

Виртуализация: **Docker**



Артур

Сагутдинов



Артур Сагутдинов

Инженер департамента
голосовых цифровых
технологий

Banks Soft Systems



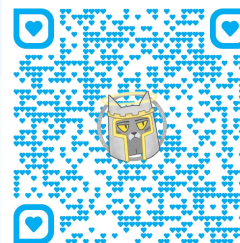
15+ лет в сфере ИТ



Разрабатываю и внедряю
линуксовую инфраструктуру



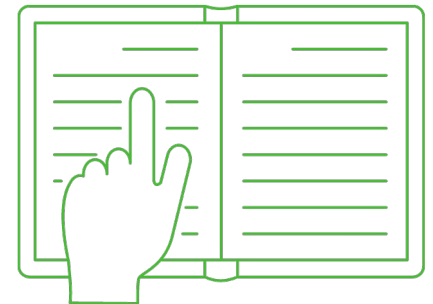
[Сисадминский блог](#)



Предисловие

На этом занятии мы:

- познакомимся с Docker и узнаем, как он работает;
- установим Docker на Debian;
- поговорим о том, что такое Docker Hub, DockerFile, Docker Image и Container;
- запустим тестовый контейнер hello-world;
- научимся управлять образами и контейнерами Docker.





План занятия

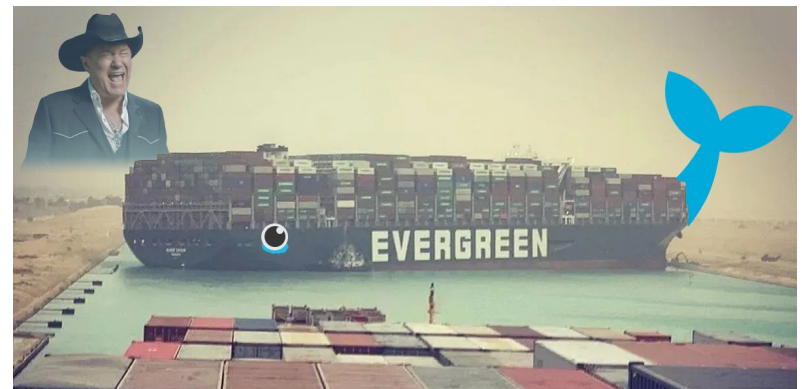
1. [Что такое Docker?](#)
2. [Как работает?](#)
3. [Какие бывают альтернативы?](#)
4. [DockerFile](#)
5. [Docker Image](#)
6. [Docker Container](#)
7. [Docker Hub](#)
8. [Практика](#)
9. [Итоги](#)
10. [Домашнее задание](#)



Что такое Docker?

Система контейнеризации на уровне OS

Docker – стандартизированное ПО для управления контейнерами и образами, которое является оберткой над встроенными в ядро линукс подсистемами `cgroups` и `namespaces`. Они отвечают за контроль над выделением и потреблением ресурсов, а также изоляцию процессов с помощью пространств имён.



Основное назначение Docker

- решает проблему с окружением, необходимым для работы приложения;
- исключает необходимость воссоздания окружения при переносе приложения с одного сервера на другой;
- запускает специальные образы, из которых вырезано почти все, что не требуется для работы приложения.

Версии Docker

Docker можно установить как на Linux, так и на Windows или Mac.

Существуют два варианта поставки:

- **Enterprise Edition** – версия с техподдержкой и различными функциями, которые могут понадобиться крупному энтерпрайзу;
- **Community Edition** – бесплатная версия с функционалом, достаточным как для наших задач, так и для продакшна.

* Мы, конечно же, будем рассматривать реализацию в разрезе Linux




Как работает?

Принципиальная схема

- Клиент через какой-либо интерфейс отправляет команды хосту, на котором стоит Docker демон;
- Демон, получая команды, запускает или останавливает Docker контейнеры из Docker Image;
- Docker Image в свою очередь скачивается или заливается в реестр;
- Реестр может быть частным или поддерживаемым разработчиками (Docker Hub);
- Также Docker Image может быть создан из Dockerfile.





**Какие бывают
альтернативы?**

Альтернативы Docker

Как пример альтернативы Docker, можно рассмотреть LXC или Podman.

Также к списку альтернатив можно добавить такие продукты:

- FreeBSD jail,
- OpenVZ,
- Solaris Containers
- Containerd.



