Docker

10 October 2015

Евгений Бахтин
SOTAL

Предыстория

Предыстория

- базовый python-пакет платформы собирается в debian-пакет
- конфигурация платформы, конфигурация зависимостей выполняется через ролевые debian-пакеты

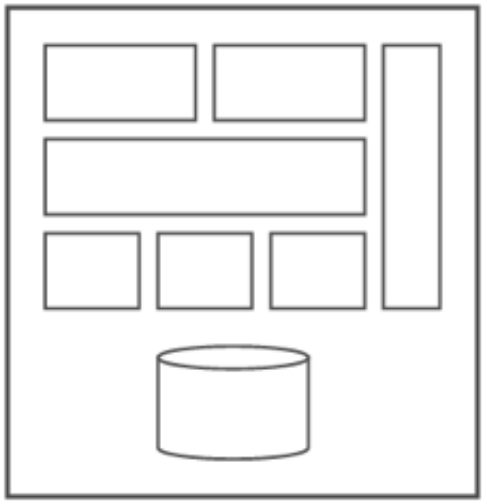
Проблемы сопровождения

- несоответствие dev/test/prod окружения приводит к тому, что какие-то баги попадают в прод
- из-за монолитной SOA-архитектуры становится психологически выкатывать обновления
- ...даже хотфиксы

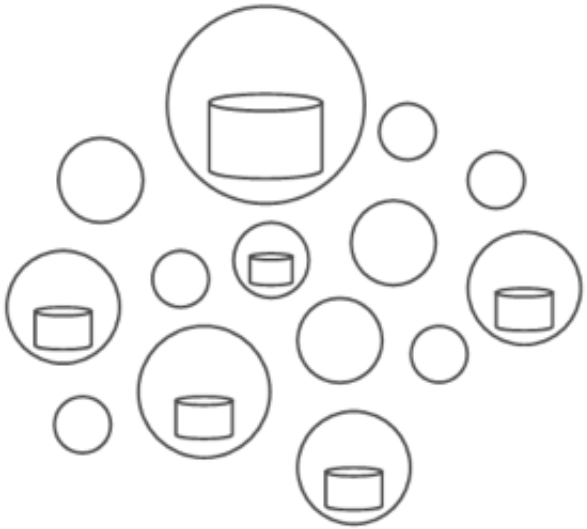
Выводы

В будущем пытаемся:

- свести несоотвестие dev/test/prod к минимуму
- разбивать монолит на маленькие приложения, которые легче тестировать/деплоить по отдельности (миркосервисная архитектура)



Monolithic / Layered



Micro-services

Начинаем новый проект, пытаемся сделать все правильно

Налаживаем CI/CD

- существующий монолит продолжаем собирать как debian-пакет
- новые фичи выделяем в микросервисы которые собираем в debian-пакеты
- для автоматической сборки используем buildbot
- для деплоя используем puppet

Чтобы добавить новое приложение в CI/CD необходимо:

- в репозиторий приложения добавить метаданные для сборки debian-пакета (debian/control and friends)
- написать puppet-модуль описывающий деплой/конфигурацию приложения и зависимостей
- в puppet-манифесте, описывающий состояние инфраструктуры, закрепить приложение за каким-либо хостом
- в конфигурацию публичного reverse-proxy добавить связь с приложением

