Микросервисы. Docker

2 декабря 2016

Владимир Ананьев
АО "Программный Регион"

Что такое микросервисная архитектура (MSA)?

Это подход, при котором информационная система строится как набор **небольших сервисов**, каждый из которых работает в собственном процессе и коммуницирует с остальными используя легковесные механизмы, как правило HTTP

Философия микросервисов фактически копирует философию UNIX, а именно **«Делай что-то одно и делай это хорошо»**. Суть заключается в следующем:

- Сервисы - небольшие, выполняют каждый свою, единственную функцию
- Организационная культура включает в себя автоматизацию разработки и тестирования, это снижает нагрузку на управление
- Каждый сервис - эластичный, легко меняющийся, элементарный и при этом законченный

Преимущества микросервисной архитектуры

- Писать небольшие сервисы всегда проще
- Удобство разработки - разные микросервисы можно легко распределять между разными командами разработки
- Бизнес-процессы очень хорошо переносятся на микросервисы (!)
- Для каждого микросервиса можно использовать абсолютно любую технолгию и язык программирования
- Меньше ошибок в коде, т.к. кода меньше
- Упрощенное модульное тестирование
- Упрощенный деплой

Недостатки микросервисной архитектуры

- Сетевые задержки
- Дополнительное API
- Форматы сообщений
- Дополнительные расходы на сериализацию/десериализацию
- Дополнительный мониторинг
- Сложности использования общих библиотек

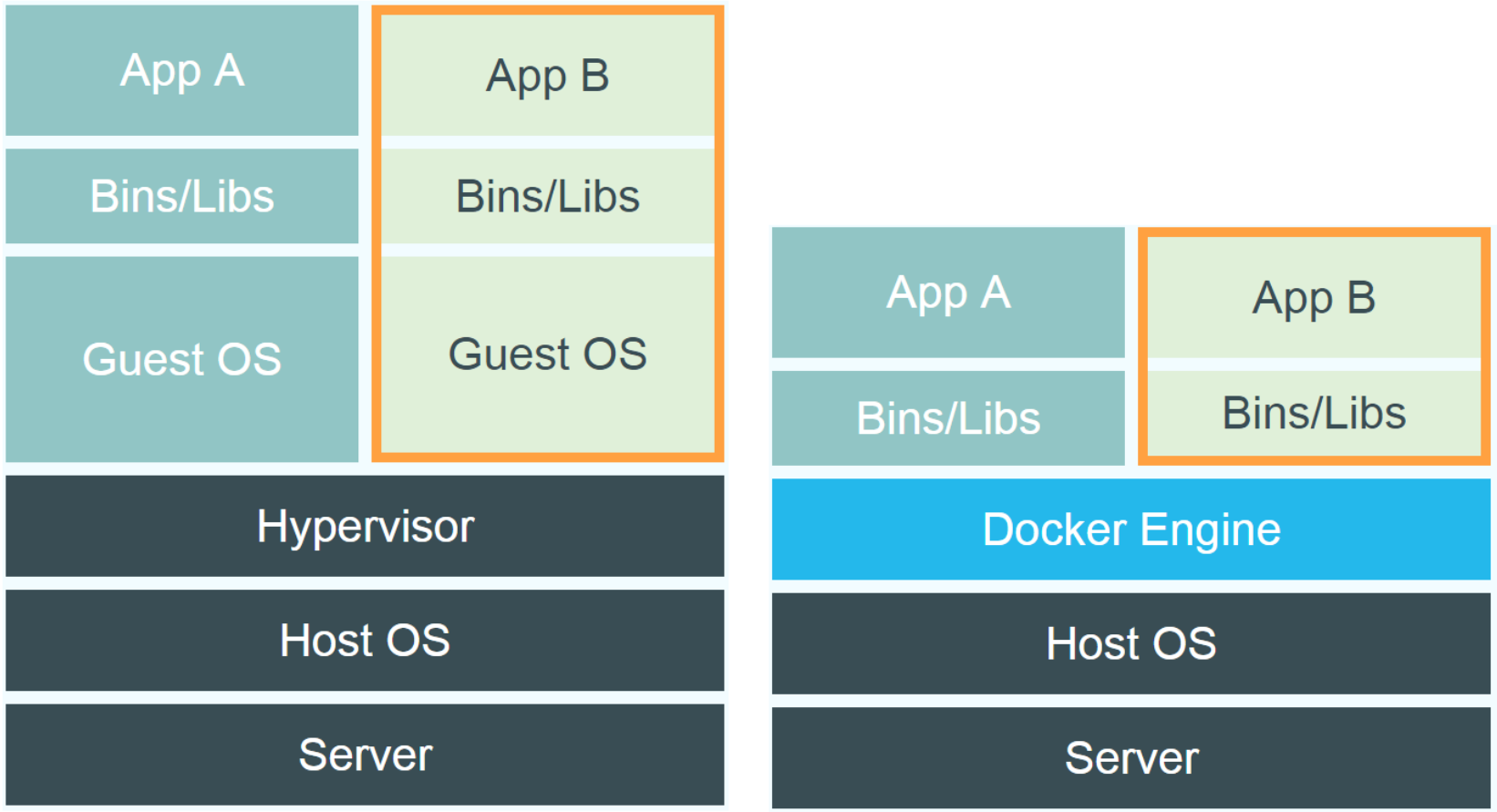
Дополнительно...

