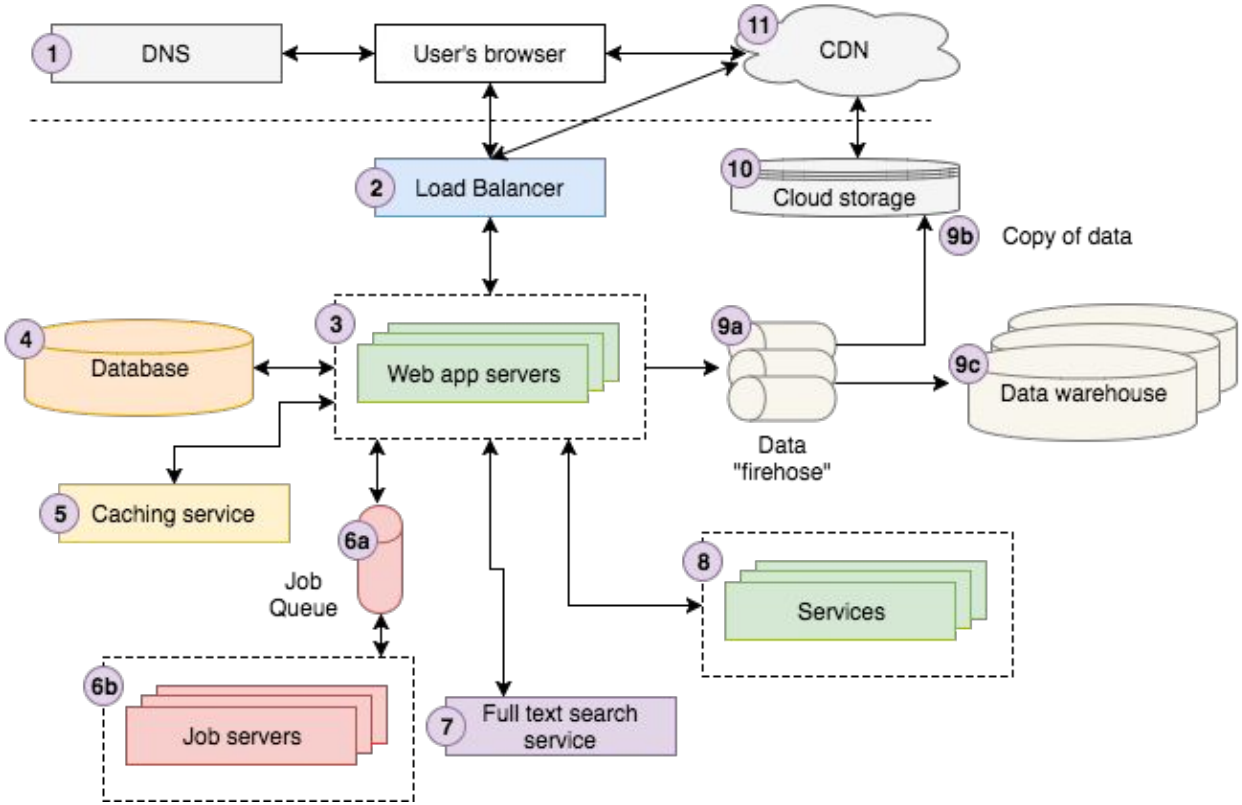
- Присылаются за 2-3 часа до начала занятий;
- Принимаются до момента их разбора в конце занятия;
- Оцениваются по 3х бальной шкале: хорошо, средне, плохо;

Экзамен, кому не хватило баллов до зачёта автоматом.
Зачёт автоматом >= 60% от возможного максимума.

