

Введение. Экосистема. Архитектура. Жизненный цикл Docker контейнера.



Олег
Букатчук



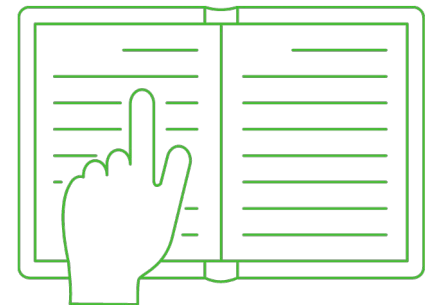
Олег Букатчук

Software Architect DevOps, crif.com

Предисловие

На этом занятии мы:

- Познакомимся с базовыми понятиями в контексте применения Docker.
- Узнаем состав и назначение компонентов экосистемы Docker.
- Рассмотрим базовую архитектуру Docker.
- Рассмотрим пример жизненного цикла Docker-контейнера.
- Научимся упаковывать приложения в Docker!





План занятия

1. [Введение в Docker](#)
2. [Экосистема Docker](#)
3. [Архитектура Docker](#)
4. [Жизненный цикл Docker-контейнера](#)
5. [Собираем первый Docker-контейнер](#)
6. [Итоги](#)
7. [Домашнее задание](#)



Введение в Docker



Контейнер

Контейнер — это способ упаковать приложение и все его зависимости в единый образ. Этот образ запускается в изолированной среде, не влияющей на основную операционную систему.

Контейнеры позволяют отделить приложение от инфраструктуры: разработчики просто создают приложение, упаковывают все зависимости и настройки в единый образ. Затем этот образ можно запускать на других системах, не беспокоясь, что приложение не запустится.



Docker

Docker — это платформа для разработки, доставки и запуска контейнерных приложений.

Docker позволяет:

- создавать контейнеры,
- автоматизировать их запуск и развертывание,
- управлять жизненным циклом,
- запускать множество контейнеров на одной хост-машине.

Контейнеризация похожа на виртуализацию, но это не одно и то же.



Docker

Виртуализация работает как отдельный компьютер со своим виртуальным оборудованием и операционной системой. При этом внутри одной ОС можно запустить другую ОС.

В случае контейнеризации, виртуальная среда запускается прямо из ядра основной операционной системы и не виртуализирует оборудование.

При этом, так как контейнеры не виртуализируют оборудование, они потребляют намного меньше ресурсов.



Преимущества использования Docker

Контейнеры в целом упрощают работу как программистам, так и инженерам, которые развертывают эти приложения.

Docker решает проблемы зависимостей и рабочего окружения.

Контейнеры позволяют упаковать в единый образ приложение и все его зависимости: библиотеки, системные утилиты и файлы конфигурации.



Преимущества использования Docker

Docker упрощает перенос приложения на другую инфраструктуру.

Например, разработчики создают приложение в системе, там все настроено и приложение работает. Когда приложение готово, его нужно перенести в систему тестирования и затем в продуктивную среду. И если в этих системах будет не хватать какой-нибудь зависимости, то приложение не будет работать. В этом случае программистам придется отвлечься от разработки и совместно с командой поддержки разбираться в ситуации.

Контейнеры позволяют избежать такой проблемы, потому что они содержат в себе все необходимое для запуска приложения.



Изоляция и безопасность Docker

Контейнер — это набор процессов, изолированных от основной операционной системы.

Приложения работают только внутри контейнеров, и не имеют доступа к основной операционной системе.

Это повышает безопасность приложений, потому что они не смогут случайно или умышленно навредить основной системе.

Если приложение в контейнере завершится с ошибкой или зависнет, это никак не затронет основную ОС.



Экосистема Docker

Компоненты экосистемы Docker

- **Docker Daemon (Docker демон)** — сервер контейнеров, входящий в состав программных средств Docker. Демон управляет Docker-объектами (сети, хранилища, образы и контейнеры). Демон также может связываться с другими демонами для управления сервисами Docker.
- **Docker Client / CLI (Docker клиент)** — интерфейс взаимодействия пользователя с Docker-демоном. Клиент и Демон — важнейшие компоненты «движка» Докера (Docker Engine). Клиент Docker может взаимодействовать с несколькими демонами.

