



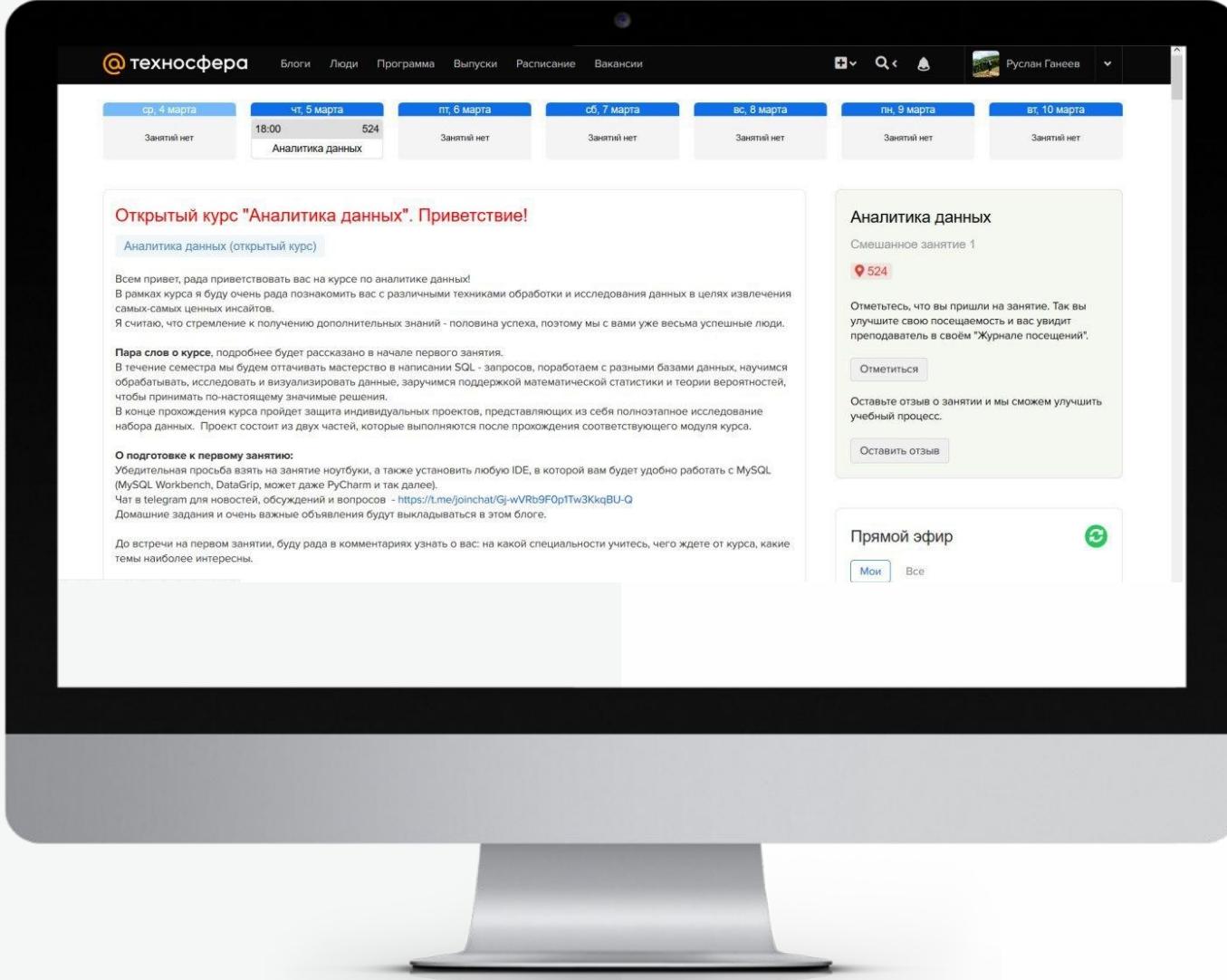
Микросервисная архитектура. Контейнеризация

Антон Кухтичев



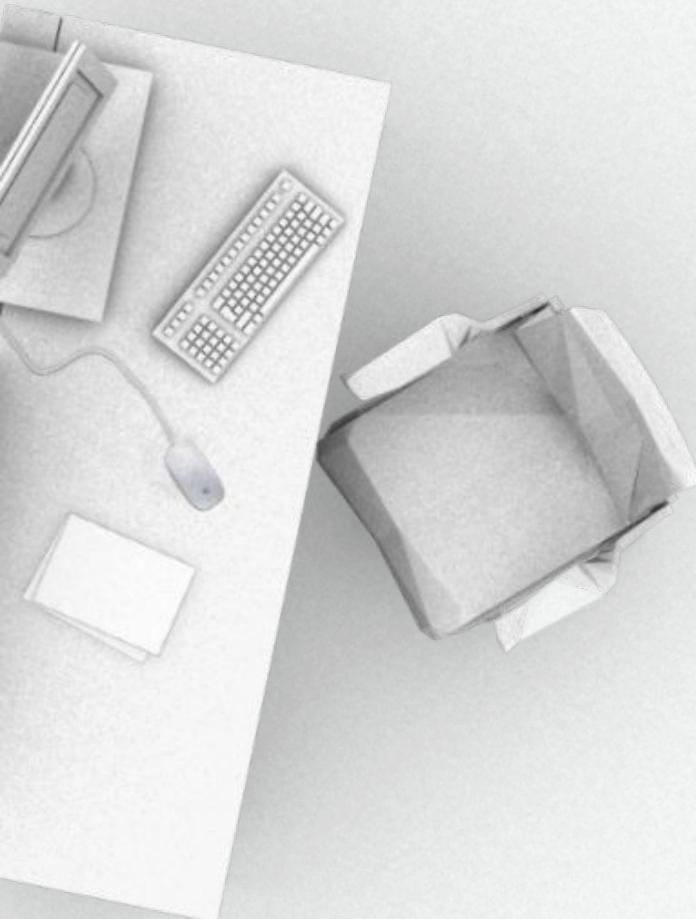
Содержание занятия

1. Микросервисная архитектура;
2. Контейнеризация;