Сегодня поговорим о …

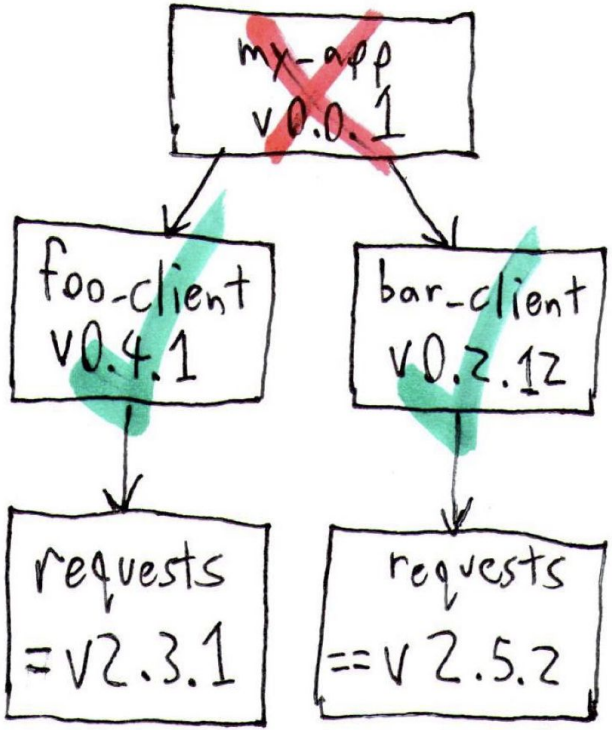
1. Какие задачи возникают при разработке сервисов
2. Зачем нужна контейнеризация приложений и какие задачи решает
3. Что такое Docker и из каких сущностей он состоит
4. Какое ещё существует ПО помимо Docker
5. Как устроены современные сервисы

Типичный портрет современного сервиса



Проблема 1. Dependency hell

Разные компоненты разрабатываемой системы могут требовать разные версии сторонних библиотек



Проблема 2. Non-safety execution

Поломка или ошибки
одного компонента системы
может привести к отказу
всего сервиса

Проблема 3. Сложный процесс внедрения

Не всегда коллеги из соседней команды понимают как запустить ваш сервис в их прогрммной инфраструктуре

```
> Firstly download libraries lib1 & lib2
> Prepare data file in csv format
> Rename prepared filed to dataset.csv
> Next copy main.py to src folder
> Set params at config file
....
> Now run you can run service!!!
```

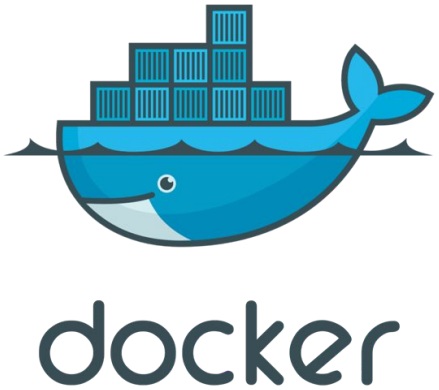
Контейнеризация

Принципы контейнеризации

1. Изоляция объектов системы друг от друга
2. Доставка объекта со всеми необходимыми компонентами
3. Универсальный интерфейс работы с контейнером
4. Независимость от способа доставки

Технология Docker

Docker - набор программных инструментов с открытым исходным кодом, решающих задачи контейнеризации, автоматического внедрения, запуска и эксплуатации приложений.

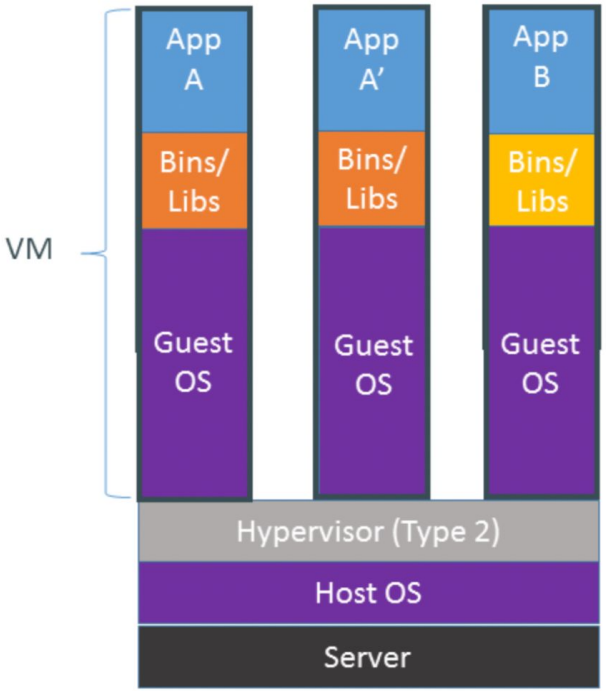


Преимущества docker

- Платформонезависимость
- Абстрагирование частей приложения друг от друга
- Универсальный интерфейс работы с приложением
- Упрощения построения отказоустойчивых систем
- Упрощение масштабирования системы
- Автоматизация рутинных задач: тестирование, CI/CD

