

Лекция 14: Проектирование распределённых приложений

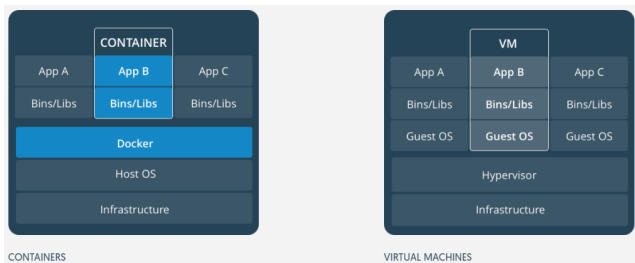
Часть вторая: высокоуровневые вещи

Юрий Литвинов
yurii.litvinov@gmail.com

08.12.2020г

Docker

- ▶ Средство для “упаковки” приложений в изолированные контейнеры
- ▶ Что-то вроде легковесной виртуальной машины



© <https://www.docker.com>

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

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:

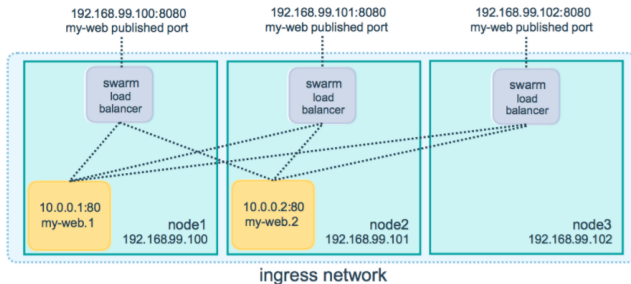
- webnet

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>

SOAP-ориентированные сервисы

- ▶ Simple Object Access Protocol
- ▶ Web Services Description Language
- ▶ Universal Discovery, Description and Integration



SOAP-сообщение

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
      <n:priority>1</n:priority>
      <n:expires>2001-06-22T14:00:00-05:00</n:expires>
    </n:alertcontrol>
  </env:Header>
  <env:Body>
    <m:alert xmlns:m="http://example.org/alert">
      <m:msg>Get up at 6:30 AM</m:msg>
    </m:alert>
  </env:Body>
</env:Envelope>
```

WSDL-описание

```
<message name="getTermRequest">  
  <part name="term" type="xs:string"/>  
</message>
```

```
<message name="getTermResponse">  
  <part name="value" type="xs:string"/>  
</message>
```

```
<portType name="glossaryTerms">  
  <operation name="getTerm">  
    <input message="getTermRequest"/>  
    <output message="getTermResponse"/>  
  </operation>  
</portType>
```


Достоинства SOAP-based сервисов

- ▶ Автоматический режим описания сервисов
- ▶ Автоматическая поддержка описаний SOAP-клиентом
- ▶ Автоматическая валидация сообщений
 - ▶ Валидность xml
 - ▶ Проверка по схеме
 - ▶ Проверка SOAP-сервером
- ▶ Работа через HTTP
 - ▶ Хоть через обычный GET

Недостатки SOAP-based сервисов

- ▶ Огромный размер сообщений
- ▶ Сложность описаний на клиенте и сервере
- ▶ Один запрос — один ответ
 - ▶ Поддержка транзакций на уровне бизнес-логики
- ▶ Сложности миграции при изменении описания

Пример: WCF

- ▶ Платформа для создания веб-сервисов
- ▶ Часть .NET Framework, начиная с 3.0
- ▶ Умеет WSDL, SOAP и т.д., очень конфигурируема
- ▶ Автоматическая генерация заглушек на стороне клиента
- ▶ ABCs of WCF:
 - ▶ Address
 - ▶ Binding
 - ▶ Contract



© <http://www.c-sharpcorner.com>

Пример, описание контракта

```
[ServiceContract(Namespace = "http://Microsoft.ServiceModel.Samples")]  
public interface ICalculator  
{  
    [OperationContract]  
    double Add(double n1, double n2);  
  
    [OperationContract]  
    double Subtract(double n1, double n2);  
  
    [OperationContract]  
    double Multiply(double n1, double n2);  
  
    [OperationContract]  
    double Divide(double n1, double n2);  
}
```

Пример, реализация контракта

```
public class CalculatorService : ICalculator
{
    public double Add(double n1, double n2)
        => n1 + n2;

    public double Subtract(double n1, double n2)
        => n1 - n2

    public double Multiply(double n1, double n2)
        => n1 * n2;

    public double Divide(double n1, double n2)
        => n1 / n2;
}
```

Пример, self-hosted service

```

static void Main(string[] args)
{
    Uri baseAddress = new Uri("http://localhost:8000/ServiceModelSamples/Service");
    ServiceHost selfHost = new ServiceHost(typeof(CalculatorService), baseAddress);

    try {
        selfHost.AddServiceEndpoint(typeof(ICalculator), new WSHttpBinding(), "CalculatorService");

        ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
        smb.HttpGetEnabled = true;
        selfHost.Description.Behaviors.Add(smb);

        selfHost.Open();
        Console.WriteLine("The service is ready. Press <ENTER> to terminate service.");
        Console.ReadLine();

        selfHost.Close();
    } catch (CommunicationException ce) {
        Console.WriteLine($"An exception occurred: {ce.Message}");
        selfHost.Abort();
    }
}

```

Пример, клиент

► Генерация заглушки:

```
svcutil.exe /language:cs /out:generatedProxy.cs /config:app.config^  
http://localhost:8000/ServiceModelSamples/service
```

► Клиент:

```
static void Main(string[] args)  
{  
    CalculatorClient client = new CalculatorClient();  
  
    double value1 = 100.00D;  
    double value2 = 15.99D;  
    double result = client.Add(value1, value2);  
    Console.WriteLine($"Add({value1},{value2}) = {result}");  
  
    client.Close();  
}
```

Пример, конфигурация клиента

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <!-- specifies the version of WCF to use-->
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5,Profile=Client" />
  </startup>
  <system.serviceModel>
    <bindings>
      <!-- Uses wsHttpBinding-->
      <wsHttpBinding>
        <binding name="WSHttpBinding_ICalculator" />
      </wsHttpBinding>
    </bindings>
    <client>
      <!-- specifies the endpoint to use when calling the service -->
      <endpoint address="http://localhost:8000/ServiceModelSamples/Service/CalculatorService"
        binding="wsHttpBinding" bindingConfiguration="WSHttpBinding_ICalculator"
        contract="ServiceReference1.ICalculator" name="WSHttpBinding_ICalculator">
        <identity>
          <userPrincipalName value="migree@redmond.corp.microsoft.com" />
        </identity>
      </endpoint>
    </client>
  </system.serviceModel>
</configuration>

```


Очереди сообщений

- ▶ Используются для гарантированной доставки сообщений
 - ▶ Даже если отправитель и получатель доступны в разное время
 - ▶ Локальное хранилище сообщений на каждом устройстве
- ▶ Реализуют модель “издатель-подписчик”, но могут работать и в режиме “точка-точка”
- ▶ Как правило, имеют развитые возможности маршрутизации, фильтрации и преобразования сообщений
 - ▶ Разветвители, агрегаторы, преобразователи порядка

RabbitMQ

- ▶ Сервер и клиенты системы надёжной передачи сообщений
 - ▶ Сообщение посылается на сервер и хранится там, пока его не заберут
 - ▶ Продвинутые возможности по маршрутизации сообщений
- ▶ Реализует протокол AMQP (Advanced Message Queuing Protocol), но может использовать и другие протоколы
- ▶ Сервер написан на Erlang, клиентские библиотеки доступны для практически чего угодно



Пример, отправитель

```
using System;
using RabbitMQ.Client;
using System.Text;

class Send
{
    public static void Main()
    {
        var factory = new ConnectionFactory() { HostName = "localhost" };
        using (var connection = factory.CreateConnection())
        {
            using (var channel = connection.CreateModel())
            {
                channel.QueueDeclare(queue: "hello", durable: false, exclusive: false,
                    autoDelete: false, arguments: null);

                string message = "Hello World!";
                var body = Encoding.UTF8.GetBytes(message);

                channel.BasicPublish(exchange: "", routingKey: "hello",
                    basicProperties: null, body: body);
            }
        }
    }
}
```

Пример, получатель

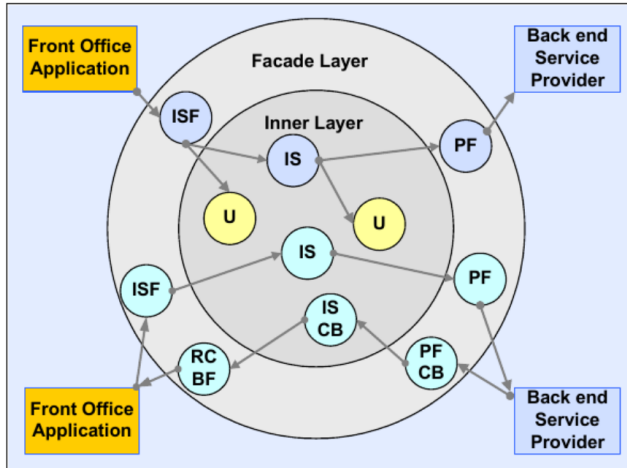
```
using RabbitMQ.Client;
using RabbitMQ.Client.Events;
using System;
using System.Text;
```

```
class Receive
```

```
{
    public static void Main()
    {
        var factory = new ConnectionFactory() { HostName = "localhost" };
        using (var connection = factory.CreateConnection())
        using (var channel = connection.CreateModel())
        {
            channel.QueueDeclare(queue: "hello", durable: false, exclusive: false, autoDelete: false, arguments: null);

            var consumer = new EventingBasicConsumer(channel);
            consumer.Received += (model, ea) =>
            {
                var body = ea.Body;
                var message = Encoding.UTF8.GetString(body);
                Console.WriteLine("[x] Received {0}", message);
            };
            channel.BasicConsume(queue: "hello", autoAck: true, consumer: consumer);
        }
    }
}
```

