нетология

Docker - обзор

Александр
Ульянцев

АЛЕКСАНДР УЛЬЯНЦЕВ

G-Core labs, Backend Software Engineer

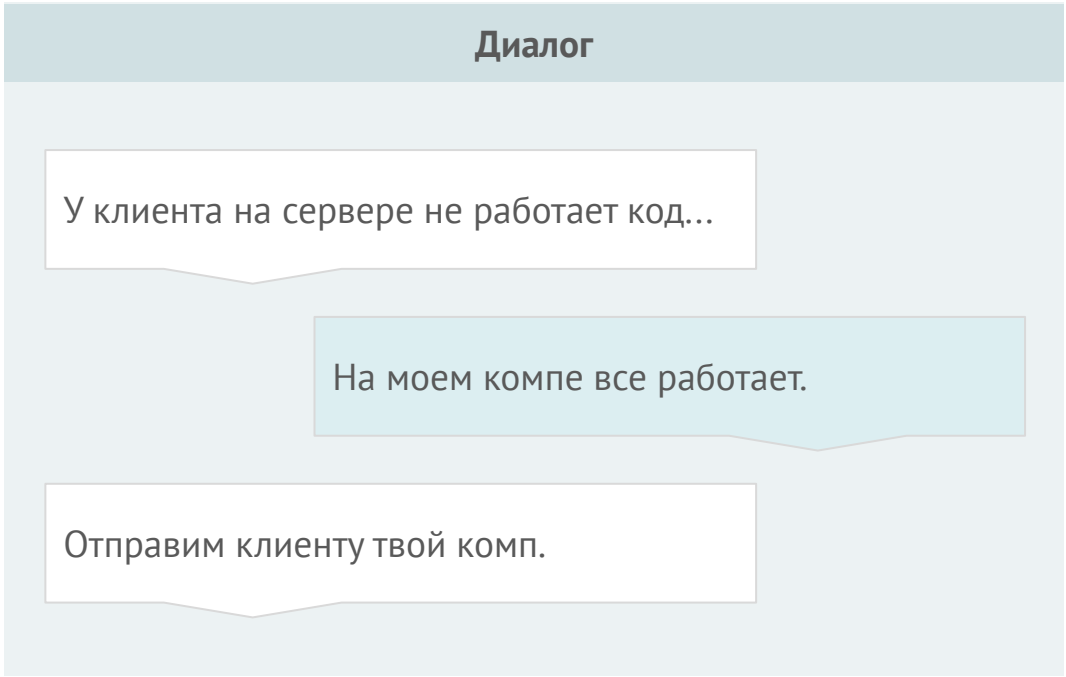
alex@uliantsev.name

@res1d3nt

План занятия

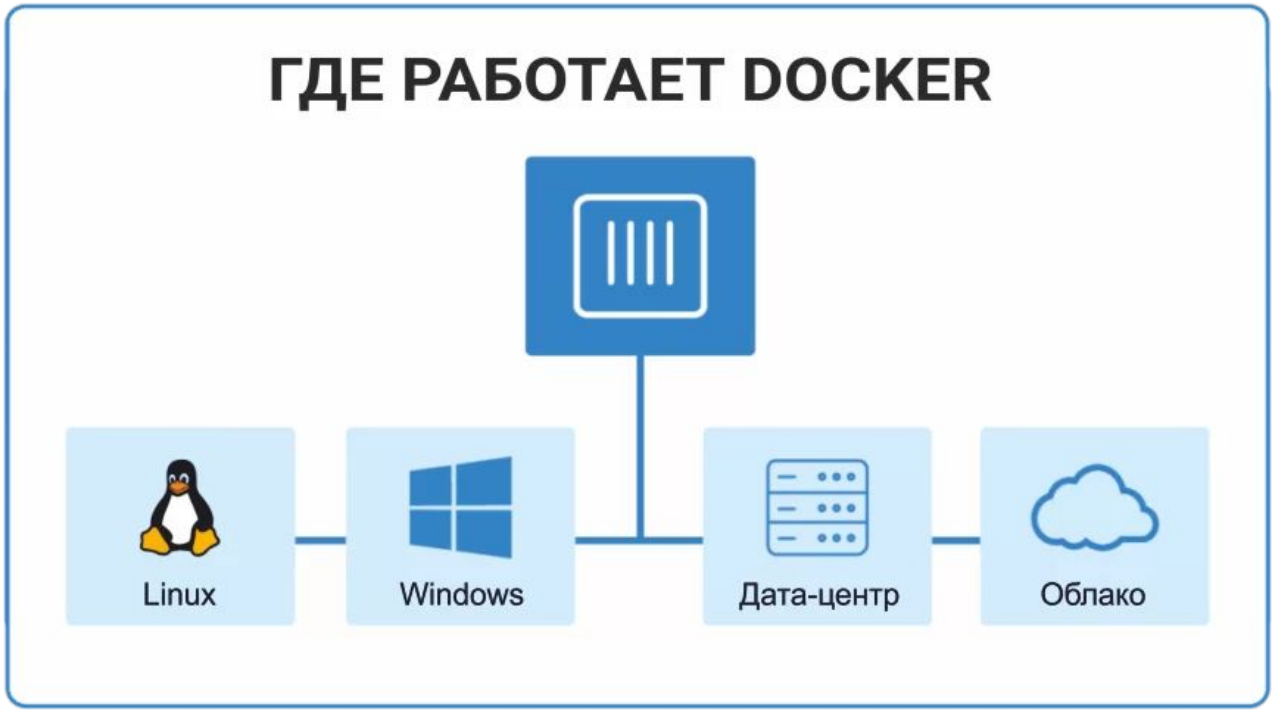
1. Определение
2. Компоненты
    a. Container & Image
    b. Volumes
    c. Network
3. Сборка
4. Dockerfile
5. Запуск
6. Pull - загрузка
7. Push - публикация
8. Домашнее задание