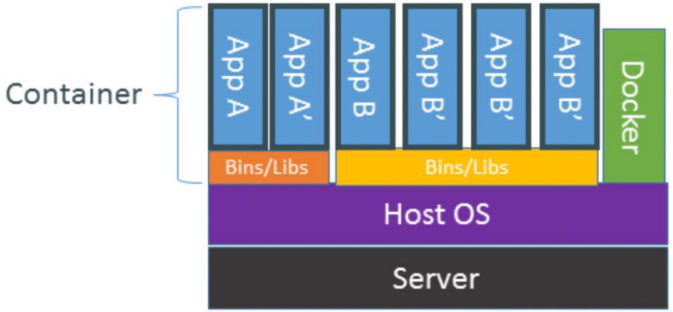
Virtual machines & docker

Зачем нужно было изобретать docker, когда уже существовали виртуальные машины?



Containers are isolated, but share OS and, where appropriate, bins/libraries

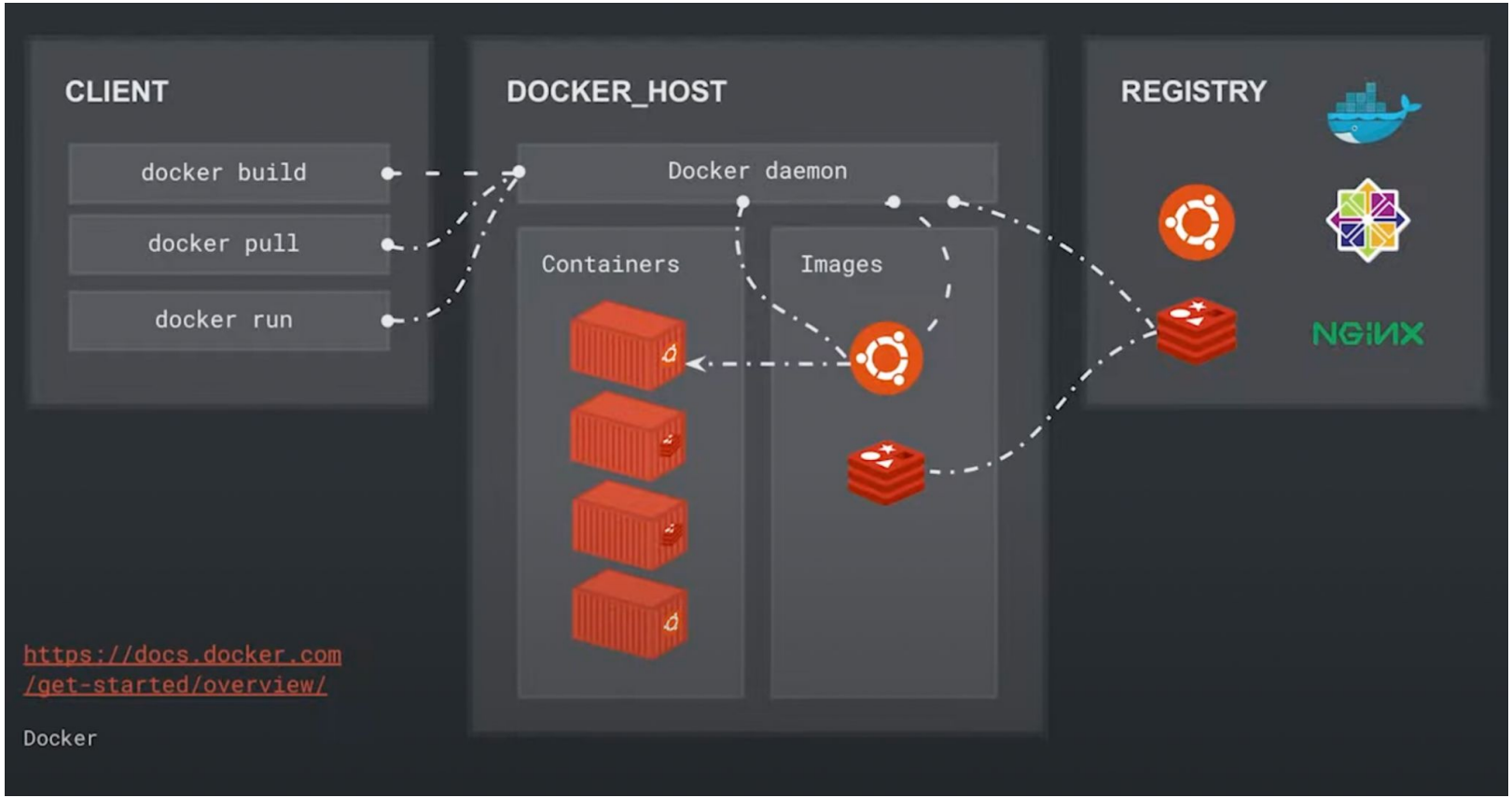
...result is significantly faster deployment, much less overhead, easier migration, faster restart