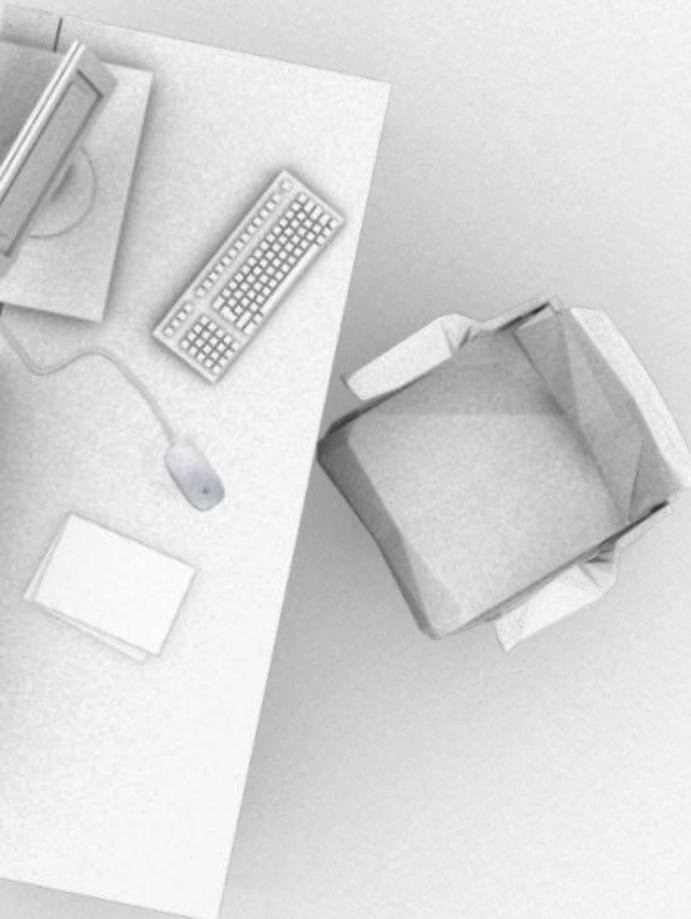


Напоминание отметиться на портале

Иначе плохо всё будет.



Квиз #11



Результаты квиза #10



Celery это

- распределённая очередь заданий на Python для хранения отложенных задач
- обвязка над системными утилитами cron/crontab
- удачное дополнение к любому салату
- драйвер для подключения к redis

Брокер очереди служит для

- исполнения CPU bound задач непосредственно
- передачи задач воркерам
- мониторинга работы celery
- синхронизации redis и rabbitMQ для работы с django

Отдельный воркер celery для запуска задач по расписанию называется

- celery delay
- celery cron
- celery beat
- celery solar
- celery runner

Что такое polling?

- синоним websocket
- открытие http соединения до сервера и удержание его сколько угодно долго до получения данных
- специальный модуль nginx
- запрос данных от сервера каждые N секунд по http

Для использования Comet на стороне сервера необходимо дополнительное ПО?

