Лекция 14: Проектирование распределённых приложений

Часть вторая: стратегические вопросы

Юрий Литвинов
y.litvinov@spbu.ru

14.12.2021

Representational State Transfer (REST)

- Модель клиент-сервер
- Отсутствие состояния
- Кэширование
- Единообразие интерфейса
- Слои

Интерфейс сервиса

- Коллекции
  - http://api.example.com/resources/
- Элементы
  - http://api.example.com/resources/item/17
- HTTP-методы
  - GET
  - PUT
  - POST
  - DELETE
- Передача параметров прямо в URL
  - http://api.example.com/resources?user=me&access_token=ASFQF

Пример, Google Drive REST API

- GET https://www.googleapis.com/drive/v2/files — список всех файлов
- GET https://www.googleapis.com/drive/v2/files/fileId — метаданные файла по его Id
- POST https://www.googleapis.com/upload/drive/v2/files — загрузить новый файл
- PUT https://www.googleapis.com/upload/drive/v2/files/fileId — обновить файл
- DELETE https://www.googleapis.com/drive/v2/files/fileId — удалить файл

Достоинства

- Надёжность
- Производительность
- Масштабируемость
- Прозрачность системы взаимодействия
- Простота интерфейсов
- Портативность компонентов
- Лёгкость внесения изменений