Типичная история



Определение

Docker — комплекс программного обеспечения для виртуализации на уровне операционной системы, что позволяет управлять множеством виртуальных образов в разной среде и без перезагрузки оборудования.



Источник: https://eternalhost.net/blog/razrabotka/chto-takoe-docker

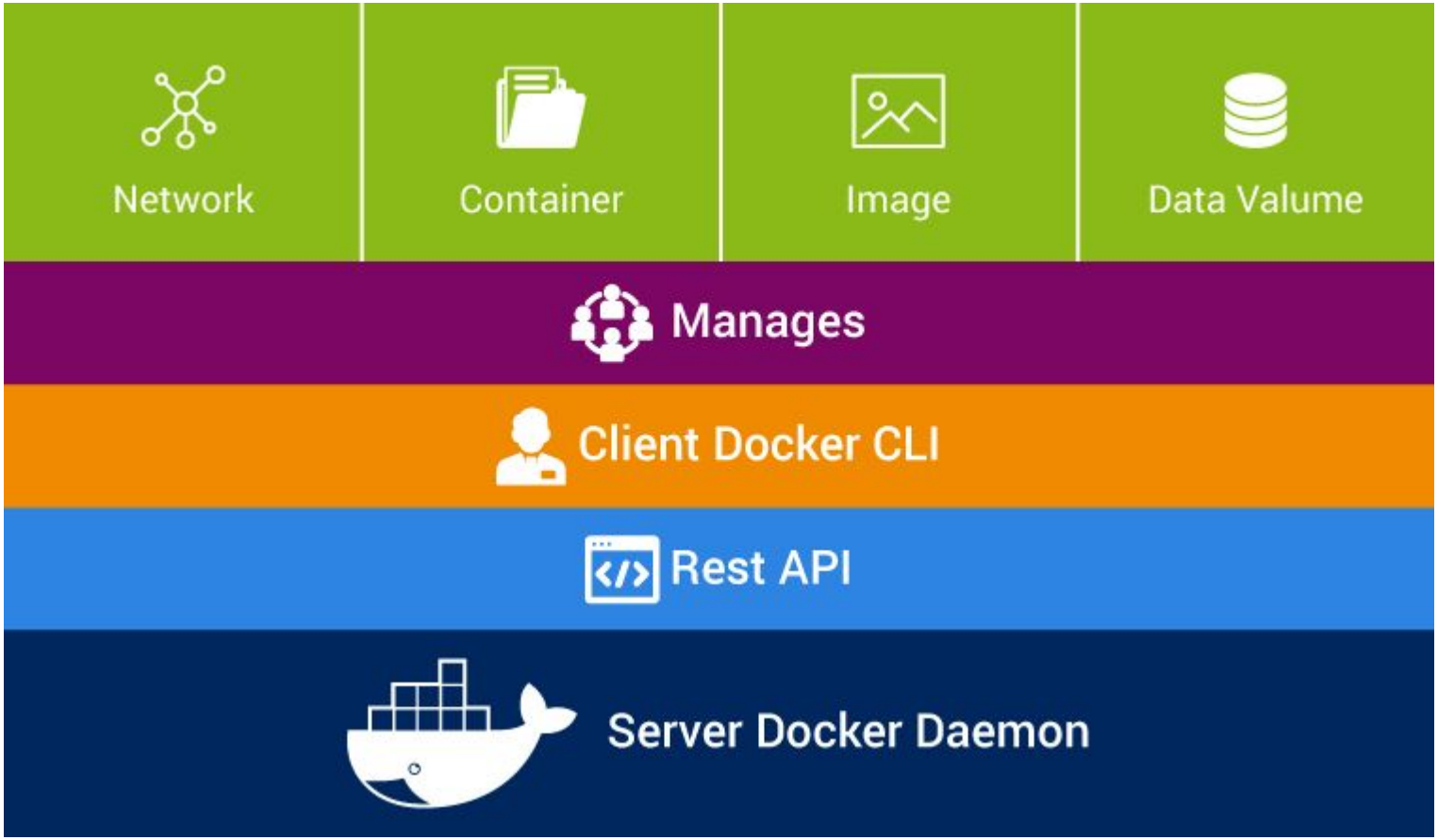
Какие задачи решает

Создание виртуального слепка системы, на которой должна запускаться программа.

В виртуальной среде устанавливаются все зависимости и остается только запустить виртуальный сервер, где всё настроено для запуска сервиса.

- лёгкий старт (не надо писать README по запуску, что объяснять)
- пакеты в системе
- различные версии питона / nodejs
- локальный конфиг для разного окружения prod/stage/dev
- стандартизация работы
- контролируемая изоляция от локальной среды (ключи, токены, env переменные)

Основные компоненты



Источник: https://www.xenonstack.com/images/blog/

Container & Image

Container (*контейнер*) — процесс, который выполняется на хосте. Хост может быть локальным или удаленным. Когда оператор выполняет запуск, докер контейнер изолирован, поскольку у него есть собственная файловая система, собственная сеть и собственное изолированное дерево процессов, отдельное от хоста. Запуск контейнера:

```
$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
```

Image (*образ*) — основа для контейнера. Создается из файла конфигурации (Dockerfile) и контекста. Контекстом являются локальные файлы или другие образы. Каждый новый образ может наслаиваться для очередной версии, по аналогии с git иерархией. Сборка образа:

```
$ docker image build [OPTIONS] PATH | URL | -
```

Volumes

Volumes (тома) — выделенный объем памяти для использования в контейнерах.

Тома позволяют:

- объединять один том между разными контейнерами
- архивировать и шифровать содержимое
- подключать внешние источники из облака или внешних носителей