Enterprise Service Bus



Representational State Transfer (REST)

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

Интерфейс сервиса

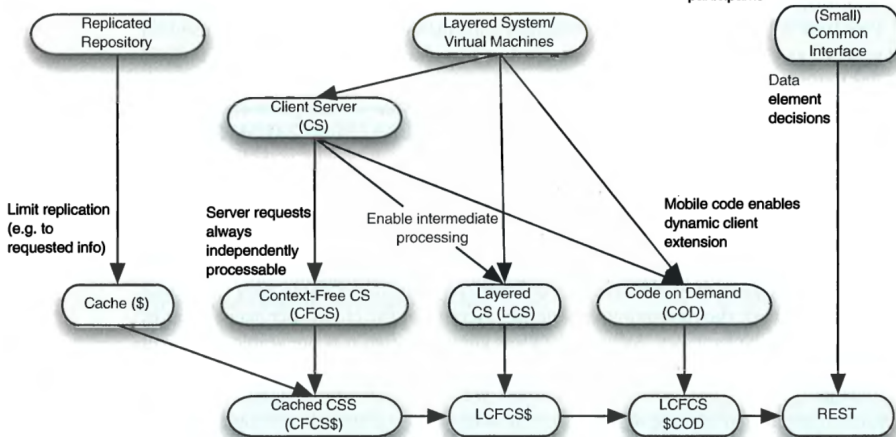
- ▶ Коллекции
 - ▶ `http://api.example.com/resources/`
- ▶ Элементы
 - ▶ `http://api.example.com/resources/item/17`
- ▶ HTTP-методы
 - ▶ GET
 - ▶ PUT
 - ▶ POST
 - ▶ DELETE
- ▶ Передача параметров прямо в URL
 - ▶ `http://api.example.com/resources?user=me&access_token=ASFQF`

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

Multiple equivalent sources
promote efficiency and
robustness

Small number of common
interfaces demanded of all
participants



Достоинства

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

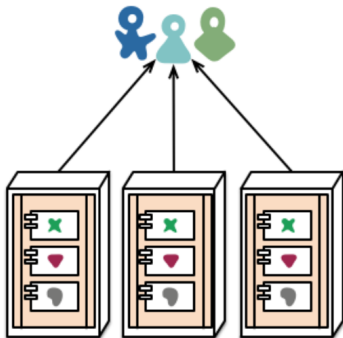
Микросервисы

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

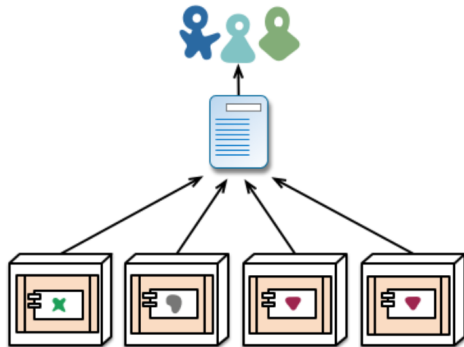
Монолитные приложения

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

Разбиение на сервисы



monolith - multiple modules in the same process



microservices - modules running in different processes

Основные особенности

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

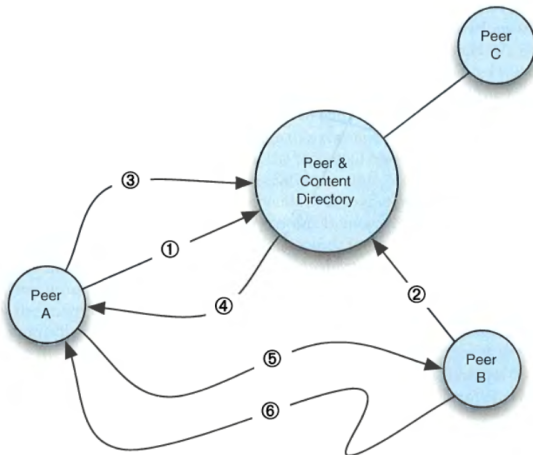
Основные проблемы

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

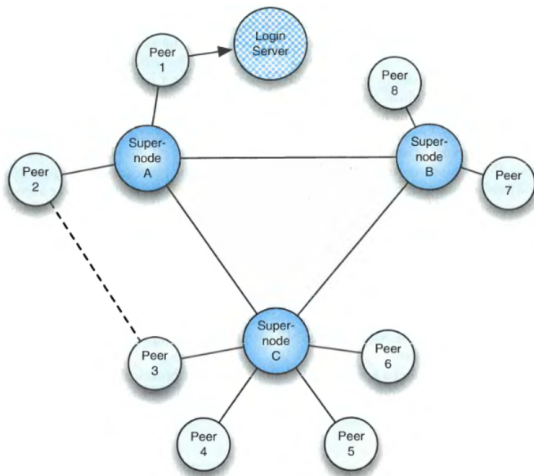
Архитектура Peer-to-Peer

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

Napster: hybrid client-server/P2P



Skype: Overlaid P2P



BitTorrent : Resource Trading P2P

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