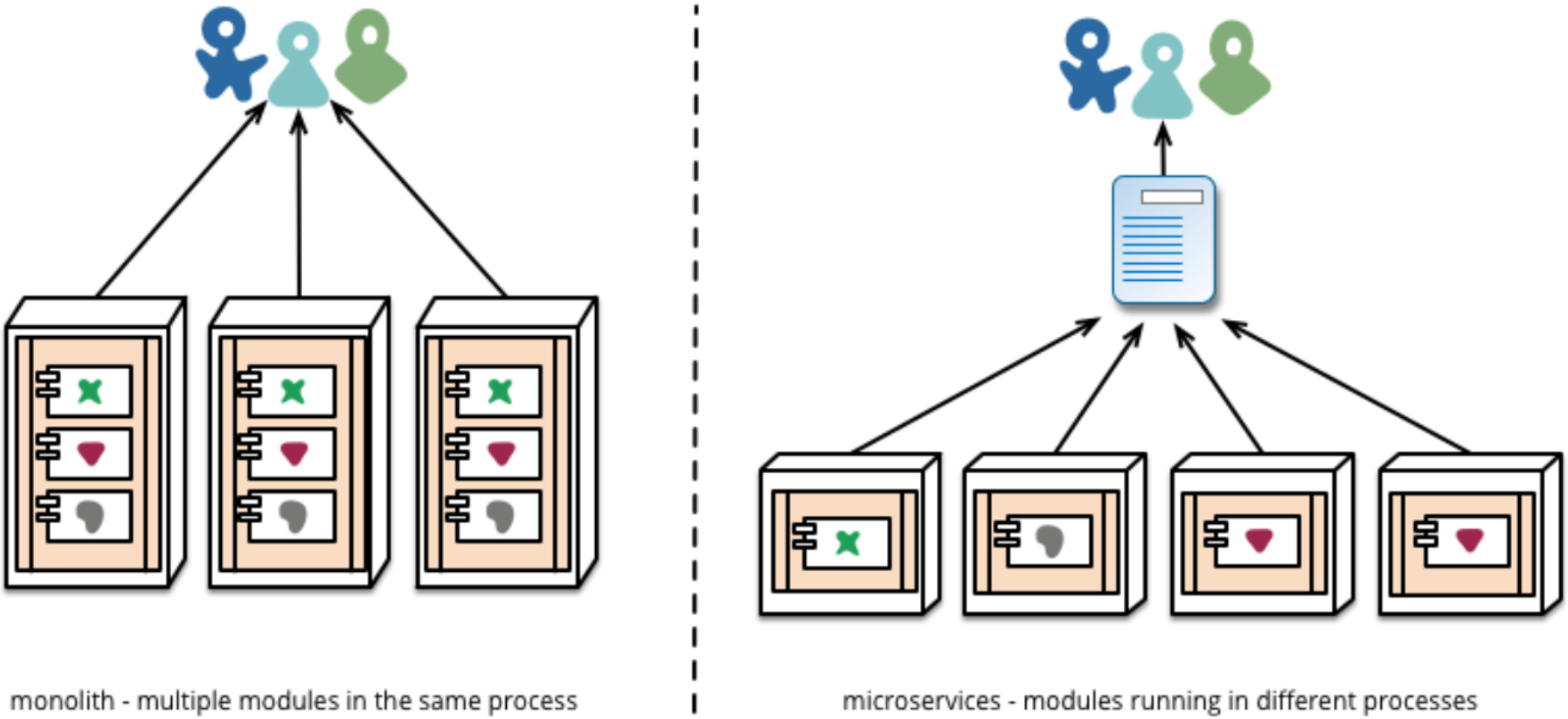
Микросервисы

- Набор небольших сервисов
  - Разные языки и технологии
- Каждый в собственном процессе
  - Независимое развёртывание
  - Децентрализованное управление
- Легковесные коммуникации

Монолитные приложения

- Большой и сложный MVC
- Единый процесс разработки и стек технологий
- Сложная архитектура
- Сложно масштабировать
- Сложно вносить изменения

Разбиение на сервисы



Основные особенности

- Микросервисы и SOA
- Smart endpoints and dumb pipes
- Проектирование под отказ
- Асинхронные вызовы
- Децентрализованное управление данными
- Автоматизация инфраструктуры
- Эволюционный дизайн