- использовать многократно

Создание тома и монтирование в контейнере:

```
$ docker volume create my-vol
$ docker run -d --name devtest --mount source=myvol2,target=/app nginx:latest
```

docker volume create	создание
docker volume inspect	отобразить подробную информацию
docker volume ls	список томов
docker volume prune	удалить все неиспользуемые локальные тома
docker volume rm	удаление томов

Volumes (bind mount)

По умолчанию любые данные, созданные внутри контейнера, доступны только внутри контейнера и только во время его работы.

Тома можно использовать для обмена файлами между хост-системой и контейнером Docker.

Предположим, вы хотите использовать официальный образ Docker Nginx и сохранить постоянную копию файлов журнала Nginx для последующего анализа.

Тогда после `$ docker run` мы напишем:

переадресация портов
- 5000 - порт хост-машины
- 80 - порт службы внутри контейнера

запуск в фоновом режиме

```
--name=nginx -d -v ~/nginxlogs:/var/log/nginx -p 5000:80 nginx
```

имя контейнера

монтирование тома “bindmount”

название запускаемого контейнера

Networks, сетевой драйвер

Контейнеры можно объединять в сеть и запускать в разных режимах сети в зависимости от выбранного драйвера:

- **bridge** — сетевой драйвер по умолчанию. Каждый контейнер работает изолированно, при необходимости маршруты настраиваются вручную.
- **host** — для автономных контейнеров, убирается изоляция и сеть с хост-машиной общая.
- **overlay** — соединяют несколько демонов Docker вместе и позволяют службам роя взаимодействовать друг с другом. Эта стратегия устраняет необходимость выполнять маршрутизацию на уровне ОС между контейнерами.
- **macvlan** — позволяют назначать MAC-адрес контейнеру, чтобы он отображался как физическое устройство в вашей сети.
- **none** — для этого контейнера отключаются все сети.
- **user-plugins** — можно устанавливать и использовать сторонние сетевые плагины с Docker.