Основные отличия docker от VM

- Docker не запускает отдельное ядро для каждого процесса по отдельности
- Контейнеры значительно быстрее запускаются
- VM требует больше ресурсов для запуска
- В VM сложно контролировать изменения, версионирование
- Контейнеры могут иметь разделяемые ресурсы между собой (бинарники, библиотеки)
- При изменении контейнера будет сохраняться только разница между исходником и новой версией
- VM имеют более надёжную изоляцию между собой
- VM чуть проще в использовании

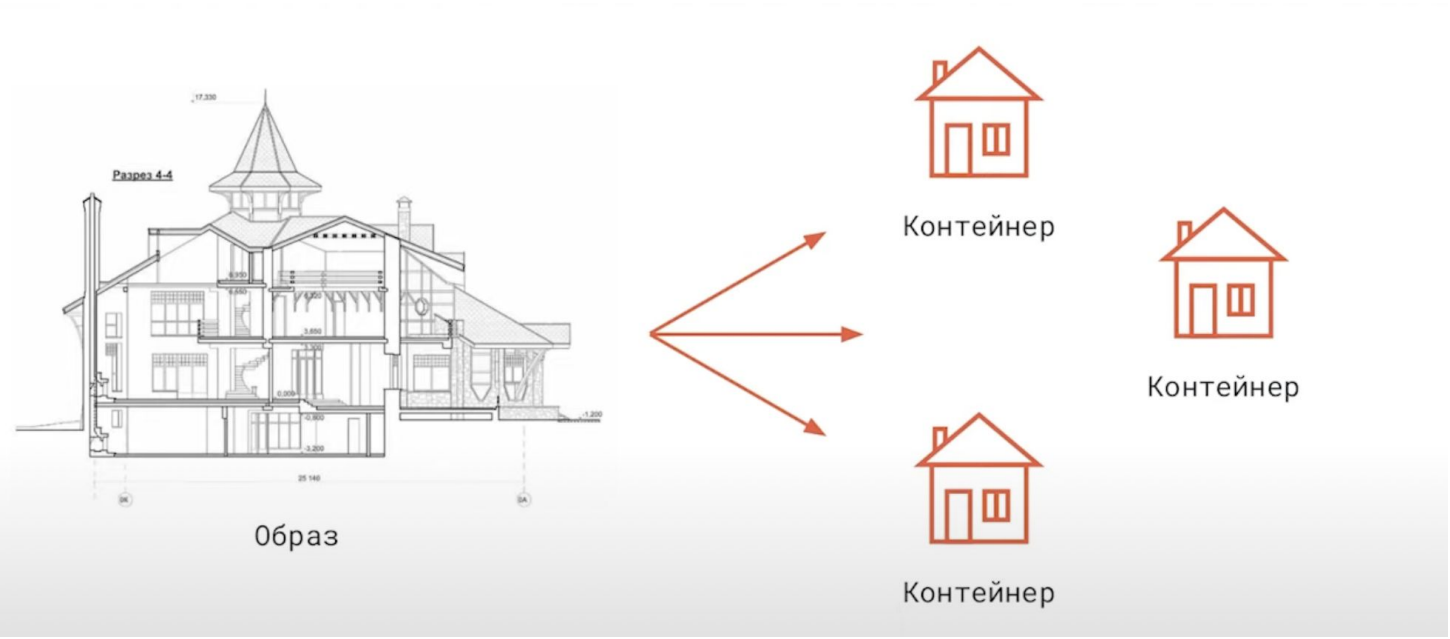
Основные компоненты технологии



Функции каждого компонента

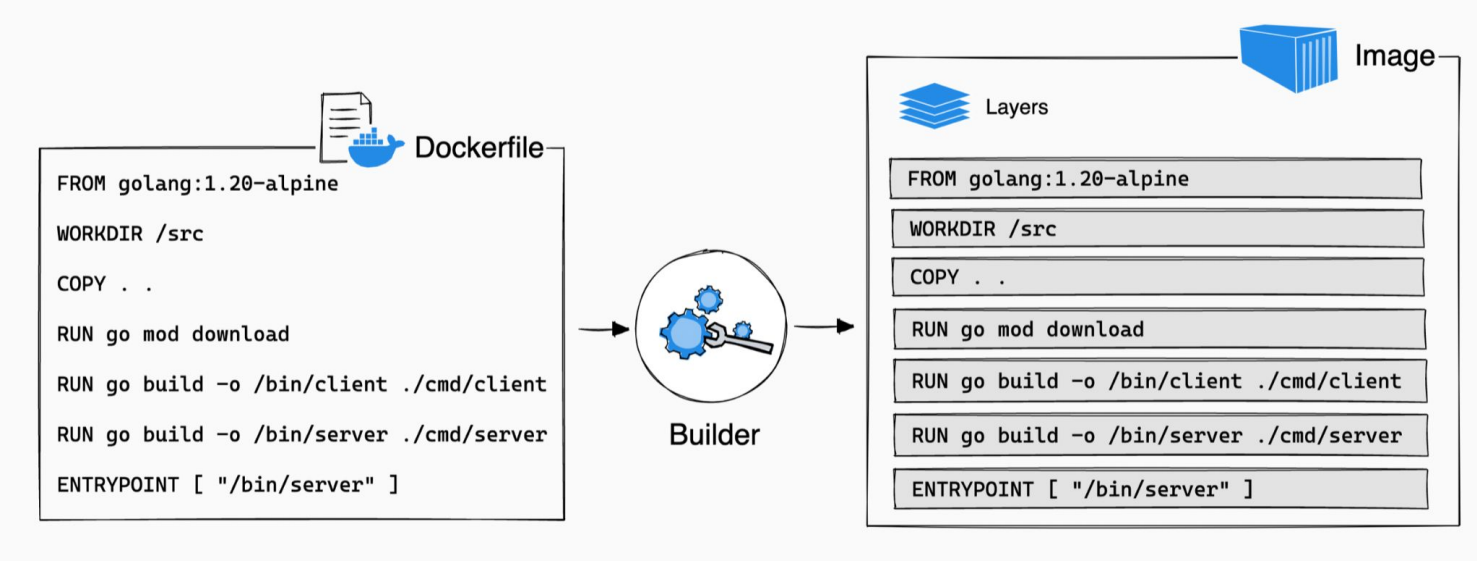
- Client - программа, из которой поступают команды
- Host - физическая машина, на которой располагаются контейнеры
- Docker daemon - компонент, который считывает команды клиента и управляет всеми объектами
- Images - инструкция того, каким должен быть контейнер
- Containers - сущность, создаваемая после запуска образа
- Registry - сервер, на котором хранятся готовые образы