Компоненты экосистемы Docker

- **Docker Image (Docker образ)** — файл, включающий зависимости, сведения, конфигурацию для дальнейшего развертывания и инициализации контейнера.
- **Dockerfile (Docker файл)** — описание правил (манифест) сборки образа, в котором первая строка указывает на базовый образ. Последующие команды выполняют копирование файлов и установку программ для создания определенной среды разработки со своим набором переменных окружения и прочих параметров.

Компоненты экосистемы Docker

- **Docker Container (Docker контейнер)** — это легкий, автономный исполняемый пакет программного обеспечения, который включает в себя все необходимое для запуска приложения: код, среду выполнения, системные инструменты, системные библиотеки и настройки.
- **Volume (Том хранения данных)** — эмуляция файловой системы для осуществления операций чтения и записи. Она создается автоматически с контейнером, поскольку некоторые приложения осуществляют персистентное хранение данных.

Компоненты экосистемы Docker

- **Docker Registry (Реестр Docker контейнеров)** — резервированный сервер, используемый для хранения docker-образов.
- **Docker Trusted Registry (Доверенный реестр Docker или DTR)** — служба docker-реестра для инсталляции на локальном компьютере или сети компании.
- **Docker Hub (Docker Хаб)** — общедоступный и бесплатный репозиторий, предназначенный для хранения образов с различным программным обеспечением.
Доступен по <https://hub.docker.com>

Компоненты экосистемы Docker

- **Docker Host (Docker хост)** — среда, на которой запускается Docker Engine в виде системного демона для запуска контейнеров с программным обеспечением, указанным в Docker файле.
- **Docker Networks (Docker сети)** — применяются для организации сетевого взаимодействия между приложениями, развернутыми в контейнерах.
Существует несколько режимов работы Docker сети.
Например: **bridge**, **host**, **overlay**, **macvlan** и **none**.

Типы режимов работы сети в Docker

- **Bridge (Мост)** — сетевой драйвер по умолчанию. Если вы не укажете драйвер, этот тип сети, инициализируется автоматически. Мостовые сети обычно используются, когда ваши приложения работают в автономных контейнерах, которым необходимо обмениваться данными.
- **Host (Хост)** — применяется для автономных контейнеров, этот тип сети удаляет слой изоляции между контейнером и Docker хостом и напрямую использует сеть хоста.

Типы режимов работы сети в Docker

- **Overlay (Оверлей)** — оверлейные сети соединяют вместе несколько демонов Docker и позволяют службам Docker Swarm взаимодействовать друг с другом в режиме кластера. Это **общий случай логической сети**, создаваемой поверх другой сети. Узлы оверлейной сети могут быть связаны либо физическим соединением, либо логическим, для которого в основной сети существуют один или несколько соответствующих маршрутов из физических соединений. Эта **стратегия устраняет необходимость выполнять маршрутизацию на уровне ОС** между этими контейнерами.

Типы режимов работы сети в Docker

- **Macvlan** — сети **Macvlan** позволяют назначать **MAC-адрес** контейнеру, чтобы он отображался как физическое устройство в вашей сети. Демон Docker направляет трафик в контейнеры по их MAC-адресам. Использование драйвера macvlan иногда является лучшим выбором при работе с устаревшими приложениями, которые ожидают прямого подключения к физической сети, а не маршрутизации через сетевой стек Docker хоста .