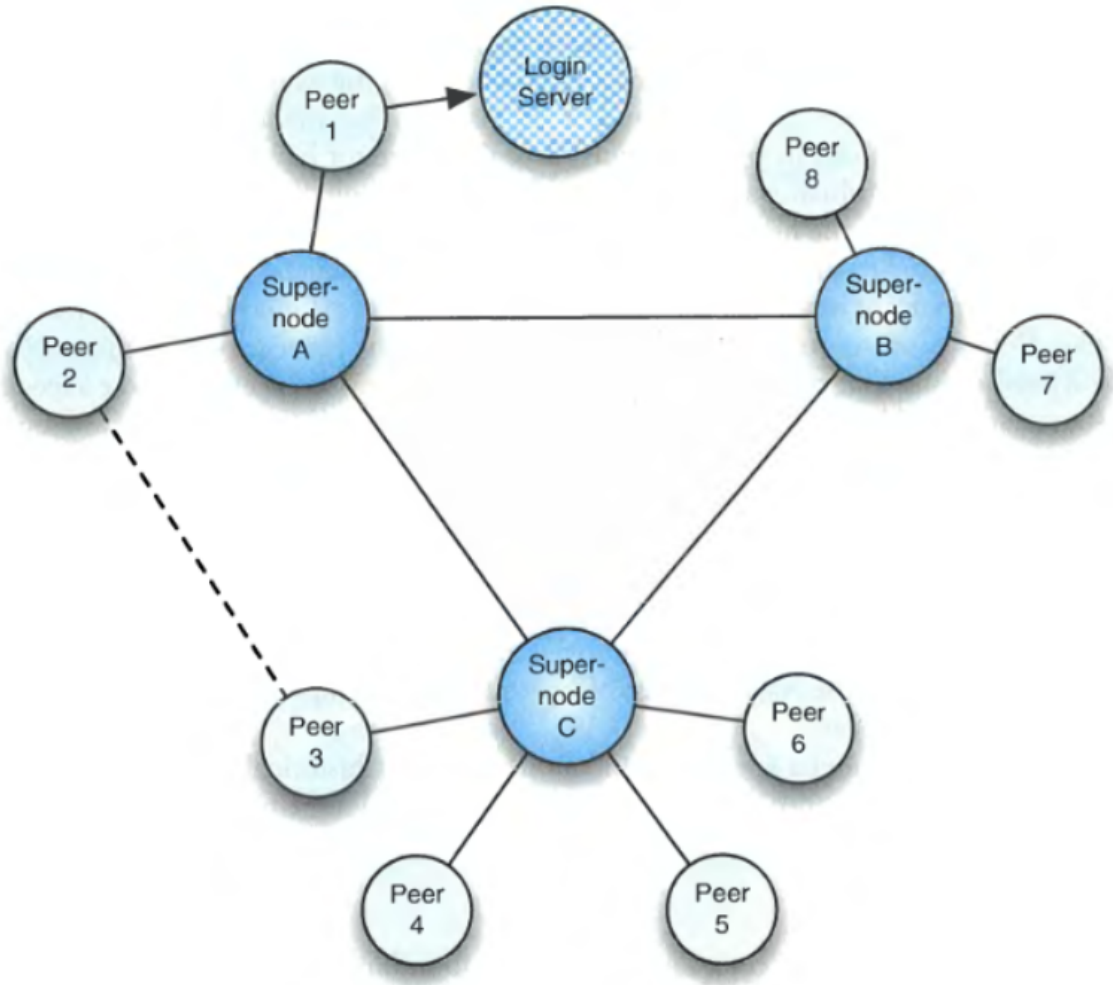
Основные проблемы

- Сложности выделения границ сервисов
- Перенос логики на связи между сервисами
  - Большой обмен данными
  - Нетривиальные зависимости
- Нетривиальная инфраструктура
- Нетривиальная переиспользуемость кода

Архитектура Peer-to-Peer

- Децентрализованный и самоорганизующийся сервис
- Динамическая балансировка нагрузки
  - Вычислительные ресурсы
  - Хранилища данных
- Динамическое изменение состава участников

Skype: Overlayed P2P

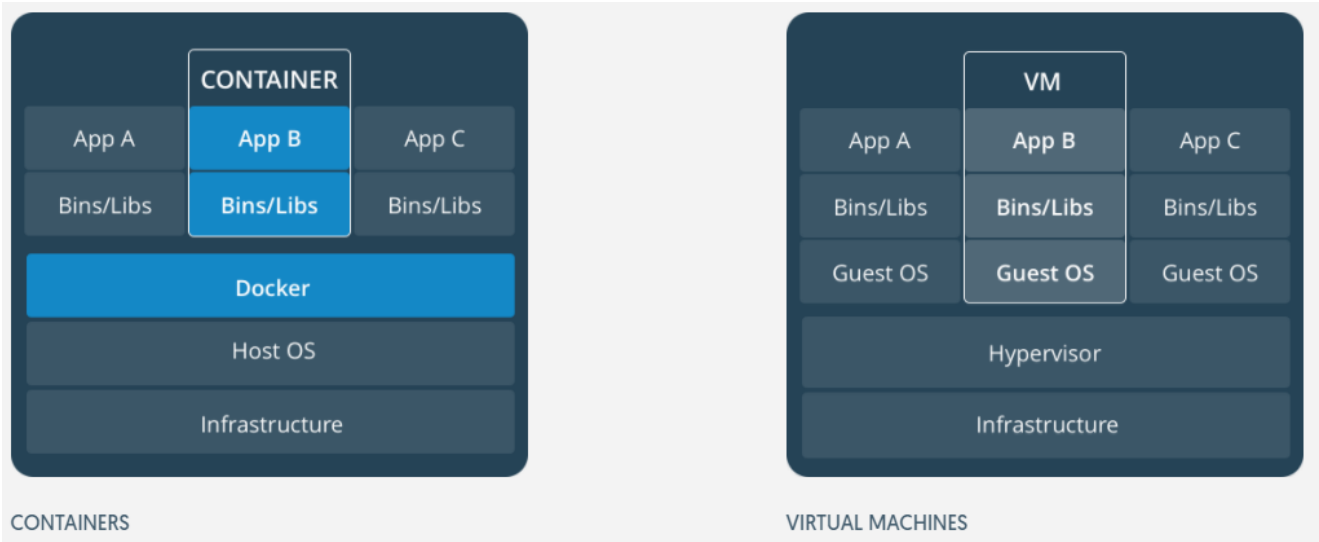


BitTorrent : Resource Trading P2P

- Обмен сегментами
- Поиск не входит в протокол
- Трекеры
- Метаданные
- Управление приоритетами
- Бестрекерная реализация