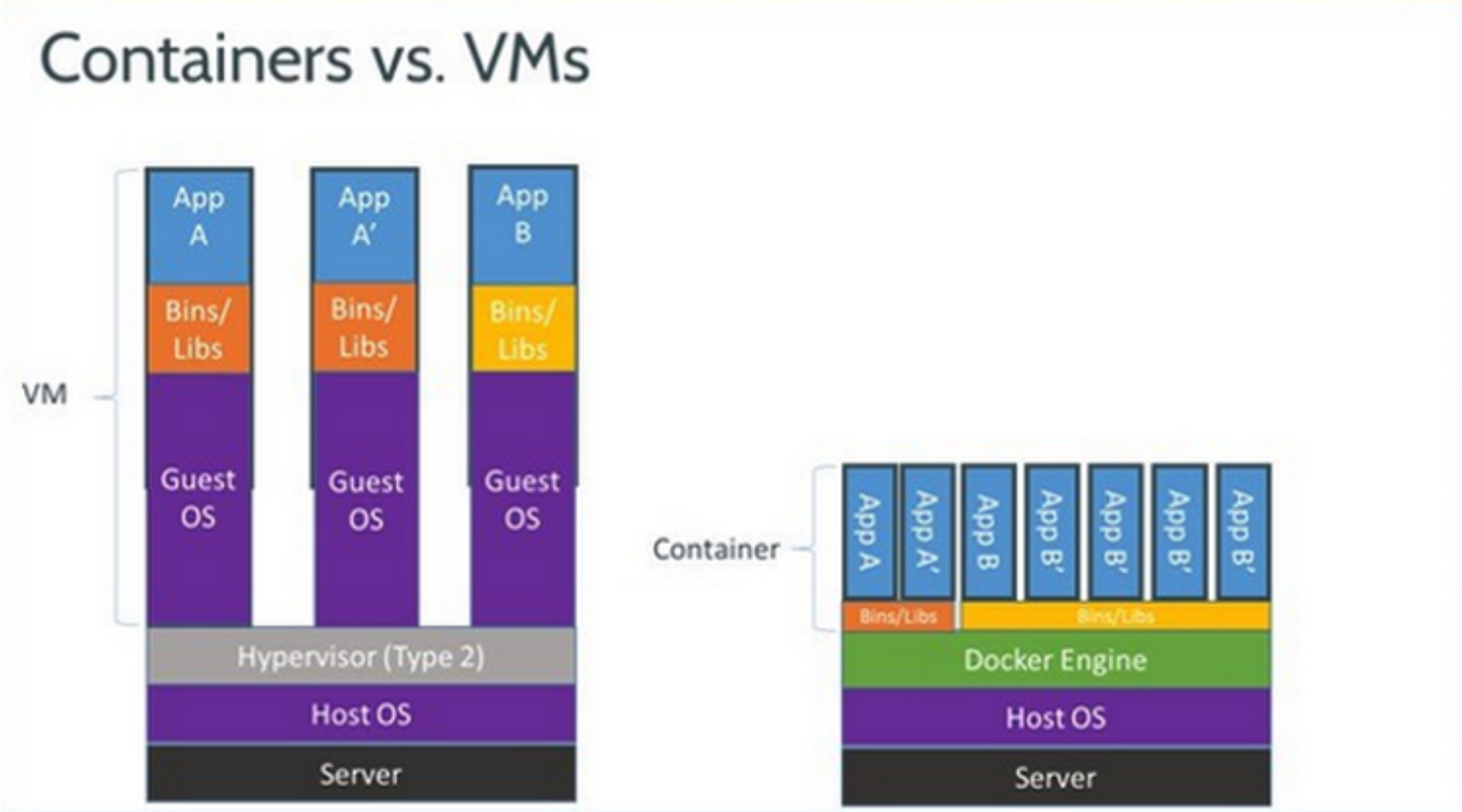
Проблемы подхода

- не решена проблема минимизации разницы между dev/test/prod окружениями
- необходимо следить за совместимостью puppet-модулей
- из-за увеличения количества приложений разработка и поддержка puppet-модулей начинает отнимать много времени

Пробуем Docker

Docker. Что это такое?

- платформа для запуска контейнеров (lightweight VMs)



Как готовить контейнеры?

- контейнеры создаются на основе образов

```
docker run -d nginx:latest
```

- образы создаются на основе Dockerfile - файл с рецептом сборки образа
- образы версионируемы

```
ubuntu:precise
ubuntu:12.04.5
ubuntu:latest
```

Пример Dockerfile

```
# Указываем базовый образ
FROM ubuntu:trusty

# Устанавливаем nginx
RUN apt-get -y install wget
RUN wget -O- http://nginx.org/keys/nginx_signing.key | apt-key add -
RUN echo "deb http://nginx.org/packages/ubuntu/ trusty nginx" | tee -a /etc/apt/sources.list
RUN apt-get update && apt-get -y install nginx
RUN rm -rf /etc/ngix/conf.d/*

# Указываем какую команду выполнить после запуска контейнера
CMD nginx -c /etc/nginx/nginx.conf -g 'daemon off;'

VOLUME ["/var/log/nginx"]

# Говорим что 80 открыт во внешний мир
EXPOSE 80

# Копируем конфигурацию nginx и собранное приложение
COPY nginx.conf /etc/nginx/conf.d/app.conf
COPY dist /var/www
```

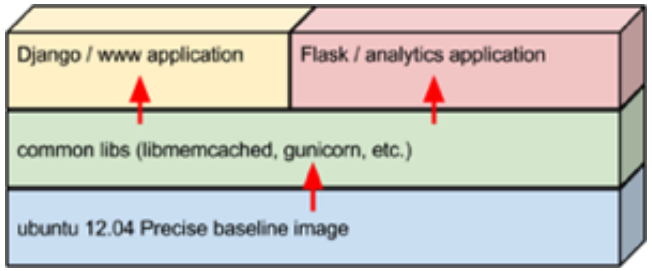
Что нам это дает?