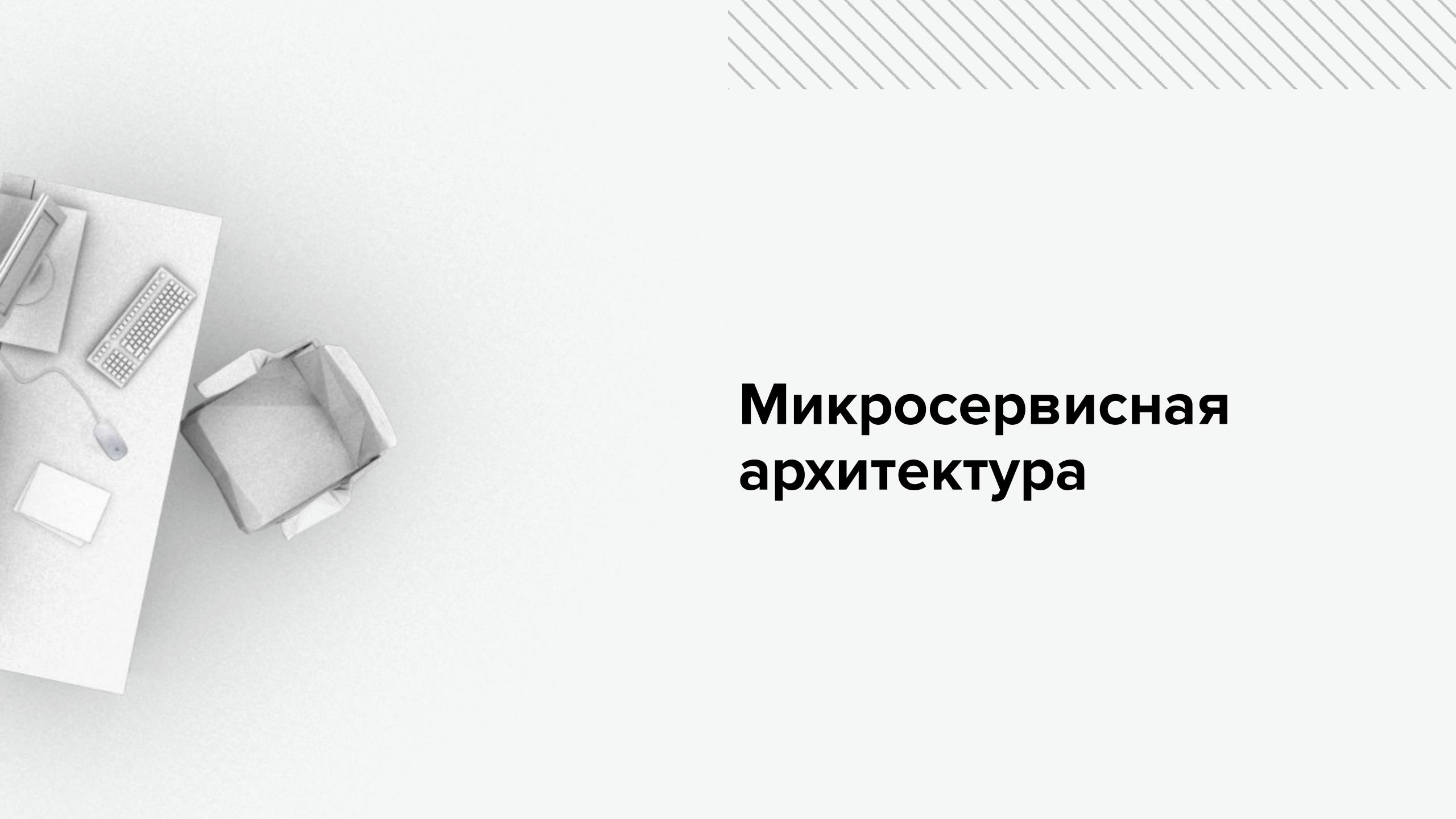
- да
- нет

Одним из главных недостатков server push технологии является

- невозможность фактической реализации на сервере
- поддержка не всеми браузерами
- слишком большое число избыточных запросов
- слишком большая нагрузка на CPU даже при малом числе пользователей

Для организации соединения по websocket клиент отправляет по http в том числе заголовок

- Upgrade: websocket
- Upgrade: connection
- Upgrade: json
- Upgrade: tcpsocket



Микросервисная архитектура

Что это такое микросервисы?

Микросервисы — это небольшие, автономные, совместно работающие сервисы. Небольшие и нацеленные на то, чтобы хорошо справляться только с одной работой.

Основные преимущества

- Технологическая разнородность;
- Устойчивость;
- Масштабирование;
- Простота развёртывания;
- Компонуемость;
- Оптимизация с последующей замены.

Принципы микросервисов

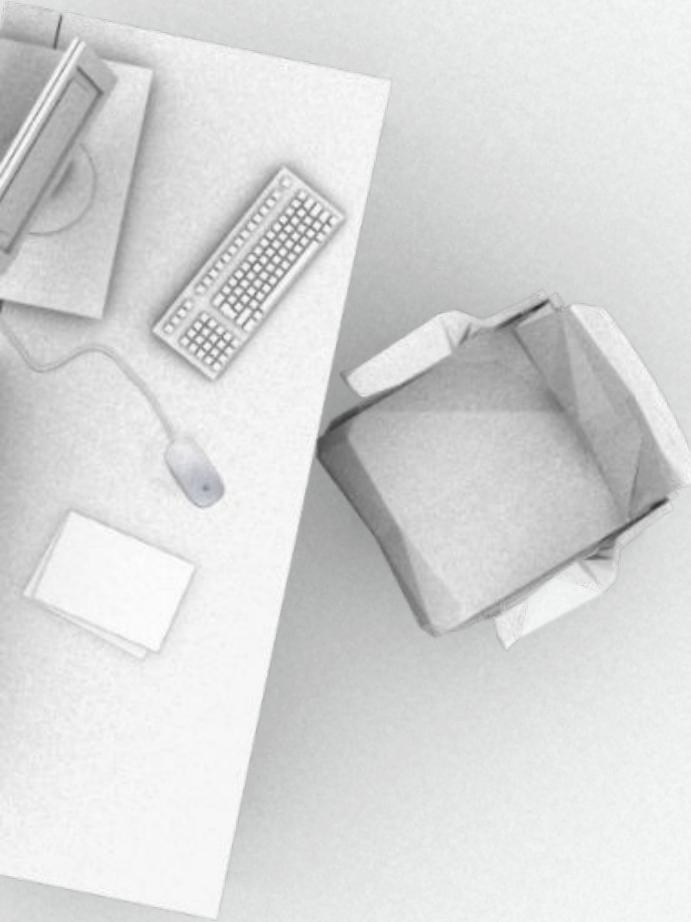


Немного критики

- Сетевые задержки;
- Форматы сообщений;
- Баланс нагрузки и отказоустойчивости;

Сравнение микросервисной и сервис-ориентированной архитектур

Параметр	SOA	Микросервисы
Межсервисное взаимодействие	Умные каналы, такие как сервисная шина предприятия, с использованием тяжеловесных протоколов вроде SOAP и других веб-сервисных стандартов	Примитивные каналы, такие как брокер сообщений, или прямое взаимодействие между сервисами с помощью легковесных протоколов наподобие REST или gRPC
Данные	Глобальная модель данных и общие БД	Отдельные модель данных и БД для каждого сервиса
Типовой сервис	Крупное монолитное приложение	Небольшой сервис



Контейнеризация

Что это такое?