Docker

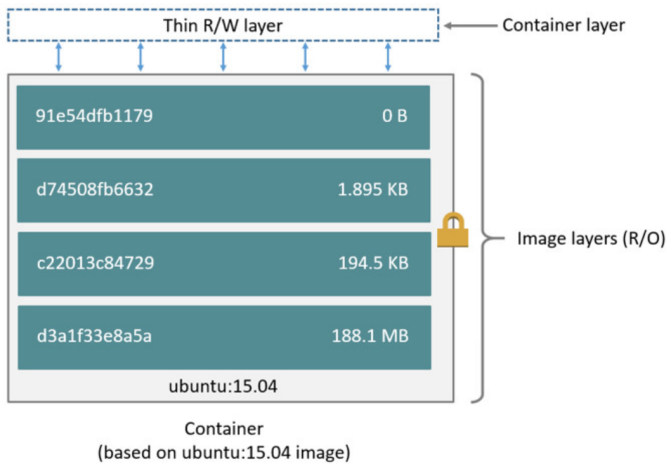
- Средство для “упаковки” приложений в изолированные контейнеры
- Что-то вроде легковесной виртуальной машины
-
- Широкий инструментарий: DSL для описания образов, публичный репозиторий, поддержка оркестраторами



© https://www.docker.com

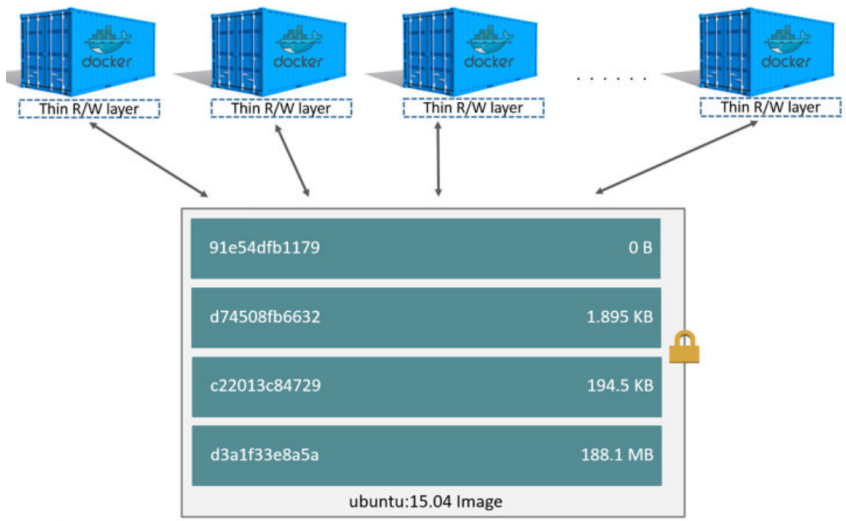
Docker Image

- Окружение и приложение
- Состоит из слоёв
  - Все слои read-only
  - Образы делят слои между собой как процессы делят динамические библиотеки
- На основе одного образа можно создать другой



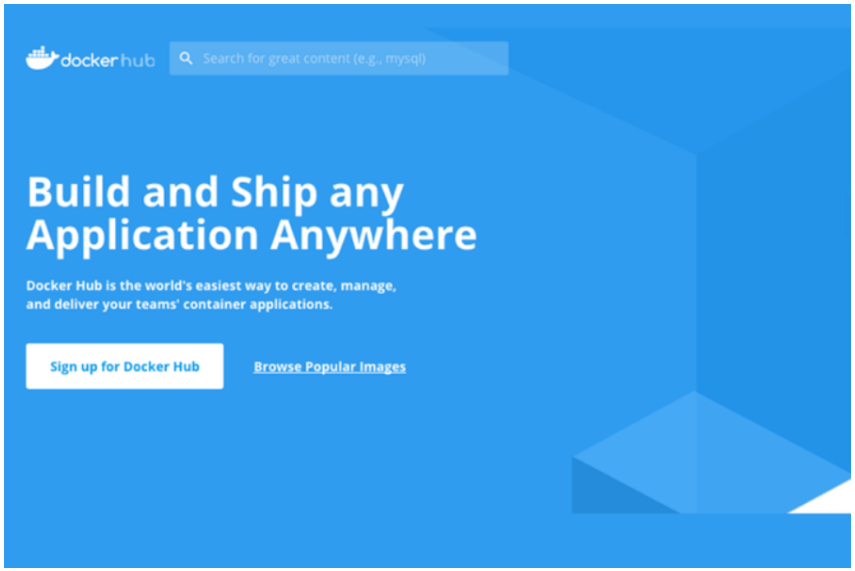
Docker Container

- Образ с дополнительным write слоем
- Содержит один запущенный процесс
- Может быть сохранен как новый образ