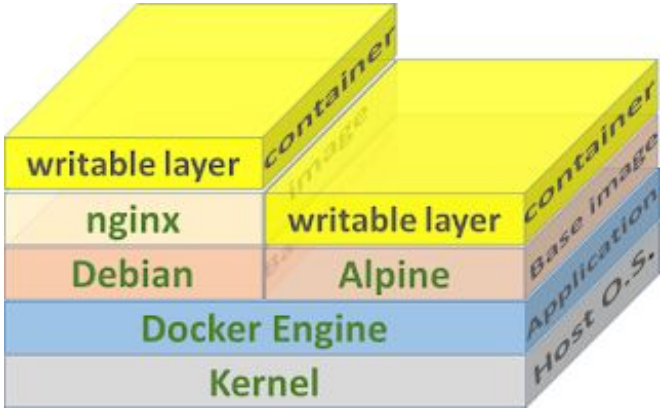
Networks, пример

Объединение двух контейнеров в сеть:

```
$ docker network create --driver=bridge test-net
$ docker run -dit --name alpine1 --network test-net alpine
$ docker run -it --name alpine2 --network test-net alpine
/ #
/ # ping -c 2 alpine1
PING alpine1 (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.057 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.058 ms
```

Основа

Исходным образом может выступать ядро операционной системы Ubintu, Debian, Alpine и пр. *Alpine популярная основа для Docker контейнеров, т.к. её приоритеты это легковестность и безопастность. Размер базовой системы Alpine Linux составляет всего лишь 4-5 Мбайт (исключая ядро).*



Поверх образа с linux можно собрать свой образ с любыми зависимостями для запуска например Python backend сервера.

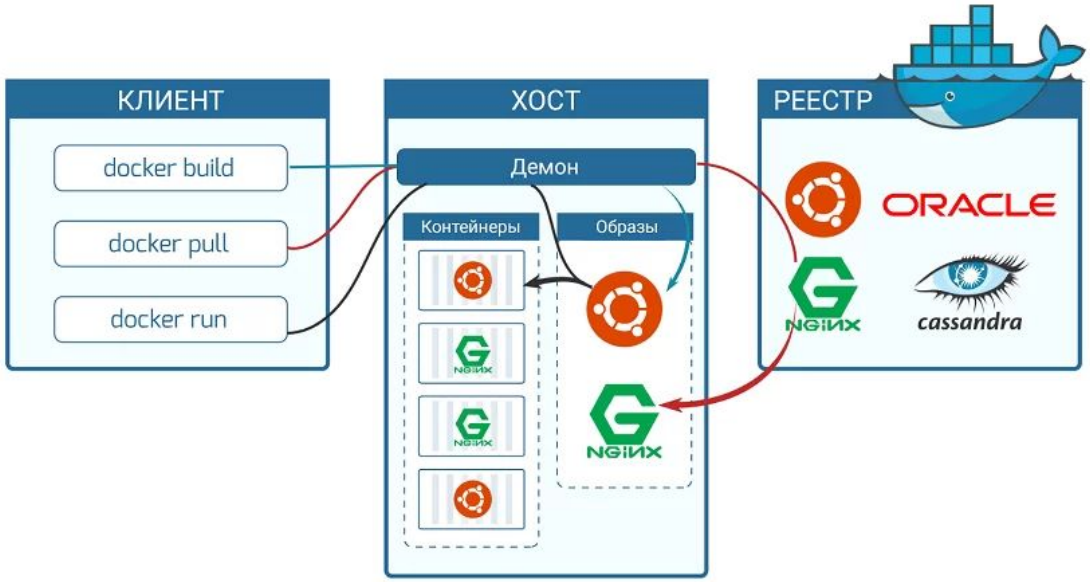
Или собрать образ с предустановленным http сервером (Nginx / Apache).

http://neokobo.blogspot.com/2017/03/container-and-host-os-kernel.html

Процесс сборки

1. Изначальный образ берется из:
   a. Приватного или общедоступного хранилища контейнеров hub.docker.com
   b. Собрать новый контейнер на основе локального linux ядра
2. Целевой образ собирается с настройками и скриптами из Dockerfile