1. Изоляция процессов
2. Ограничение ресурсов
 - a. CPU
 - b. RSS
 - c. I/O
 - d. Disk usage
3. Экосистема образов

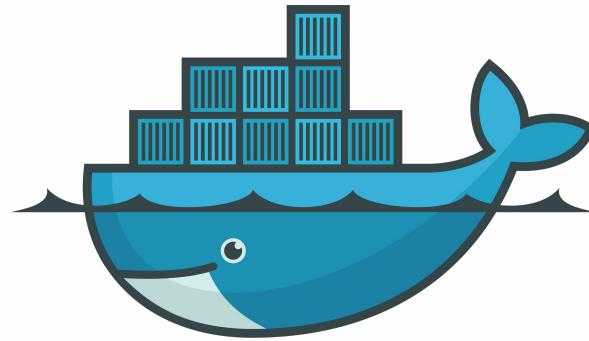
vs Виртуализация

- + Легковесность;
- + Почти нет накладных расходов;
- + Готовые образы, инфраструктура доставки;
- ОС / Ядро фиксированы;
- Худшая безопасность.

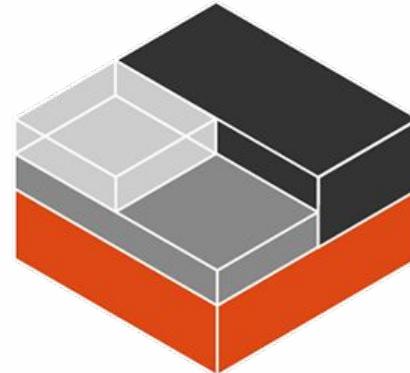
Зачем это нужно?

- Повышение утилизации железа;
- Гибкое управление зависимостями;
- Способ доставки ПО на сервера;
- Простое развёртывание тестовых сред;
- (*) Декларативное описание структуры проекта

Системы контейнеризации



docker



LXC



OpenVZ

Механизмы

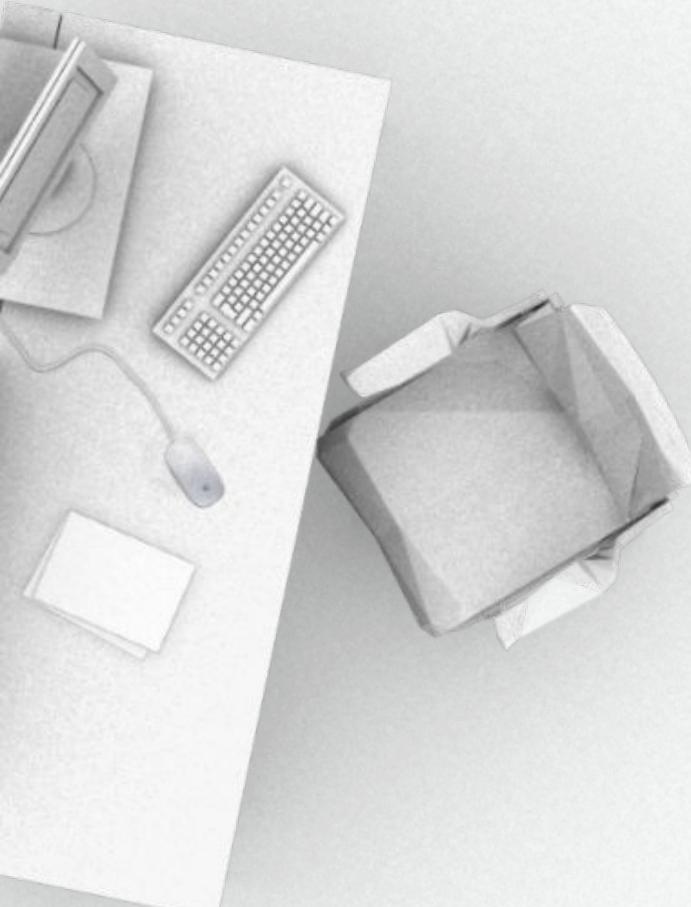
namespaces

Механизм изоляции: PID, NET, MNT, USER, ...

Отвечают за изоляцию контейнеров, гарантируют, что файловая система, имя хоста, пользователи полностью отделены от остальной части системы.

cgroups

Механизм, отвечающий за управление ресурсами, используемыми контейнером (процессор, оперативная память и т.д.).



Docker

Установка Docker

<https://docs.docker.com/engine/install/ubuntu/>

На 21.12.2020

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key  
add -  
  
sudo add-apt-repository \  
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \  
  $(lsb_release -cs) \  
  stable"  
  
sudo apt-get update  
  
sudo apt-get install docker-ce  
  
sudo usermod -aG docker `id -un`  
  
sudo systemctl start docker
```

А что пользователи MacOS?

Подробная инструкция [тут](#)

#027

Контейнеры

Контейнер / Container – группа процессов работающих в изолированном окружении, в своей файловой системе, возможно, с ограничением ресурсов.

Контейнер может содержать как одну запущенную программу (например Nginx), так и целое окружение (init, bash, и т.д.).

Основные команды

