

Контейнеризация

Что это такое ?

- Изоляция процессов
- Ограничение ресурсов
 - CPU
 - RSS
 - I/O
 - Disk usage
- Экосистема образов

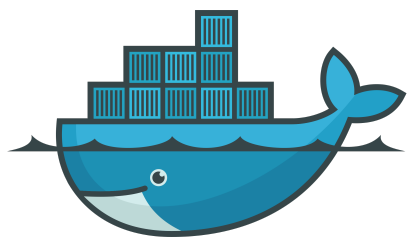
vs Виртуализация

- + Легковеность
- + 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 разворачивание тестовых сред
- (*) Декларативное описание структуры проетка

Системы контейнеризации



docker



LXC



rkt



OpenVZ

Механизмы

namespaces

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

cgroups

Механизм ограничения ресурсов процесса

```
mkdir /sys/fs/cgroup/memory/new_group  
echo 655360 > /sys/fs/cgroup/memory/new1/memory.limit_in_bytes  
echo `pidof project` > /sys/fs/cgroup/memory/new1/tasks
```

Docker

Установка Docker

<https://docs.docker.com/install/linux/docker-ce/ubuntu/>

На 27.10.2018

```
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 -G docker `id -un`
```


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

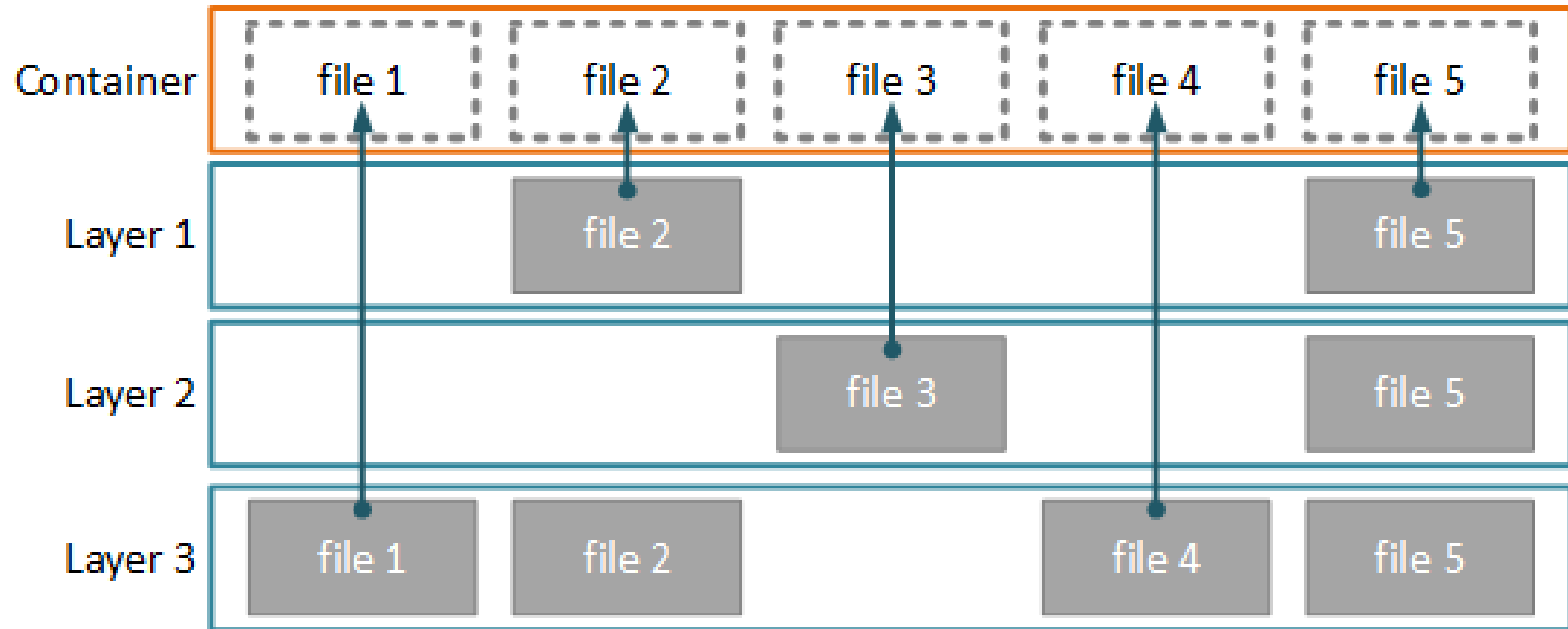
Образ / Image - образец (шаблон) файловой системы для контейнера. Образ содержит все необходимые образу программы и файлы настроек, но не содержит пользовательских данных.