Типы режимов работы сети в Docker

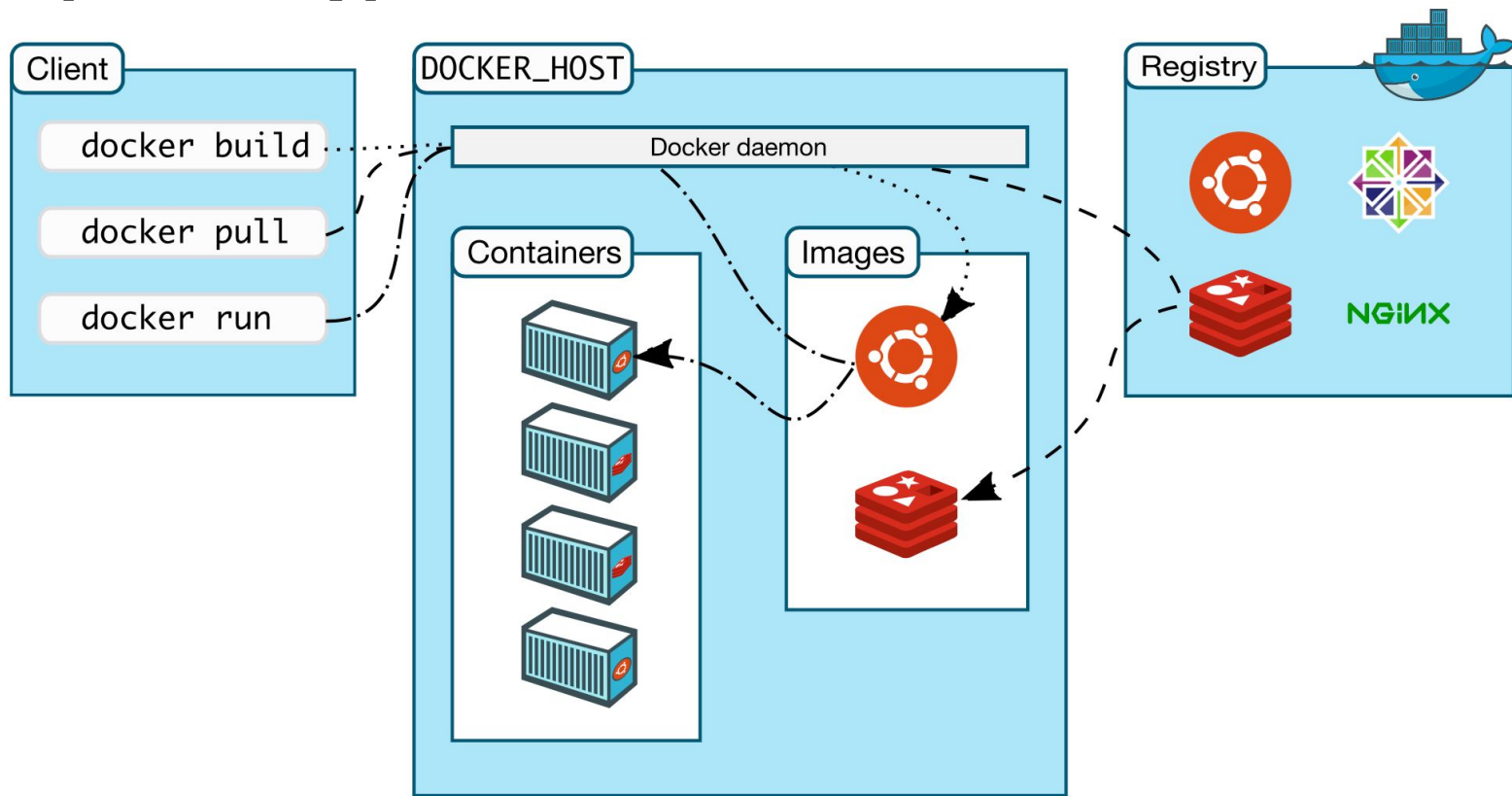
- **None** — в этом режиме работы отключаются все сети. Такой режим работы сети обычно используется вместе с сторонними сетевыми драйверами.

None недоступен для служб Docker Swarm. Вы можете установить и использовать сторонние сетевые плагины с Docker. Эти плагины доступны в Docker Hub или у сторонних поставщиков. Пример плагина: [weave2](#).




Архитектура Docker

Архитектура Docker



Подробнее <https://docs.docker.com/get-started/overview>



Жизненный цикл Docker-контейнера

Docker CLI (Command Line Interface)

- **docker pull**

Загрузить образ из Docker реестра на Docker хост.

```
# Загрузка образа Docker контейнера из публичного репозитория hub.docker.com
$ docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
b8dfde127a29: Pull complete
Digest: sha256:61bd3cb6014296e214ff4c6407a5a7e7092dfa8eefdbbec539e133e97f63e09f
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest

$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
hello-world   latest    d1165f221234   6 months ago  13.3kB
```

Docker CLI (Command Line Interface)

- **docker run**

Запуск Docker контейнера из локально реестра на Docker хосте.

```
# Загрузка образа Docker контейнера из публичного репозитория hub.docker.com
$ docker run hello-world
```

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

<https://hub.docker.com/>

For more examples and ideas, visit:

<https://docs.docker.com/get-started/>

Docker CLI (Command Line Interface)