```
docker run -d nginx      # запустить контейнер  
docker ps                  # список контейнеров  
docker ps -a                # список всех контейнеров  
docker logs 5a592c        # посмотреть логи  
docker exec -it 5a592c bash # "подключиться"  
docker stop 5a592c        # остановить контейнер  
docker rm 5a592c          # удалить контейнер  
docker inspect 5a592c     # информация о контейнер
```

Образы

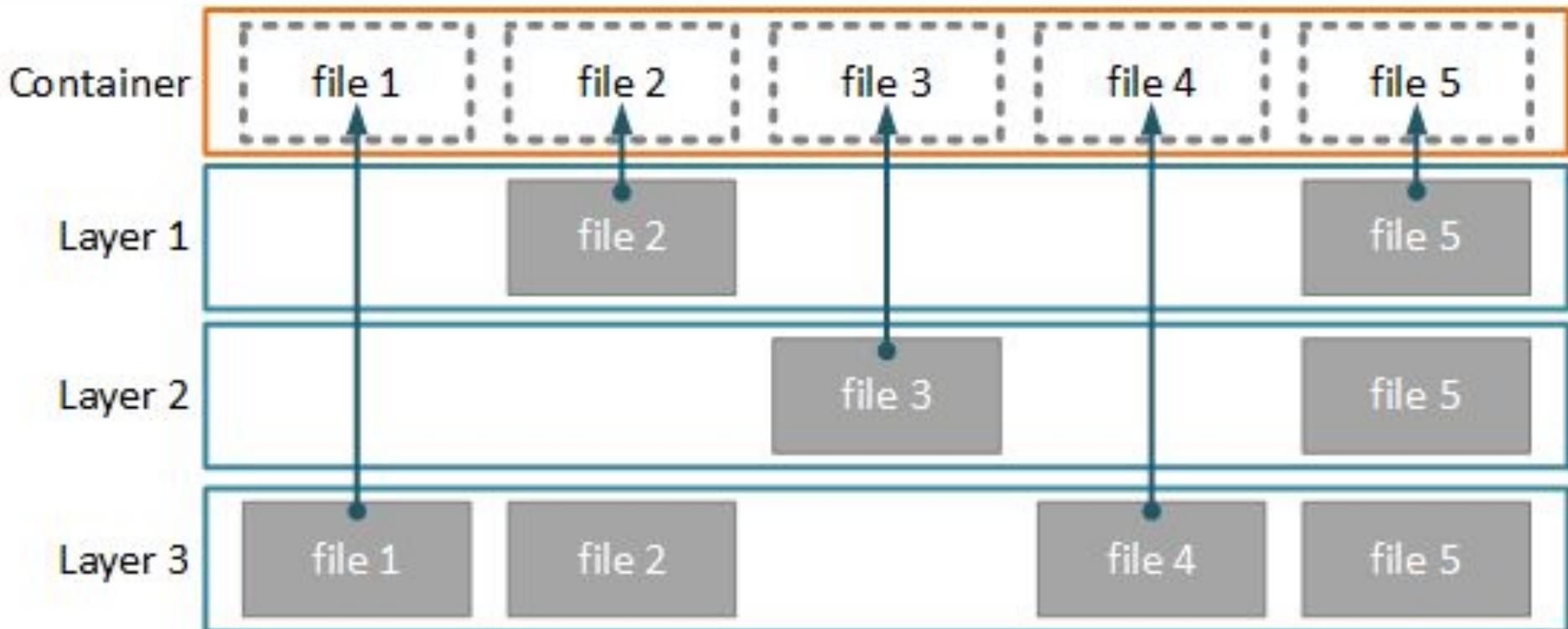
Образ / Image — образец (шаблон) файловой системы для контейнера. Образ содержит все необходимые для образу программы и файлы настроек, но не содержит пользовательских данных.

Образы могут наслаждаться друг на друга.

Основные команды

```
docker pull nginx      # скачать образ из registry  
docker images         # список образов  
docker rmi nginx     # удалить образ  
docker run -d nginx   # запустить контейнер  
                      # на основе образа  
docker push my_proj:v2 # загрузить образ в registry
```

OverlayFS



#032

Порты и директории (1)

Ок, а как использовать nginx ?