DockerFile

Сценарий воссоздания среды

В классическом методе настройки LAMP сервера, вы:

- открываете мануал,
- читаете, какие команды использовать,
- «бьете» эти команды в консоль, используя, например, мануал от Ubuntu 17.04, а настраивая при этом Ubuntu Server 20,
- вам «валится» куча ошибок,
- ничего не получается,
- и «Да почему эта шайтан машина не работает?!»

В Docker ещё до создания Docker Image, есть DockerFile. Это инструкции, используемые для генерации Docker Image.



DockerFile

Docker Hub хранит в себе превеликое множество готовых Docker Image. Но когда нужно сделать что-то свое, мы прибегаем к **Dockerfile**.

Вместо того, чтобы каждый раз производить настройку вручную, мы один раз пишем сценарий этой настройки. Указывая все, что Docker необходимо сделать, чтобы на выходе получить единственно верный Docker Image.

DockerFile: пример работы с Apache2

Сам файл с инструкциями так и называется: Dockerfile.
Создадим данный файл и вставим в него следующее содержимое:

```
# Используем как основу последний образ Debian
FROM debian:latest
# Указываем создателя имиджа
MAINTAINER Test Netology
# Указываем версию
LABEL version="1.0"
# Указываем команду которая будет выполнена при сборке контейнера
RUN DEBIAN_FRONTEND="noninteractive" apt install -y tzdata && apt
update && apt install -y apache2 nano
# Копируем файл внутрь нашего контейнера
#COPY ./index.html /var/www/html/index.html
# Включаем возможность прокидывать трафик на 80й TCP порт
EXPOSE 80/tcp
# Запускаем апач
CMD apachectl -D FOREGROUND
```

Команда EXPOSE не прокидывает трафик с хоста в контейнер, а только говорит что мы можем так делать.

DockerFile: пример работы с Apache2

Чтобы собрать из этого файла пригодный для работы образ, используем команду:

```
docker build -t netologytest1:1.1 .
```

После сборки найдем ID собранного контейнера с помощью команды:

```
docker image ls
```

DockerFile: пример работы с Apache2

Запустим только что созданный образ с помощью команды:

```
docker run -d -p 80:80 id_созданного_образа
```

Параметр -p говорит Docker Engine, что мы хотим с порта 80 Docker сервера, прокинуть трафик на порт 80 контейнера.

Теперь можно в браузере открыть адрес

http://ip_докерсервера.

Мы увидим стандартную страницу, демонстрирующую, что Apache2 работает.

DockerFile: пример работы с Apache2

Кроме просто установки ПО внутри контейнера, мы можем доставить внутрь контейнера какие-то свои файлы.

Давайте заменим стандартную страницу Apache своим файлом, раскомментировав строку с командой COPY

```
# Используем как основу последний образ Debian
FROM debian:latest
# Указываем создателя имиджа
MAINTAINER Test Netology
# Указываем версию
LABEL version="1.0"
# Указываем команду которая будет выполнена при сборке контейнера
RUN DEBIAN_FRONTEND="noninteractive" apt install -y tzdata && apt update && apt
install -y apache2 nano
# Копируем файл внутрь нашего контейнера
COPY ./index.html /var/www/html/index.html
# Включаем возможность прокидывать трафик на 80й TCP порт
EXPOSE 80/tcp
# Запускаем апач
CMD apachectl -D FOREGROUND
```

DockerFile: пример работы с Apache2

Создадим рядом с файлом Dockerfile файл index.html:

```
nano ./index.html
```

Внутри файла укажем произвольный текст, например Netology Mega Docker Guide, сохраним и запустим сборку образа:

```
docker build -t netologytest1:1.2 .
```

Запустим только что созданный образ с помощью команды:

```
docker run -d -p 80:80 id_созданного_образа
```

Перейдя на адрес http://ip_докерсервера, мы увидим содержимое созданного нами файла index.html

Docker слои

Давайте посмотрим на содержимое любого созданного нами образа. Для этого нужна команда:

```
docker image history id_образа
```

Мы увидим упорядоченные «слои», из которых состоит наш образ.

Docker слои

- Слои позволяют не создавать каждый раз весь образ целиком, а пересоздавать только слой, относящийся к команде, которую мы изменили, и все идущие следом за ним;
- Слои можно рассматривать как «коммиты» в репозитории. Они описывают процесс поэтапной модификации образа, вследствие которой он пришёл к своему текущему состоянию.
- Мы можем взять любой готовый образ и использовать его, указав в строке FROM нашего Dockerfile. Тогда, не внося изменений в базовый образ, Docker поэтапно наложит все наши модификации на него при создании нового образа.