- Презентация с конференции разработчиков DevConf2016 "Как подготовить микросервис к продакшену" (Файлик devconf2016.pdf)

Идеальный вариант применения - Docker

- **Docker** — программное обеспечение для автоматизации развёртывания и управления приложениями в среде виртуализации на уровне операционной системы (ru.wikipedia.org/wiki/Docker (https://ru.wikipedia.org/wiki/Docker))
- Свободное программное обеспечение
- **Написан на языке Golang**

Docker vs. Virtual Machine



Как установить и запустить Docker

www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-16-04

(https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-16-04)

- Docker работает только на **64-разрядных Linux системах**
- Docker требует версию **ядра Linux 3.10** и выше
- Проверить свою систему можно командой:

```
$ uname -r
3.11.0-15-generic
```

- Рассмотрим подробно установку для **Ubuntu 16.04**

Установка, продолжение

```
-- Обновляем информацию о доступных пакетах
$ sudo apt-get update

-- Добавляем ключ для официального репозитория Docker
$ sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80
--recv-keys 58118E89F3A912897C070ADBF76221572C52609D

-- Добавляем репозиторий Docker
$ sudo apt-add-repository 'deb https://apt.dockerproject.org/repo ubuntu-xenial main'

-- Еще раз обновляем информацию о пакетах
$ sudo apt-get update

-- Проверяем, что Docker еще не установлен
$ apt-cache policy docker-engine
...
docker-engine:
  Installed: (none)
  Candidate: 1.11.1-0~xenial
...

-- Устанавливаем docker
-- -y, --assume-yes - Автоматический ответ "Да" на вопросы при установке
$ sudo apt-get install -y docker-engine
```

Установка, продолжение

- После всех действий должен быть запущен демон Докера
- Проверить статус демона можно командой:

```
$ sudo systemctl status docker
```

- Проверить работу можно, запустив тестовый образ **hello-world**:

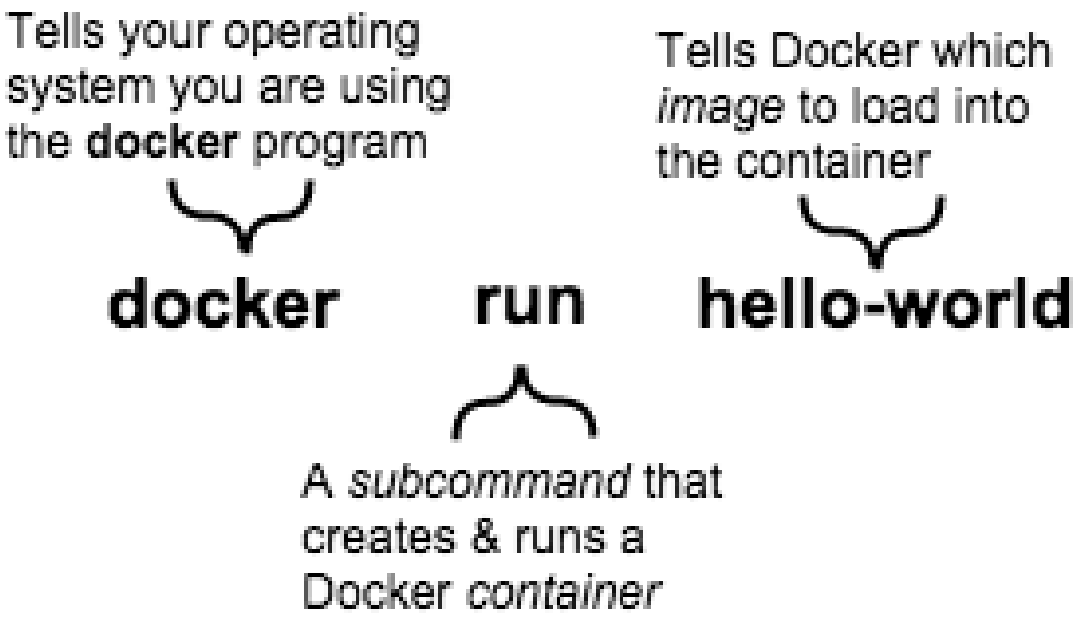
```
$ docker run hello-world
Hello from Docker.
This message shows that your installation appears to be working correctly.
...
```

- Для других версий Ubuntu docs.docker.com/engine/installation/linux/ubuntulinux/

  (https://docs.docker.com/engine/installation/linux/ubuntulinux/)

Образы и контейнеры

- **Образ** - это read-only шаблон, набор файлов и конфигурационных параметров
- **Контейнер** - запущенный образ
- **Простыми словами:** можно запускать много контейнеров на основе одного и того же образа



Структура Dockerfile

- Образы создаются с помощью Dockerfile
- Это текстовый файл с набором команд для создания образа
- docs.docker.com/engine/reference/builder/ (https://docs.docker.com/engine/reference/builder/)
- Некоторая последовательность действий, которую мы бы делали руками, чтобы создать билд приложения

```
FROM scratch # Базовый образ, в данном случае пустой

ADD track-server ./track-server

CMD ["./track-server", "-log_dir", "logs"]
# Параметр -log_dir используется в данном случае библиотекой glog
```

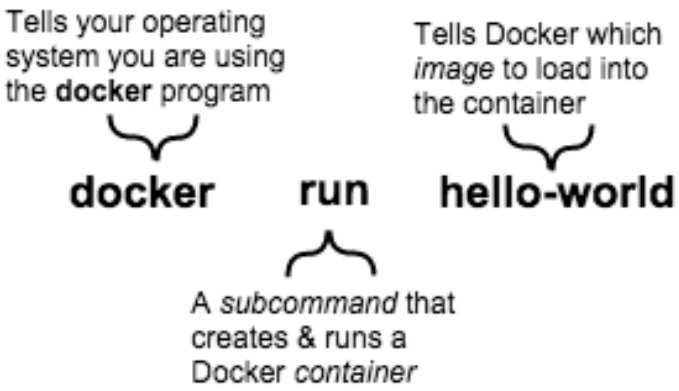
Структура Dockerfile, продолжение

- Какие есть варианты действий:
- RUN
- EXPOSE
- ENV
- ADD
- WORKDIR
- VOLUME

Сборка образа

```
docker build -t app .
```

Запуск образа, параметры запуска



- **-e** (environment), **-v** (volume), **-p** (port), **-d** (detached)

Помимо docker run:

- docker start
- docker stop
- docker rm

Демо - создание контейнера для Docker

Задание

- Устанавливаем **Docker**
- Устанавливаем **Consul**
- Берем **track-server** с Consul конфигурацией
- Запускаем его в **Docker-контейнере**
- Будет необходим проброс портов из контейнера (**EXPOSE**)
- Также будет необходимо настроить **переменную окружения** для Consul внутри docker-контейнера

Thank you

2 декабря 2016

Владимир Ананьев
АО "Программный Регион"
vladimir.ananyev@regium.com (mailto:vladimir.ananyev@regium.com)