DockerHub

- Внешний репозиторий образов
  - Официальные образы
  - Пользовательские образы
  - Приватные репозитории
- Простой CI/CD
- Высокая доступность



Базовые команды

- docker run — запускает контейнер (при необходимости делает pull)
  - -d — запустить в фоновом режиме
  - -p host_port:container_port — прокинуть порт из контейнера на хост
  - -i -t — запустить в интерактивном режиме
  - Пример: docker run -it ubuntu /bin/bash
- docker ps — показывает запущенные контейнеры
  - Пример: docker run -d nginx; docker ps
- docker stop — останавливает контейнер (шлёт SIGTERM, затем SIGKILL)
- docker exec — запускает дополнительный процесс в контейнере

Dockerfile

```
# Use an official Python runtime as a parent image
FROM python:2.7-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
ADD . /app

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

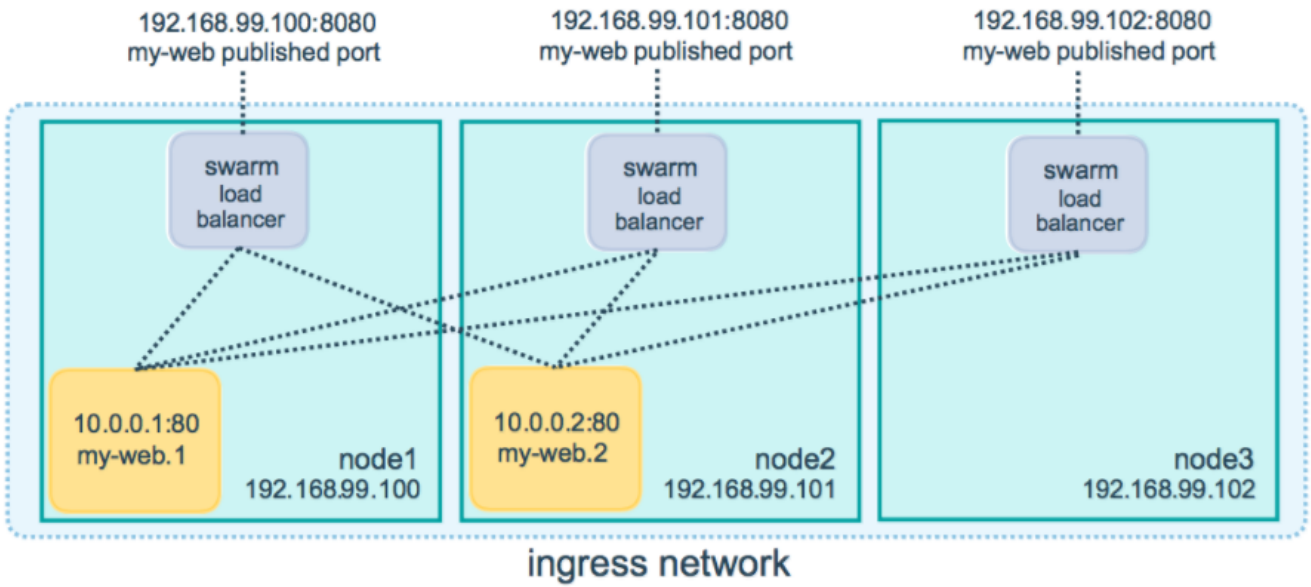
Балансировка нагрузки

docker-compose.yml

```yaml
version: "3"
services:
  web:
    # replace username/repo:tag with your name and image details
    image: username/repo:tag
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
      restart_policy:
        condition: on-failure
    ports:
      - "80:80"
    networks:
      - webnet
networks:
  webnet:
```

Swarm-ы

- Машина, на которой запускается контейнер, становится главной
- Другие машины могут присоединяться к swarm-у и получать копию контейнера
- Docker балансирует нагрузку по машинам



© https://www.docker.com