

Технология контейнеризации. Введение в Docker



Устанавливаем Docker

- Docker – 17.06+
- docker-compose – 1.14+
- docker-machine – 0.12.0+

Устанавливаем Docker

- Ubuntu Linux: <https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/>
- Mac OS: <https://download.docker.com/mac/stable/Docker.dmg>
- Windows: <https://download.docker.com/win/stable/Docker%20for%20Windows%20Installer.exe>

Устанавливаем Docker

- `docker version` – версии `docker client` и `server`
- `docker info` – информация о текущем состоянии `docker daemon`

Устанавливаем Docker

Все ок:

```
> docker version
```

Client:

Version: 17.06.2-ce

API version: 1.30

Go version: go1.8.3

Git commit: cec0b72

Built: Tue Sep 5 20:12:06 2017

OS/Arch: darwin/amd64

Server:

Version: 17.06.2-ce

API version: 1.30 (minimum version 1.12)

Go version: go1.8.3

Git commit: cec0b72

Built: Tue Sep 5 19:59:19 2017

OS/Arch: linux/amd64

Experimental: true

Устанавливаем Docker

Docker daemon не запустился:

```
> docker version
```

```
Client:
```

```
Version: 1.13.1
```

```
API version: 1.26
```

```
Go version: go1.7.5
```

```
Git commit: 092cba3
```

```
Built: Wed Feb 8 08:47:51 2017
```

```
OS/Arch: darwin/amd64
```

Cannot connect to the Docker daemon. Is the docker daemon running on this host?

Проверьте, что docker daemon запущен и обратитесь к преподавателям

Docker hello-world

Запустим первый контейнер

```
> docker run hello-world
```

```
Unable to find image 'hello-world:latest' locally
```

```
latest: Pulling from library/hello-world
```

```
5b0f327be733: Pull complete
```

```
Digest: sha256:b2ba691d8aac9e5ac3644c0788e3d3823f9e97f757f01d2ddc6eb5458df9d801
```

```
Status: Downloaded newer image for hello-world:latest
```

```
Hello from Docker!
```

```
<skip>
```

Что произошло?

- docker client запросил у docker engine запуск container из image hello-world
- docker engine не нашел image hello-world локально и скачал его с Docker Hub
- docker engine создал и запустил container из image hello-world и передал docker client вывод stdout контейнера

Docker ps

Список запущенных контейнеров

```
>docker ps
```

| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES |
|--------------|-------|---------|---------|--------|-------|-------|
|--------------|-------|---------|---------|--------|-------|-------|

Список всех контейнеров

```
>docker ps -a
```

| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES |
|--------------|-------------|----------|---------------|--------------------------|-------|----------------|
| 7113ae217dc6 | hello-world | "/hello" | 2 seconds ago | Exited (0) 1 seconds ago | | tiny_ramanujan |

Docker images

Список сохраненных образов

```
>docker images
```

| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|-------------|--------|--------------|-------------|----------|
| hello-world | latest | 05a3bd381fc2 | 2 weeks ago | 1.848 kB |

Docker run

- Команда run создает и запускает контейнер из image

```
>docker run -it ubuntu:16.04 /bin/bash  
root@8d0234c50f77:/# echo 'Hello world!' > /tmp/file  
root@8d0234c50f77:/# exit
```

```
>docker run -it ubuntu:16.04 /bin/bash  
root@4e727649fb85:/# cat /tmp/file  
cat: /tmp/file: No such file or directory  
root@4e727649fb85:/# exit
```

Docker run

- Docker run каждый раз запускает **новый** контейнер
- Если не указывать флаг `--rm` при запуске `docker run`, то после остановки контейнер вместе с содержимым остается на диске

Docker ps

- Найдем ранее созданный контейнер в котором мы создали /tmp/file
- Это должен быть предпоследний контейнер запущенный из образа ubuntu:16.04

```
>docker ps -a --format "table {{.ID}}\t{{.Image}}\t{{.CreatedAt}}\t{{.Names}}"
```

| CONTAINER ID | IMAGE | CREATED AT | NAMES |
|---------------------|--------------|-------------------------------|----------------------|
| 4e727649fb85 | ubuntu:16.04 | 2016-11-16 16:23:21 +0300 MSK | stupefied_montalcini |
| 8d0234c50f77 | ubuntu:16.04 | 2016-11-16 16:22:39 +0300 MSK | drunk_murdock |
| 7113ae217dc6 | hello-world | 2016-11-16 16:02:51 +0300 MSK | tiny_ramanujan |

Docker start & attach

- start запускает остановленный(уже созданный) контейнер
- attach подсоединяет терминал к созданному контейнеру

```
> docker start 8d0234c50f77 #(or docker start drunk_murdock)
> docker attach 8d0234c50f77 #(or docker attach drunk_murdock)
```

ENTER

```
root@8d0234c50f77:/#
root@8d0234c50f77:/# cat /tmp/file
Hello world!
```

Ctrl + p, Ctrl + q

```
> docker ps
```

| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES |
|--------------|--------------|-------------|---------------|--------------|-------|---------------|
| 8d0234c50f77 | ubuntu:16.04 | "/bin/bash" | 9 minutes ago | Up 9 minutes | | drunk_murdock |