Отношение между образом и контейнером



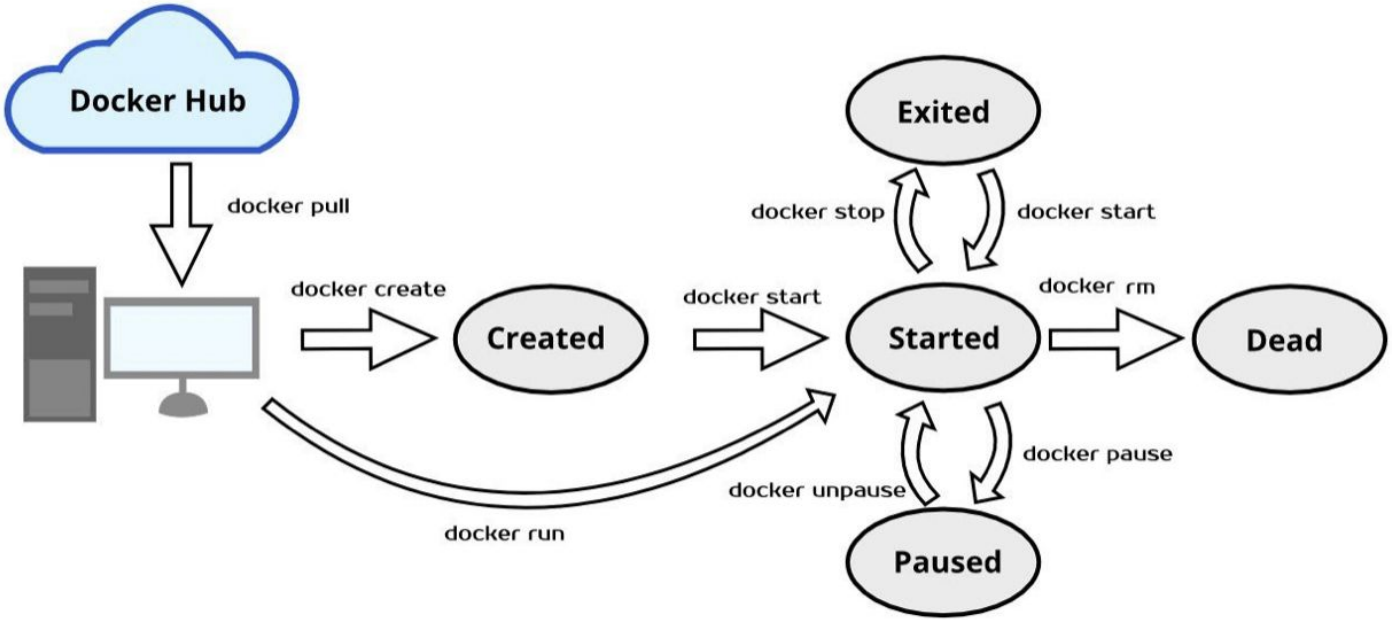
- Образ - план того, каким должен быть контейнер
- Контейнер - конкретная реализация, построенная на основе образа

Слои образа



- **Dockerfile -** файл с описанием процедуры создания файловой системы контейнера
- **Layer** - атомарный набор изменений файловой системы, как правило, описывается командой в Dockerfile

Жизненный цикл контейнера

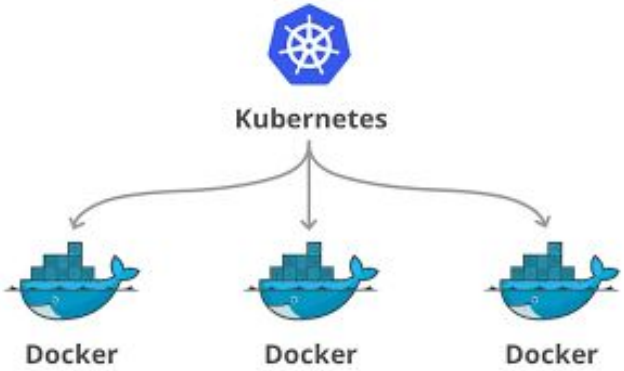


Какие есть альтернативы?

Построение сервиса из нескольких контейнеров

После того как все компоненты - контейнеры - программной системы будут созданы, производят их развёртку посредством специальных систем развёртывания, называемые оркестраторами контейнеров.

Основная задача, которую решают данные системы - масштабирование и координация контейнеров.



Что ещё почитать

1. https://docs.docker.com/get-started/
2. https://karpov.courses/docker
3. https://cloudacademy.com/blog/docker-vs-virtual-machines-differences-you-should-know/