```
docker run -d --name ngx1 nginx
docker inspect -f '{{.NetworkSettings.IPAddress}}' ngx1
# 172.18.0.2
```

Проверяем:

<http://172.18.0.2/>

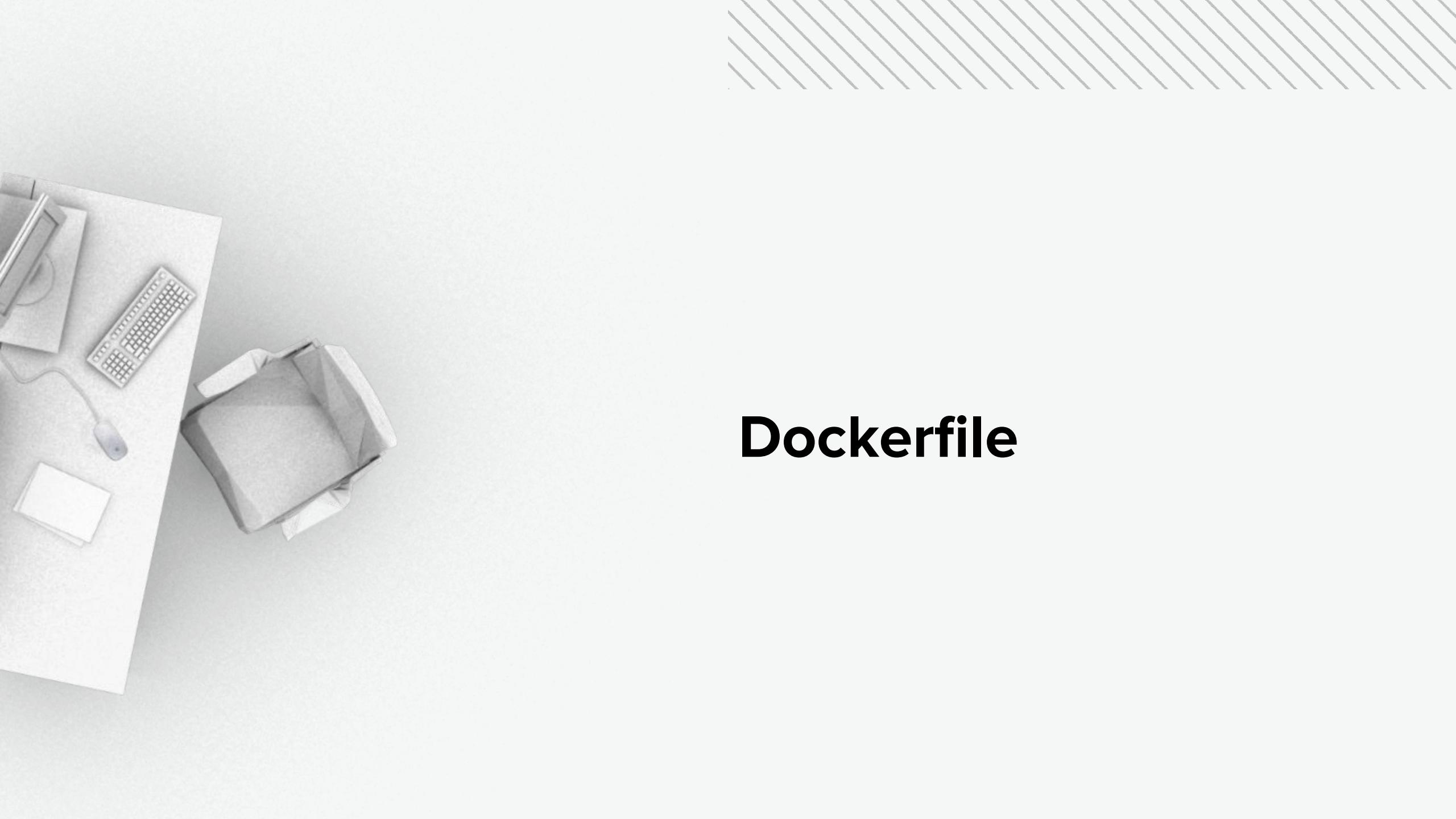
Порты и директории (2)

```
docker run -d \
    -p 8080:80 \
    -v /home/user/proj:/usr/share/nginx/html:ro \
    -e NGINX_HOST=foobar.com \
    --name ngx1 \
    nginx
```

-p local_port:container_port — проброс порта

-v local_dir:container_dir — проброс директории (volume)

-e NAME=val — установка переменной окружения



Dockerfile

Как собрать свой образ?

```
/path/to/project          # Сборочная директория
  ├── ask
  ├── askme
  ├── templates
  ├── static
  ├── manage.py
  ├── Dockerfile          # Сборка
  ├── docker-compose.yaml # Оркестрация
  └── .db_data            # Volume для базы
    requirements.txt
```

Синтаксис Dockerfile

```
FROM ubuntu:20.04
ADD . /app
RUN apt-get update
RUN apt-get install -y python3.6 python3-pip
RUN pip3 install -r /app/requirements.txt
EXPOSE 8000
USER nobody
WORKDIR /app
CMD /usr/local/bin/gunicorn askme.wsgi
```

Сборка образа

```
docker build -t askme:v2 /path/to/project
```

- `askme:v2` — название (и возможно тэг) образа
- `/path/to/project` — путь к директории с Dockerfile

Образ для разработки

В Dockerfile указываем точку монтирования

```
FROM ubuntu:20.04
```

```
...
```

```
VOLUME /app
```

```
...
```

При запуске образа монтируем директорию с проектом

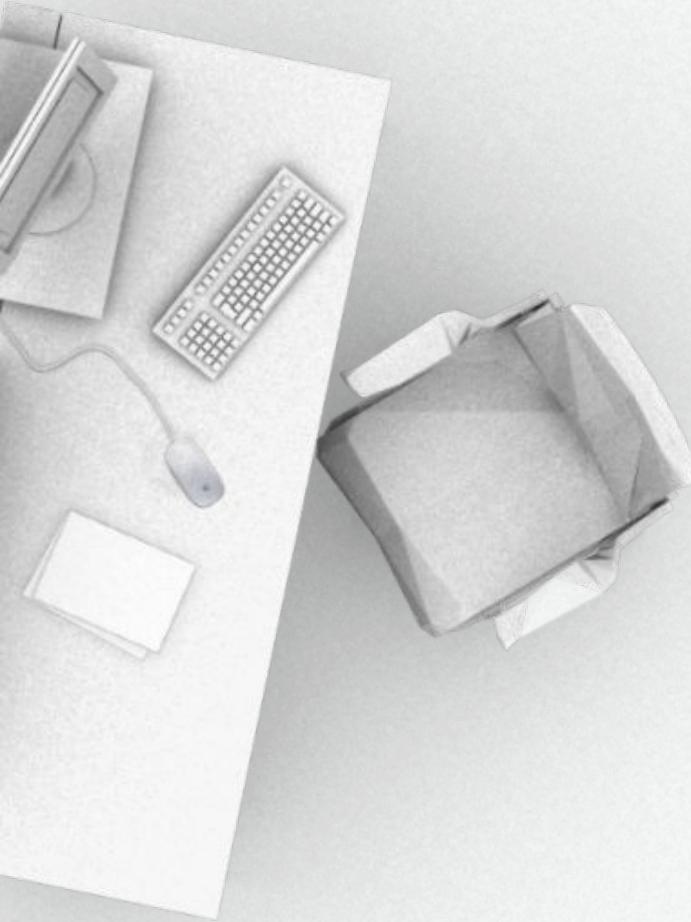
```
docker run -d -v /path/to/project:/app askme
```

Code time!



Попробуем написать два образа:

1. Cowsay + fortune;
2. Джанго-приложение.



Docker Compose

Проблема оркестрации

Для запуска нескольких взаимодействующих контейнеров нужно согласовать:

- IP адреса / имена хостов
- Логины и пароли
- Порядок запуска
- Проверка работоспособности

Это нужно сделать воспроизводимым.

Установка Compose

<https://docs.docker.com/compose/install/>

На 21.12.2020