   Новый образ можно опубликовать в репозитории для скачивания.
3. Запуск контейнера



Источник: https://eternalhost.net/blog/razrabotka/chto-takoe-docker

Сборка

```
$ docker build --help
Usage:    docker build [OPTIONS] PATH | URL | -

$ docker build --tag my-server:1.0 .
```

Сборка контейнера из файла Dockerfile в текущем каталоге и пометить образ указанным тегом

Запуск

```
$ docker run --help
Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

$ docker run --publish 8000:8000 --detach --name my-name my-server:1.0
```

- **--publish** или **-p** просит Docker перенаправить трафик на хост с 8000 на 8080 порт. Контейнеры по умолчанию изолированы в своей сети, чтобы сетевой трафик был доступен извне, его необходимо перенаправить по указанному порту.
- **--detach** или **-d** запуск Docker в фоновом режиме, чтобы освободить консоль.
- **--name** или **-n** специальное имя, по которому можно в дальнейшем ссылаться на контейнер. По умолчанию выдается случайное.

Конфигурация Dockerfile

```
# Использовать официальный образ родительского образа / слепка.
FROM python:3.8

# Установка рабочей директории, откуда выполняются команды внутри
контейнера.
WORKDIR /code

# Скопировать все файлы с локальной машины внутрь файловой системы
виртуального образа.
COPY . .

# Запустить команду внутри образа, установка зависимостей.
RUN pip install -r requirements.txt

# Добавить мета-информацию к образу для открытия порта к прослушиванию.
EXPOSE 8000

# Выполнить команду внутри контейнера
CMD gunicorn my_proj.wsgi -b 0.0.0.0:8000
```

Основные команды

- **FROM** (+FROM as) — *контекст из базового образа*
- **ADD** и **COPY** — копирование файлов из хоста в контейнер
- **RUN** — запуск команды внутри образа
- **CMD** — запуск команды с разделением аргументов внутри массива
- **ENTRYPOINT** — запуск команды в контейнере через **run**
- **VOLUME** — настройка томов
- **ARG** — задает переменные, которые пользователь передает сборщику образа docker build с помощью флага --build-arg <varname>=<value>
- **ENV** — замена переменных окружения для контейнера
- **WORKDIR** — устанавливает рабочий каталог для всех инструкций RUN, CMD, ENTRYPOINT, COPY и ADD, которые будут выполнены в Dockerfile.

Pull (загрузка контейнера из реестра)

Шаблон команды:

```
$ docker pull [OPTIONS] NAME[:TAG|@DIGEST]
```

Запуск на примере сервера **nginx** https://hub.docker.com/_/nginx

```
$ docker pull nginx
$ docker run --name some-nginx -d -p 8080:80 nginx
```

В браузере доступен URL http://localhost:8080/, страница **Welcome to nginx!**

Используя простой Dockerfile, можно настроить свою конфигурацию или html

```
FROM nginx:latest
COPY nginx.conf /etc/nginx/nginx.conf
COPY static-html-directory /usr/share/nginx/html
```

```
$ docker build -t my-nginx ./Dockerfile
$ docker run --name my-nginx-server -d -p 8081:80 my-nginx
```

Push (загрузка контейнера в реестр)

Шаблон команды:

```
$ docker push [OPTIONS] NAME[:TAG]
```

Пометим текущий образ нужным тэгом для удобства и проверим его в списке:

```
$ docker tag my-nginx my-nginx:develop
$ docker images
REPOSITORY    TAG    IMAGE ID     CREATED          SIZE
my-nginx    develop    b87668f7655e 13 minutes ago  133MB
my-nginx     latest    b87668f7655e 13 minutes ago  133MB
```

Авторизуемся и публикуем контейнер на docker-hub:

```
$ docker login
Username: my-login
Password:
Login Succeeded
$ docker push my-nginx:develop
```

Можно загружать и запускать свой контейнер на другом хосте.

Итоги

- Познакомились со структурой Docker контейнеров
- Разобрали создание конфигурации Docker контейнеров
- Научились выгружать и загружать Docker образы на внешний репозиторий

Домашнее задание

Давайте посмотрим ваше домашнее задание.

- Вопросы по домашней работе задаём в чате Slack!
- Задачи можно сдавать по частям.
- Зачёт по домашней работе проставляется после того, как приняты **все задачи**.

нетология

Задавайте вопросы и пишите отзыв о лекции!

Александр Ульянцев

Aleksandr Uliantsev