- **docker run [в интерактивном режиме]**

Запуск Docker контейнера из локально реестра на Docker хосте.

```
# Запуск образа ubuntu из публичного репозитория hub.docker.com
$ docker run -it ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
35807b77a593: Pull complete
Digest: sha256:9d6a8699fb5c9c39cf08a0871bd6219f0400981c570894cd8cbea30d3424a31f
Status: Downloaded newer image for ubuntu:latest
$ root@b65a0676dd1d:/# cat /etc/*release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=20.04
DISTRIB_CODENAME=focal
DISTRIB_DESCRIPTION="Ubuntu 20.04.3 LTS"
NAME="Ubuntu"
VERSION="20.04.3 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.3 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
root@b65a0676dd1d:/# exit
```

Docker CLI (Command Line Interface)

- **docker exec**

Запуск команды в запущенном Docker контейнере.

```
# Запуск команды в запущенном контейнере
$ docker run -d nginx
ba2a5dfd1b05dc380765c877722ac56932376abad4d8bb85927ae35bf101bf98

$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
ba2a5dfd1b05   nginx    "/docker-entrypoint...."  4 seconds ago Up 3 seconds  80/tcp       objective_yalow

$ docker exec -it objective_yalow nginx -v
nginx version: nginx/1.21.3
```

Docker CLI (Command Line Interface)

- **docker stop**

Остановка запущенного Docker контейнера.

```
# Остановка запущенного Docker контейнера.
```

```
$ docker stop objective_yalow  
objective_yalow
```

```
$ docker ps
```

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

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
ba2a5dfd1b05	nginx	"/docker-entrypoint...."	12 minutes ago	Exited (0) 4 minutes ago		
objective_yalow						
b65a0676dd1d	ubuntu	"bash"	18 minutes ago	Exited (0) 15 minutes ago		
flamboyant_gates						
c052919f6d04	hello-world	"/hello"	26 minutes ago	Exited (0) 26 minutes ago		
practical_mirzakhani						

Docker CLI (Command Line Interface)

- **docker rm**

Удаление остановленного Docker контейнера.

```
# Удаление остановленного Docker контейнера.
```

```
$ docker rm objective_yalow  
objective_yalow
```

```
$ docker ps
```

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

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
b65a0676dd1d	ubuntu	"bash"	18 minutes ago	Exited (0) 15 minutes ago		flamboyant_gates
c052919f6d04	hello-world	"/hello"	26 minutes ago	Exited (0) 26 minutes ago		practical_mirzakhani

Docker CLI (Command Line Interface)

- **docker rmi**

Удаление образа Docker контейнера.

```
# Удаление образа Docker контейнера.  
$ docker rmi nginx  
Untagged: nginx:latest  
Untagged: nginx@sha256:853b221d3341add7aaadf5f81dd088ea943ab9c918766e295321294b035f3f3e  
Deleted: sha256:ad4c705f24d392b982b2f0747704b1c5162e45674294d5640cca7076eba2865d  
Deleted: sha256:cf45bd1acd3159a35178bfe8a63f910f010990175050ea6c8c333ba3afaf5123  
Deleted: sha256:a9e7419d7f7c4fe55c85ce08c4f0a8b45abe9b714aa19880f553859797e0332c  
Deleted: sha256:13184aa93ccd585fade03704e048828c29eed86090e7399b208edbe022aaf563  
Deleted: sha256:3161f310d154031dbd57f90c07715335a25a31bcf20a4abf3e040ab86bcac633  
Deleted: sha256:88f95677408c5f02b15064ad1f41a2c74e40e1800cd3536f8fb45b9e6939704b  
Deleted: sha256:d000633a56813933cb0ac5ee3246cf7a4c0205db6290018a169d7cb096581046  
  
$ docker images  
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE  
ubuntu        latest    fb52e22af1b0   2 weeks ago   72.8MB  
hello-world    latest    d1165f221234   6 months ago   13.3kB
```

Docker CLI (Command Line Interface)

- **docker system prune**

Удаление неиспользуемых Docker образов.

```
# Удаление неиспользуемых Docker образов.
$ docker system prune
WARNING! This will remove:
- all stopped containers
- all networks not used by at least one container
- all dangling images
- all dangling build cache

Are you sure you want to continue? [y/N] y
Deleted Containers:
b65a0676dd1d65025c4954d7b20bca108464d3e347225bf3728fcba896e5dbc3
c1fda7c70de824e08c98059ab861e73e8d3138c95632c2c939cfc3006168d281
c0522f9fd4e146844b425246418bf2814072d457aafd151d7f9f3d76d5750412
d824b31993e9f31770a5acbd90061ba435e5eca5b42ce8e216ea5754dbd54e56
c052919f6d049ad559a2d2faac5309f1aa28ac932b8cdfc3258fa532f8bf9fd1

Total reclaimed space: 40B

$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
ubuntu        latest    fb52e22af1b0   2 weeks ago    72.8MB
hello-world    latest    d1165f221234   6 months ago   13.3kB
```