Docker слои

Подход, основанный на наложении слоев, позволяет не тратить время на повторную сборку слоев, которые не были затронуты очередным изменением.

Если же попытаться повторно собрать образ из Dockerfile, в который не было внесено каких-либо изменений, сборка произойдет почти моментально.

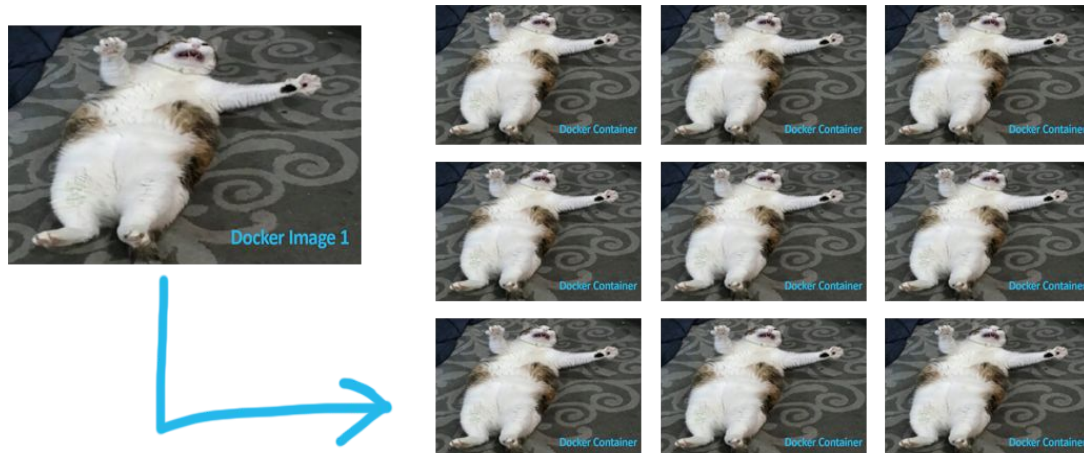


Docker Image

Стандартизированное ПО

Docker Image – это стандартизированный образ.

Хорошим примером будет сравнение с аудиокассетой или диском. Вы покупаете подобный носитель и вне зависимости от проигрывателя, в который вы его вставляете, вы услышите одну и ту же песню. По аналогии с этим: из тысяч Docker серверов, запустивших один и тот же Docker Image, все дадут одинаковый результат.





Docker Container

Docker Container – запущенный Image



Опять же по аналогии, если **Docker Image** – это **кассета**, которую можно вставить в магнитофон.

То **Docker Container** – это вставленная в магнитофон и **играющая кассета**.

10 раз запустив один Docker Image – мы создадим 10 одинаковых Docker контейнеров.

Тот же Docker Image от Ubuntu 18 весит меньше 100мб.

Изоляция Docker Container

Любой контейнер изолирован от других контейнеров и от хоста, на котором он запущен. **Никакое действие** или бездействие внутри контейнера **не повлияют на другие контейнеры** или Docker сервер. Можно удалить, например, пакет php7 из одного контейнера, но это никак не повлияет на другие контейнеры, в которых этот пакет стоит.

pWK3bZOxA



Docker Hub

Docker Hub

Публичное облачное хранилище, куда можно загрузить свои Docker Image для последующей работы с ними.

Поддерживается самими разработчиками Docker.

Репозиторий в Docker Hub может быть как публичным, так и приватным.





Практика

Docker Engine – поддерживаемые ОС

Docker Engine – основа системы. Docker предоставляет .deb и .rpm пакеты для следующих систем:

Платформа	x86_64/amd64	ARM	ARM64/AARCH64
CentOS	v		v
Debian	v	v	v
Fedora	v		v
Raspbian		v	v
Ubuntu	v	v	v

PRKB1E

* Мы, конечно же, будем рассматривать установку на Debian.