Docker run vs start

- `docker run` = `docker create` + `docker start` + `docker attach`*
- `docker create` используется, когда не нужно стартовать контейнер сразу
- в большинстве случаев используется `docker run`

* при наличии опции `-i`

Docker run

- Через параметры передаются лимиты(cpu/mem/disk), ip, volumes
- -i – запускает контейнер в foreground режиме (docker attach)
- -d – запускает контейнер в background режиме
- -t создает TTY
- `docker run -it ubuntu:16.04 bash`
- `docker run -dt nginx:latest`

Docker exec

- Запускает новый процесс внутри контейнера
- Например, `bash` внутри контейнера с приложением

Docker exec

```
>docker exec -it 8d0234c50f77 bash
```

```
root@8d0234c50f77:/#
```

```
ps axf
```

| PID | TTY | STAT | TIME | COMMAND |
|-----|-----|------|------|-----------|
| 12 | ? | Ss | 0:00 | bash |
| 22 | ? | R+ | 0:00 | _ ps axf |
| 1 | ? | Ss+ | 0:00 | /bin/bash |

Docker history

- Показывает историю создания image
- Показывает все слои образа и какие команды использовались при создании образа
- `docker history <image_id> --no-trunc`

Docker history

>docker history **2d696327ab2e**

| IMAGE | CREATED | CREATED BY | SIZE | COMMENT |
|--------------|------------|---|--------|---------|
| 2d696327ab2e | 9 days ago | /bin/sh -c #(nop) CMD ["/bin/bash"] | 0B | |
| <missing> | 9 days ago | /bin/sh -c mkdir -p /run/systemd && echo '... | 7B | |
| <missing> | 9 days ago | /bin/sh -c sed -i 's/^#\s*(deb.*universe\... | 2.76kB | |
| <missing> | 9 days ago | /bin/sh -c rm -rf /var/lib/apt/lists/* | 0B | |
| <missing> | 9 days ago | /bin/sh -c set -xe && echo '#!/bin/sh' >... | 745B | |
| <missing> | 9 days ago | /bin/sh -c #(nop) ADD file:5ed435208da6621... | 122MB | |

Docker commit

- Создает image из контейнера
- Контейнер при этом остается запущенным

Docker commit

```
> docker commit 8d0234c50f77 yourname/ubuntu-tmp-file  
sha256:c9b7e0f6b390a8c964bc4af7736e7f1015f4ce8da8648d95d1b88917742c8773
```

```
> docker images
```

| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|--------------------------|--------|---------------------|---------------|-------|
| yourname/ubuntu-tmp-file | latest | c9b7e0f6b390 | 3 seconds ago | 122MB |

Docker inspect



Сравните вывод двух следующих команд

```
>docker inspect 8d0234c50f77 # container id
```

```
>docker inspect c9b7e0f6b390 # image id
```

Docker kill & stop

- kill сразу посылает SIGKILL
- stop посылает SIGTERM, и через 10 секунд(настраивается) посылает SIGKILL
- SIGTERM - сигнал остановки приложения
- SIGKILL - безусловное завершение процесса
- Подробнее про сигналы в Linux

Docker kill

```
>docker ps -q
```

```
8d0234c50f77
```

```
>docker kill $(docker ps -q)
```

```
8d0234c50f77
```

docker system df

- Отображает сколько дискового пространства занято образами, контейнерами и volume'ами
- Отображает сколько из них не используется и возможно удалить

docker system df

```
> docker system df
```

| TYPE | TOTAL | ACTIVE | SIZE | RECLAIMABLE |
|---------------|-------|--------|---------|---------------|
| Images | 7 | 5 | 6.481GB | 347.3MB (5%) |
| Containers | 5 | 1 | 176.3MB | 176.3MB (99%) |
| Local Volumes | 0 | 0 | 0B | 0B |

Docker rm & rmi

- `rm` удаляет **контейнер**, можно добавить флаг `-f`, чтобы удалялся работающий container(будет послан `sigkill`)
- `rmi` удаляет **image**, если от него не зависят запущенные контейнеры

Docker rm & rmi

>docker rm \$(docker ps -a -q) # удалит все незапущенные контейнеры

4e727649fb85

8d0234c50f77

7113ae217dc6

>docker rmi \$(docker images -q)

Untagged: <your-login>/docker-mk-01:first

Untagged: <your-login>/docker-

mk-01@sha256:f733255c83330f7d5897293736ab9b3c3edc7268a501ecc1c6ed8c33646e4b58

Deleted: sha256:c5a8113a357e7c62c42169b2758e42f0d2c48775c568da0202e9f366b120bbe1

mk-01@sha256:adabd21101c54f318d2539c8df746713114c84ee2cc9314492f472774ac31ac3

Deleted: sha256:a26e5bf03f515e83827e7f4c5b54f2644e06613a969395ee2b4d37555f5959b3

Deleted: sha256:47146b5be28e33939eda699625215d5c2e8c5a2510e6c7bf66c9b7a09fd4ba35

<skip>

docker system prune

- Удаляет все остановленные контейнеры, неиспользуемые volume'ы, сети и образа
- --all, -a - удаляет все image'и
- --filter - задает фильтр по образам
- --force, -f - не спрашивать подтверждение
- --volumes - чистить в том числе и volume'ы

Самостоятельное задание



- Разобраться что такое dangling images
- И написать о них @vitkhab в slack'e