Образы могут наслаиваться 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



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/>

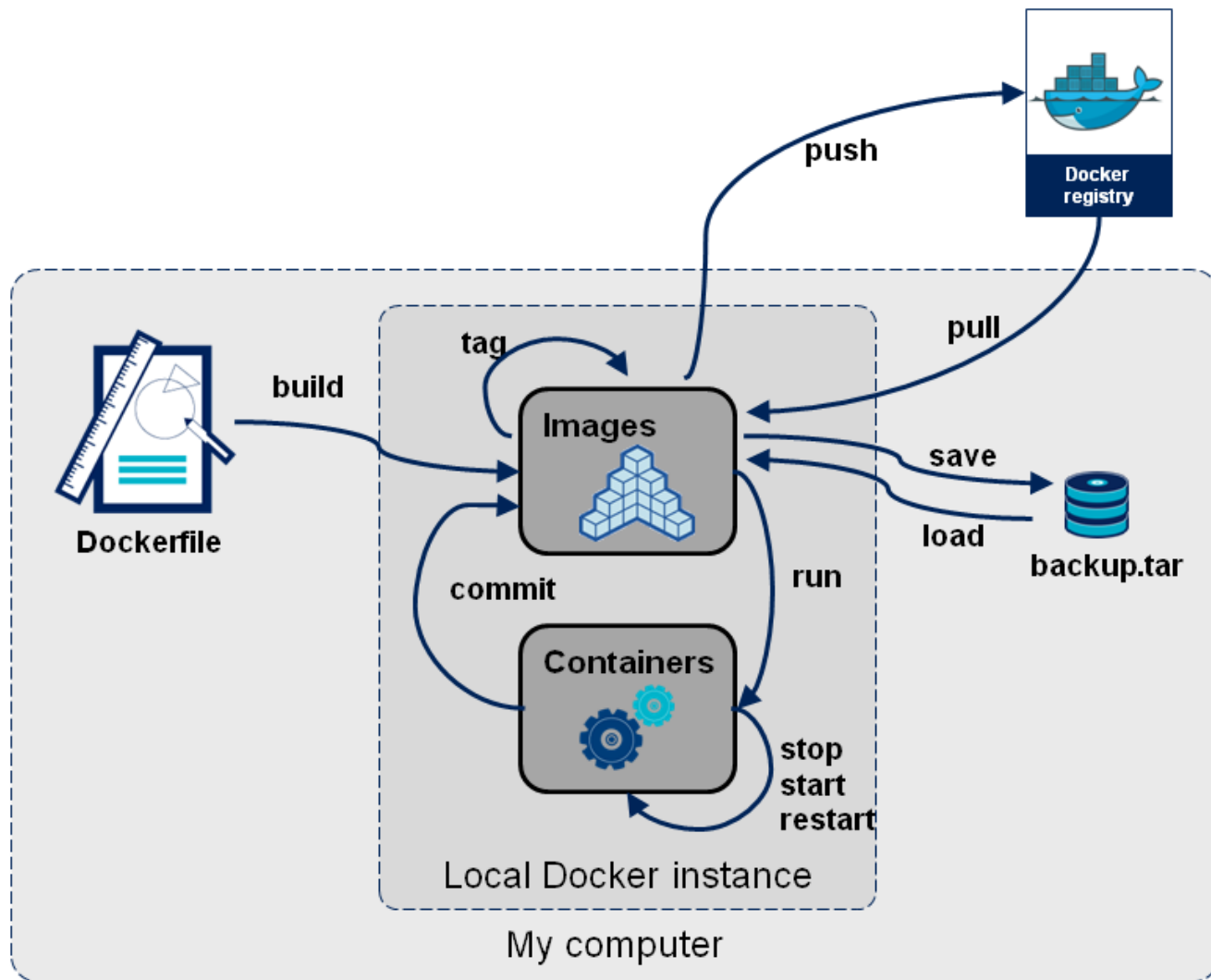
Порты и директории (2)

```
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      # BuildDir
├── ask
├── askme
├── templates
├── static
├── manage.py
├── Dockerfile          # Сборка
├── docker-compose.yml  # Оркестрация
├── .db_data            # Volume для базы
└── requirements.txt
```

Синтаксис Dockerfile

```
FROM ubuntu:16.04
ADD . /app
RUN apt-get update
RUN apt-get install -y python3.5 python3-pip
RUN pip3 install -r /app/requirements.txt
EXPOSE 8000
USER nobody
WORKDIR /app
CMD /usr/local/bin/gunicorn askme.wsgi
```

Синтаксис Dockerfile

- `FROM` - базовый образ
- `ADD` - добавить файлы из сборочной директории
- `RUN` - запустить команду при сборке образа
- `EXPOSE` - информация о том какой порт прослушивается
- `CMD` - команда, которая будет запущена при старте контейнера
- `USER` - пользователь под которым будет запущена `CMD`
- `WORKDIR` - директория в которой будет запущена `CMD`

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:16.04
```

```
...
```

```
VOLUME /app
```

```
...
```

При запуске образа монтируем директорию с проектом

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

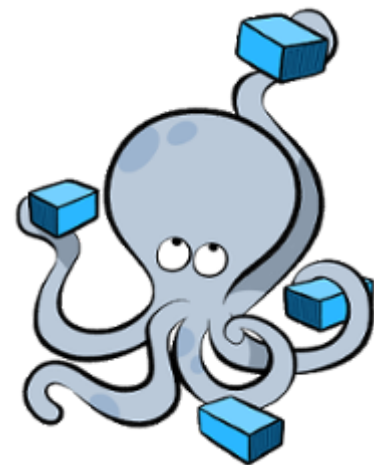
Docker Compose

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 27.10.2018

```
sudo curl -L \  
"https://github.com/docker/compose/releases/download/"\  
"1.22.0/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**

```
version: "2.1"
services:
  db:
    image: postgres:10
    restart: always
    volumes: [ '.db_data:/var/lib/postgresql/data' ]
    environment:
      POSTGRES_DB: dev_db
      POSTGRES_USER: dev_user
      POSTGRES_PASSWORD: dev_pass
  webapp:
    depends_on: [ db ]
    build:
      context: .
      dockerfile: Dockerfile
    image: askme
    volumes: [ './app' ]
    ports: [ '8000:8000' ]
    environment:
      DB_NAME: dev_db
      DB_HOST: db
      DB_USER: dev_user
      DB_PASSWORD: dev_pass
```

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
```

Запуск команд внутри контейнера

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

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

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
```

Использование Makefile

```
make migrate
```