- инкапсуляция приложения и всех его зависимостей
- идентичность dev/test/prod окружения
- единый форм-фактор артефактов деплоя
- окружение для canary deployment с минимумом усилий

Немного деталей

Данные

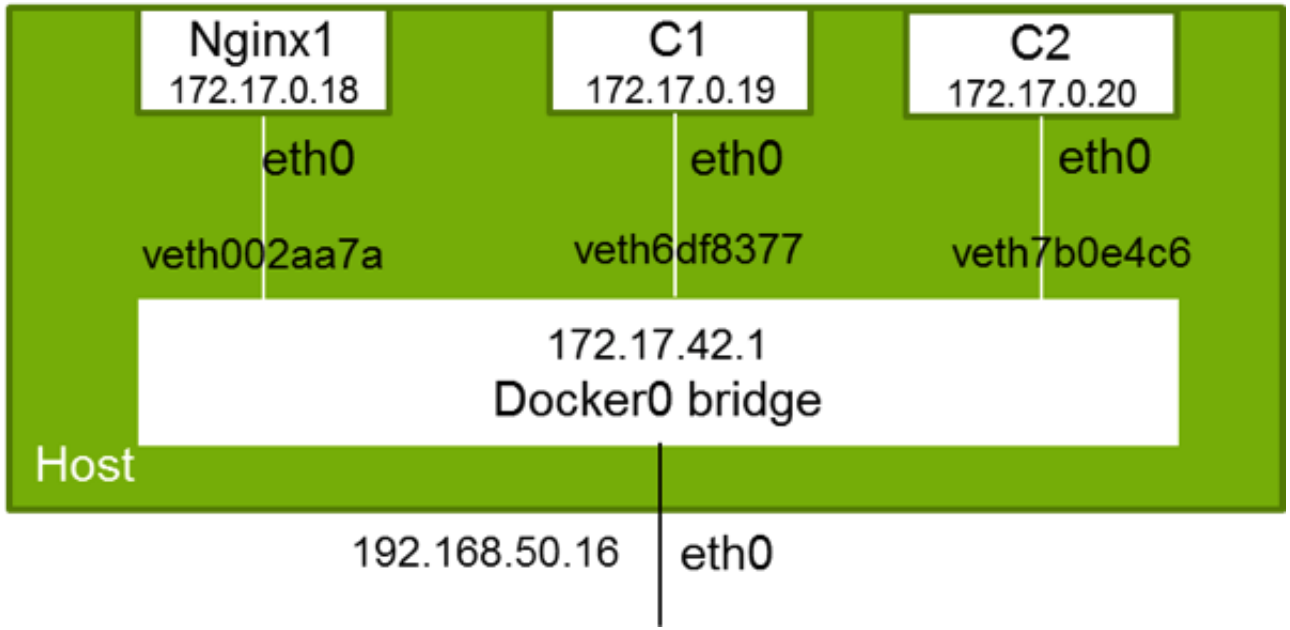
- в качестве корневой файловой системы контейнеры используют union fs



- контейнеры сами по себе эфемерны и не имею состояния - после удаления контейнера (что происходит при каждом обновлении образа контейнера) все данные теряются
- если есть необходимость в персистентном хранилище, используются docker volumes, которые позволяют:
- ... сохранять данные в определенной директории контейнера между запусками контейнера
- ... либо выполнить маппинг какой-либо директории хост-машины внутрь контейнера

Сеть

- контейнеры работают в собственном сетевом пространстве которое задается в настройках docker



- Docker позволяет:
- ... пробрасывать exposed порты на хост-машину
- ... вместо изолированного сетевого окружения использовать окружение хост-машины

Переходим к проблемам

Service Discovery

consul.io

- строго консистентный распределенный KV-storage
- service discovery
- DNS API

```
admin@hashicorp: dig web-frontend.service.consul. ANY
; <<>> DiG 9.8.3-P1 <<>> web-frontend.service.consul. ANY
;; global options: +cmd

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29981
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;web-frontend.service.consul. IN    ANY

;; ANSWER SECTION:
web-frontend.service.consul. 0 IN    A    10.0.3.83
web-frontend.service.consul. 0 IN    A    10.0.1.109
```

Multi-Host Networking

quagga

Open source реализация протоколов динамической маршрутизации (RIP, OSPF, BGP).

Плюсы:

- отсутствие зависимостей

Минусы:

- для обмена маршрутами используется OSPF, требуется multicast
- необходимо вручную заниматься выледением подсетей docker-хостов

Flannel

Агент работающий на каждом Docker-хосте. Выполняет распределение общего сетевого пространства между docker-хостам и организацию связности между контейнерами на разных хостах.

Плюсы:

- минимум конфигурации
- несколько способов организации связности между контейнерами (host-gw, собвественная реализация overlay network, VXLAN, AWS VPC)

Минусы:

- требует etcd

Storage

- docker позволяет использовать только директории на хост-машине в качестве volume'ов
- если контейнер использует volume'ы, он становится привязанным к хост-машине

Логирование

- необходимо налаживать централизованный сбор логов
- ... logstash
- ... fluentd

Заключение

К чему пришли

- в качестве CI используем Jenkins
- при наличии новых коммитов в VCS Jenkins собирает приложение и Docker-образ, который публикует в приватный Docker Registry
- Ansible осуществляет деплой новых контейнеров, обновление существующих
- приложения работают внутри Docker-контейнеров, после запуска регистрируют себя как сервис в consul
- публичный reverse-proxy находит контейнеры приложений через consul по имени сервиса

Thank you

Евгений Бахтин
SOTAL
bahtin@sotal.tv (mailto:bahtin@sotal.tv)