Docker CLI (Command Line Interface)

- `docker system prune -a -f` (ключ `-a` использовать с осторожностью)

Удаление всех Docker образов.

```
# Удаление всех Docker образов.  
$ docker system prune -a -f  
Deleted Images:  
untagged: hello-world:latest  
untagged: hello-world@sha256:61bd3cb6014296e214ff4c6407a5a7e7092dfa8eefdbbec539e133e97f63e09f  
deleted: sha256:d1165f2212346b2bab48cb01c1e39ee8ad1be46b87873d9ca7a4e434980a7726  
deleted: sha256:f22b99068db93900abe17f7f5e09ec775c2826ecfe9db961fea68293744144bd  
untagged: ubuntu:latest  
untagged: ubuntu@sha256:9d6a8699fb5c9c39cf08a0871bd6219f0400981c570894cd8cbea30d3424a31f  
deleted: sha256:fb52e22af1b01869e23e75089c368a1130fa538946d0411d47f964f8b1076180  
deleted: sha256:4942a1abcbfa1c325b1d7ed93d3cf6020f555be706672308a4a4a6b6d631d2e7  
  
Total reclaimed space: 72.79MB  
  
$ docker image ls  
REPOSITORY    TAG          IMAGE ID     CREATED     SIZE
```

Docker Exec vs Docker Run

```
docker exec – yourContainerName bash
```

```
docker run – yourImageName bash
```

Ключевые различия:

- **docker exec** — предназначен для выполнения бинарного файла, отличного от указанного в ENTRYPOINT (если он существует в манифесте образа) в работающем контейнере.
- **docker run** — предварительно скачивает образ (если образ не обнаружен в локальном реестре) и запускает контейнер из образа. Без дополнительных параметров запускается бинарный файл, указанный в качестве точки входа для выполнения.

Вывод диагностической информации Docker контейнера

- **docker logs** — если хотите получить диагностическую информацию из запущенного контейнера.

```
# Диагностика Docker контейнера.
$ docker logs --tail 1 stoic_archimedes
2021/09/19 15:04:15 [notice] 1#1: start worker process 32

$ docker logs -f stoic_archimedes
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2021/09/19 15:04:15 [notice] 1#1: using the "epoll" event method
2021/09/19 15:04:15 [notice] 1#1: nginx/1.21.3
2021/09/19 15:04:15 [notice] 1#1: built by gcc 8.3.0 (Debian 8.3.0-6)
2021/09/19 15:04:15 [notice] 1#1: OS: Linux 5.10.25-linuxkit
2021/09/19 15:04:15 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2021/09/19 15:04:15 [notice] 1#1: start worker processes
2021/09/19 15:04:15 [notice] 1#1: start worker process 31
2021/09/19 15:04:15 [notice] 1#1: start worker process 32
```

Вывод диагностической информации Docker контейнера

- **docker attach** — если необходимо перенаправить поток данных в контейнер.

```
# Перенаправление потока в Docker контейнер.
$ docker run -d nginx
1cff1abd6615d651a5d01a09431ec6a78e040eec6a33b0621a663f7f4fc23584

$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS        NAMES
1cff1abd6615   nginx     "/docker-entrypoint...." 3 seconds ago  Up 2 seconds  80/tcp       hopeful_meitner

$ docker attach hopeful_meitner
^C2021/09/19 15:09:52 [notice] 1#1: signal 2 (SIGINT) received, exiting
2021/09/19 15:09:52 [notice] 33#33: exiting
2021/09/19 15:09:52 [notice] 32#32: exiting
2021/09/19 15:09:52 [notice] 33#33: exit
2021/09/19 15:09:52 [notice] 32#32: exit
2021/09/19 15:09:52 [notice] 1#1: signal 17 (SIGCHLD) received from 32
2021/09/19 15:09:52 [notice] 1#1: worker process 32 exited with code 0
2021/09/19 15:09:52 [notice] 1#1: signal 17 (SIGCHLD) received from 33
2021/09/19 15:09:52 [notice] 1#1: worker process 33 exited with code 0
2021/09/19 15:09:52 [notice] 1#1: exit

$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS        NAMES
```

Сходство с ООП

Сходство с объектно-ориентированным программированием:

- **Образы** концептуально подобны **классам**.
- **Слои** концептуально похожи на **наследование**.
- **Контейнеры** концептуально похожи на **экземпляры** классов.