Установка Docker Engine на Debian

Устанавливаем пакеты для работы apt через HTTPS

```
# Обновляем кеш
sudo apt update
# Устанавливаем необходимые пакеты
sudo apt install apt-transport-https ca-certificates curl gnupg lsb-release
```

Добавляем GPG ключ

```
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o \
    /usr/share/keyrings/docker-archive-keyring.gpg
```

Установка Docker Engine на Debian

Добавляем stable репозиторий для x86_64 / amd64

```
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
\      https://download.docker.com/linux/debian $(lsb_release -cs) stable" | sudo
\      tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Устанавливаем Docker Engine

```
sudo apt update
sudo apt install docker-ce docker-ce-cli containerd.io
```

Запускаем Docker

```
sudo systemctl enable docker
sudo systemctl start docker
```

Запускаем классический Hello World

Первое, с чего можно начать – это Hello World. Для этого используем следующую команду для запуска образа hello-world:

```
docker run hello-world
```

Docker, не обнаружив образ hello-world локально, подключится к Docker Hub, скачает его и запустит контейнер. Об этом говорят характерные строки в выводе:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
b8dfde127a29: Pull complete
Digest: sha256:f2266cbfc127c960fd30e76b7c792dc23b588c0db76233517e1891a4e357d519
Status: Downloaded newer image for hello-world:latest
```

Запускаем классический Hello World

В результате `m7RwMqyF` увидим текст:
Hello from Docker!

И дополнительную информацию, которая разъясняет, что произошло и дает ссылки на документацию.

Обновление контейнера

Например, если свой контейнер вы взяли из репозитория, hello-world, то при выходе обновления вам понадобится команда pull.

```
docker pull hello-world
```



Смотрим список локальных Docker Images

По аналогии с командой `ls` (которая отображает содержимое каталога), чтобы посмотреть хранимые локально Docker Images, можно использовать следующую команду:

```
docker images
```

Результат будет представлен в виде текстовой таблицы

vSPC3

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hello-world	latest	d1165f221234	2 months ago	13.3kB



Смотрим список Docker Container'ов

Узнать список работающих контейнеров можно командой:

```
docker container ls
```



Смотрим логи Docker Container'ов

Узнав ID контейнера, можно посмотреть его логи с помощью команды:

```
docker logs id_контейнера
```


Смотрим список Docker Container'ов

CONTAINER ID PORTS NAMES	IMAGE	COMMAND	CREATED	STATUS
ca424f568436 ago	hello-world confident_bouman	"/hello"	4 seconds ago	Exited (0) 3 seconds
d289de3f4720 ago	hello-world great_montalcini	"/hello"	6 seconds ago	Exited (0) 5 seconds
78768d88fb52 ago	hello-world laughing_chandra	"/hello"	30 minutes ago	Exited (0) 30 minutes
d5cd52db2068 ago	hello-world bold_taussig	"/hello"	31 minutes ago	Exited (0) 31 minutes

- CONTAINER ID – Идентификатор контейнера,
- IMAGE – Образ, использованный при создании контейнера,
- COMMAND – Запущенная команда,
- CREATED – Время, прошедшее с момента запуска,
- STATUS – Статус контейнера, работает ли или завершил работу,
- PORTS – Проброшенные в контейнер порты,
- NAMES – Случайно сгенерированное имя

Docker run VS Docker start

Важно упомянуть про разницу между run и start

```
docker run имя_образа
```

Находит\скачивает Image, создает из него контейнер и запускает созданный контейнер. 10 раз запустив эту команду с именем какого-то образа, вы создадите 10 контейнеров.

```
docker start id_контейнера (или его имя)
```

Запускает уже существующий, но остановленный контейнер. Таким образом, вы можете запустить контейнер, произвести в нем какие-то настройки, остановить с помощью команды `docker stop` и потом запустить, не потеряв произведенные настройки.

Удаляем лишние образы

Если необходимо удалить какой-либо образ, чтобы освободить место, необходимо сперва удалить все созданные на его основе контейнеры. Удалим, к примеру, hello-world. Сначала смотрим имеющиеся контейнеры:

```
docker container ls -a
```

Смотрим на id или имена и с их помощью удаляем контейнеры:

```
docker container rm id\имя_контейнера
```

Когда все контейнеры, использующие образ удалены, можно удалить сам образ

```
docker image rm hello-world
```



Итоги

Итоги

Сегодня мы рассмотрели Docker для Linux и узнали как:

- установить Docker на Debian;
- что такое Docker Hub, DockerFile, Docker Image и Container;
- запустить тестовый контейнер hello-world;
- управлять образами и контейнерами Docker.





Домашнее задание

Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера .
- Задачу можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты задача полностью**.

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

Артур Сагутдинов