```
sudo curl -L \
"https://github.com/docker/compose/releases/download/" \
"1.27.4/docker-compose-$(uname -s)-$(uname -m)" \
-o /usr/local/bin/docker-compose
```

```
sudo chmod +x /usr/local/bin/docker-compose
```

Синтаксис compose файла

```
version: "2.1"
services:
  serviceA:
    image: postgres:10
  serviceB:
    image: askme
  volumes:
    - host_dir:container_dir
    - host_port:container_port
```

Формат файла - YAML

Основные команды

```
docker-compose build      # пересобрать все образы
docker-compose create    # создать все контейнеры
docker-compose start     # запустить контейнеры
docker-compose stop      # остановить контейнеры
docker-compose rm        # удалить контейнеры
docker-compose logs      # посмотреть логи
docker-compose up        # build, create, start, logs -f

# часто можно указать конкретный контейнер
docker-compose restart webapp
```



Makefile

Makefile

up:

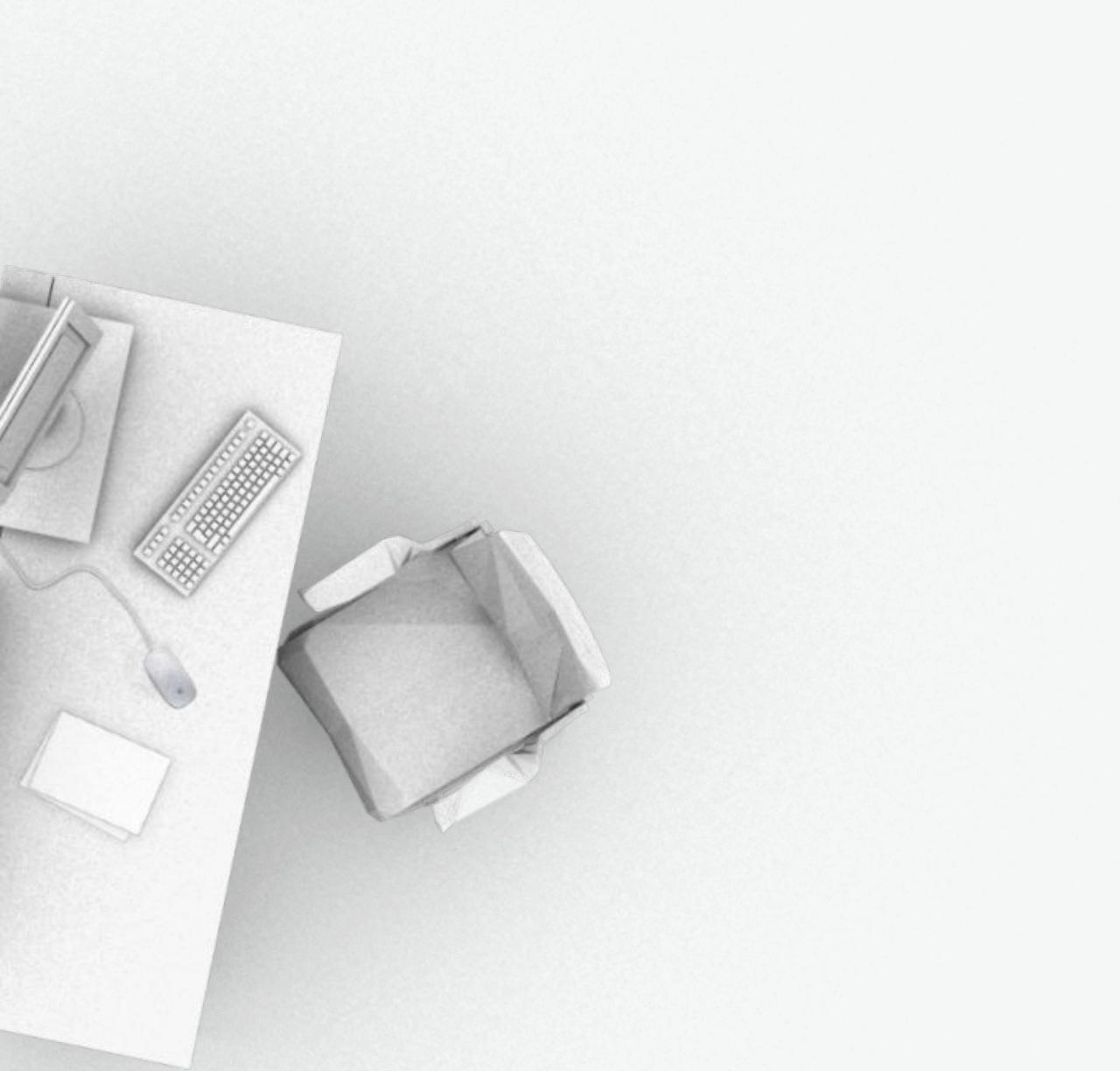
```
    docker-compose up
```

test: up