Для чего и где использовать теги образов?

- **Теги Docker образов схожи с Git-тегами.** Они представляют собой указатель на образ с соответствующим идентификатором.
- **Добавление тега не переименовывает образ,** а исключительно добавляет тег.

Пример: при использовании собственного реестра:

`docker.mycompany.com/jenkins/jenkins-ant:1.10.5` где

`docker.mycompany.com/jenkins/jenkins-ant` — это изначальное имя образа, а `1.10.5` — версия. Чтобы выполнить `docker pull` с указанием тега, нужно указать так:

```
# Загрузка Docker образа из частного реестра.  
$ docker pull docker.mycompany.com/jenkins/jenkins-ant:1.10.5
```

Для чего и где использовать теги образов?

- Образы могут обладать тегами для определения версий или вариантов образа.
- **docker pull ubuntu** будет ссылаться на **ubuntu:latest**.
- Тег **latest** часто является самым последним состоянием (зачастую не стабильным).
- **ubuntu** — это по сути:
library/ubuntu или **index.docker.io/library/ubuntu**

Для чего и где использовать теги образов?

Необходимо использовать теги:

- при использовании образа в продакшн окружении.
- чтобы гарантировать, что одна и та же версия будет использоваться везде (на всех окружениях).
- чтобы получить идемпотентный результат.

Не стоит использовать теги:

- при проведении экспресс-тестирования и прототипирования.
- при проведении экспериментов.
- когда вам нужна последняя версия.

Как корректно применять тег **latest**?

- Убедитесь, что вы задали тег в целом и этот тег корректный.

Если тег не задан, то по умолчанию будет использован тег “latest”.

- Проблема с тегом “latest” — никто не знает, на что он указывает:
 - последнее изменение в репозитории?
 - последний коммит в какой-то ветке? и какой именно?
 - последний созданный git тег в данном репозитории ?
 - какая-то произвольная версия?
- Если вы каждый раз перезаписываете “latest”, то теряете возможность отката на предыдущую версию.
- Теги образов должны иметь осмысленные имена, то есть соответствовать ветвям кода, тегам или хэшам.

Контекст сборки в Docker

- **Контекст сборки** — это рабочая директория, содержащая Dockerfile и дополнительные файлы (части сборки). Зачастую, это текущая директория “.”, передается в команде при сборке docker образа.
- Содержание контекста сборки отправляется (в виде архива) клиентом Docker демону Docker.
- Чем больше дискового пространства занимает директория контекста сборки, тем потенциально дольше займет сборка образа.

Переименование запущенных контейнеров

Вы можете переименовать контейнеры с помощью команды **docker rename**.

Это позволяет "освободить" имя, не удаляя текущий (работающий) Docker контейнер.

```
# Переименование запущенного Docker контейнера.
```

```
$ docker run -d nginx  
f49eb0b8ddac59bc7b34421a86371f788cca733056f08e11bf2a1bac0ad0fe9b
```

```
$ docker ps  
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES  
f49eb0b8ddac   nginx    "/docker-entrypoint...."  4 seconds ago Up 2 seconds  80/tcp       eloquent_cannon
```

```
$ docker rename eloquent_cannon nginx_netology
```

```
18:46:16 @ ~ []
```

```
└─ $ ▶ docker ps
```

```
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES  
f49eb0b8ddac   nginx    "/docker-entrypoint...."  31 seconds ago Up 29 seconds  80/tcp       nginx_netology
```

Docker run -v для сохранения и общего использования данных

С помощью подключение локальной директории из хостовой машины в контейнер можно реализовать:

- Постоянное хранение данных, полученных в результате запущенного в контейнере приложения.
- Совместное использование данных в двух и более запущенных контейнерах.

```
# Подключение локальной директории для использования в Docker контейнере.  
$ docker run -v /local-data:/data-in-container --name container_name -d image-name
```


Создание и использование сети для запущенных контейнеров

Для реализации сетевого соединения между контейнерами:

- Создайте новую виртуальную сеть или используйте существующую.
- Запустите новый контейнер из образа или подключите сеть к уже запущенному контейнеру.

Возможно подключение контейнера(ов) к одной или нескольким сетям.

```
# Переименование запущенного Docker контейнера.  
$ docker network create network-name  
  
$ docker run -d --name=container-name --net=network-name image-name  
  
$ docker network connect network-name container-name
```



Собираем первый Docker-контейнер

Dockerfile

Манифест Docker образа в котором будет выполняться Ansible.

```
# Манифест Docker образа.
FROM alpine:3.14
RUN CARGO_NET_GIT_FETCH_WITH_CLI=1 && \
    apk --no-cache add \
        sudo python3 py3-pip openssl ca-certificates sshpass openssh-client rsync git && \
    apk --no-cache add \
        --virtual build-dependencies python3-dev libffi-dev musl-dev gcc cargo openssl-dev \
        libressl-dev \
        build-base && \
    pip install --upgrade pip wheel && \
    pip install --upgrade cryptography cffi && \
    pip install ansible==2.9.24 && \
    pip install mitogen ansible-lint jmespath && \
    pip install --upgrade pywinrm && \
    apk del build-dependencies && \
    rm -rf /var/cache/apk/* && \
    rm -rf /root/.cache/pip && \
    rm -rf /root/.cargo

RUN mkdir /ansible && \
    mkdir -p /etc/ansible && \
    echo 'localhost' > /etc/ansible/hosts

WORKDIR /ansible

CMD [ "ansible-playbook", "--version" ]
```

Собираем Docker образ

Собираем Docker образ, в котором будет выполняться Ansible.

```
# Сборка Docker образа.
$ cd
/Users/olegbukatchuk/git/netology.ru/virt-homeworks/05-virt-03-docker-usage/src/build/ansible

$ docker build -t olegbukatchuk/ansible:2.9.24 .
... more output strings)))
OK: 98 MiB in 69 packages
Step 3/5 : RUN mkdir /ansible &&      mkdir -p /etc/ansible &&      echo 'localhost' >
/etc/ansible/hosts
---> Running in 05d83f4f0b02
Removing intermediate container 05d83f4f0b02
---> d6bbad65c025
Step 4/5 : WORKDIR /ansible
---> Running in 2d744a795644
Removing intermediate container 2d744a795644
---> 6788cf704b9c
Step 5/5 : CMD [ "ansible-playbook", "--version" ]
---> Running in 81d1f8ad28af
Removing intermediate container 81d1f8ad28af
---> b5878eb55f00
Successfully built b5878eb55f00
Successfully tagged olegbukatchuk/ansible:2.9.24
```

```
# !!! Lifehack: verbose mode !!!
$ DOCKER_BUILDKIT=0 docker build -t olegbukatchuk/ansible:2.9.24 .
```


Загрузка в публичный реестр

Выгружаем **Docker** образ в публичный [реестр](#)

```
# Авторизация в публичном реестре Docker Hub.  
$ docker login -u olegbukatchuk  
Password:  
Login Succeeded  
  
$ docker push olegbukatchuk/ansible:2.9.24  
The push refers to repository [docker.io/olegbukatchuk/ansible]  
444dd64430d4: Pushed  
fb7eb8195ff4: Pushed  
e2eb06d8af82: Mounted from library/alpine  
2.9.24: digest: sha256:01460d9c51dddfef785859c5968e1b33a467a5d5a6d0176dbc2e5b73f5c98fc8e size:  
947
```

Теперь этот **Docker** образ находится в публичный [реестре](#) и доступен для использования всем по [адресу](#).

```
# Загрузка из публичного реестра Docker Hub.  
$ docker push olegbukatchuk/ansible:2.9.24
```



Итоги

Что мы узнали?

- Рассмотрели, компоненты экосистемы Docker;
- Узнали о преимуществах, использования контейнеров;
- Рассмотрели базовый набор Docker CLI для управления жизненным циклом контейнеров;
- Научились загружать, собирать, запускать, переименовывать, удалять контейнеры, а также очищать локальный реестр от ненужных Docker образов;
- Поняли, как можно использовать Docker в контексте Continuous integration, Continuous delivery и Continuous deployment.

Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Telegram.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

⌘ нетология

**Задавайте вопросы и
пишите отзыв о лекции!**

Олег Букатчук