```
    docker-compose exec webapp python3 /app/manage.py test
```

migrate: up

```
    docker-compose exec webapp python3 /app/manage.py migrate
```



Домашнее задание

Домашнее задание #12

1. Установить docker и docker-compose (1 балл);
2. Создание Dockerfile для Django приложения (2 балла);
3. Создание docker-compose для проекта:
 - a. nginx (2 балла),
 - b. База данных (2 балла),
 - c. elasticsearch (2 балла).
4. Создание Makefile для проекта (1 балл);

Преподаватель должен иметь возможность, имея установленными только git, docker и docker-compose склонировать проект, выполнить команды `make migrate` и увидеть успешную миграцию.

Домашнее задание по уроку #12

Домашнее задание №12

#050

9

Баллов
за задание

Сроков нет, но
вы держитесь

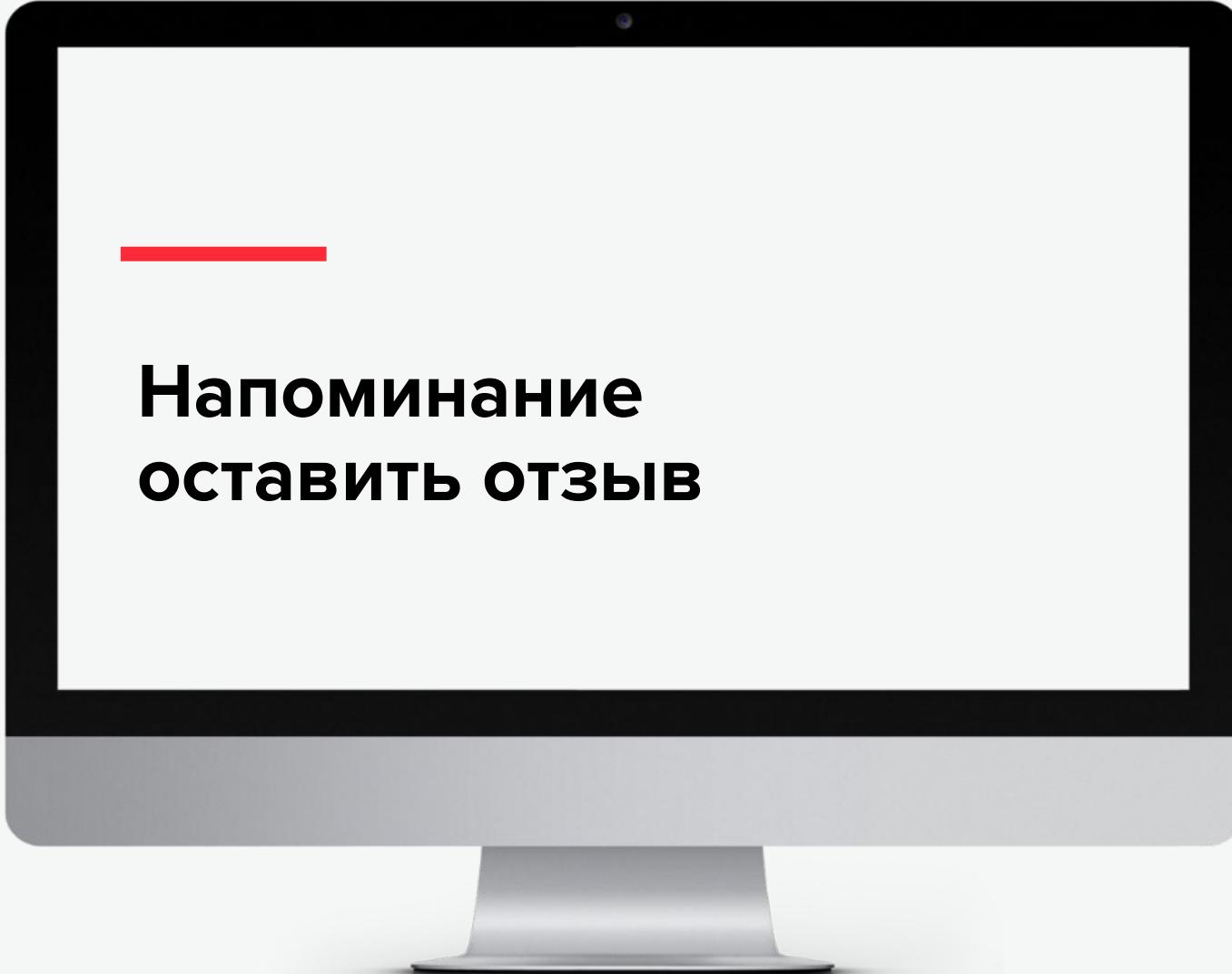
Срок
сдачи

- [«Docker на практике» | Сейерс Э.Х., Милл А.](#)
- [«Использование Docker» | Моэт Эдриен](#)
- [«Микросервисы. Паттерны разработки и рефакторинга» | Ричардсон Крис](#)
- [«Создание микросервисов» | Сэм Ньюмен](#)
- [«Чистая архитектура. Искусство разработки программного обеспечения» | Мартин Роберт](#)



Для саморазвития (опционально)

Чтобы не набирать двумя пальчиками



**Напоминание
оставить отзыв**

**СПАСИБО
ЗА ВНИМАНИЕ**

