



Основы Docker | Docker и Python

[Виртуализация и контейнерная виртуализация \(контейнеризация\)](#)

[Отличие контейнера от виртуальной машины](#)

[Что такое Docker](#)

[Чуть подробнее про Образы и контейнеры \(Images и Container\)](#)

[Dockerfile](#)

[Docker Compose](#)

[Docker Swarm](#)

[Kubernetes](#)

[Установка Docker](#)

[Установка Docker для ОС Linux \(Ubuntu 20.04\)](#)

[Установка Docker для ОС Windows 10 \(на основе WSL\)](#)

[Основы работы с Docker](#)

[Практика 1. Установка Nextcloud](#)

[Загрузка образов Docker в репозиторий Docker \(Docker Hub\)](#)

[Практика 2: Создание Docker образа с приложением на Python](#)

[Создание Flask приложения](#)

[Упаковка Flask приложения в контейнер Dockerfile](#)

[Создание образа контейнера на основе Dockerfile](#)

[Полезные ресурсы](#)

[Заключение](#)

[Список использованных источников](#)

Представим, что мы разрабатываем приложение на **Python**. Чтобы наше приложение запускалось на других серверах (компьютерах), мы должны на каждый компьютер установить:

- интерпретатор Python;
- все необходимые библиотеки (при этом не напутав с версиями, что иногда бывает критично);
- другие зависимые “программы”, такие как СУБД, пуллер соединений или брокер сообщений;
- создать переменные окружения, настроить конфигурацию и т.д.

Всё это делать придётся вручную и хорошо, если только на одном компьютере. А если на 20? А если на 200? Чтобы не делать это самостоятельно, мы просто создадим готовый образ (сборку) на основе которого развернём специальный контейнер, где уже всё установлено, настроено и готово к работе.

Звучит как решение. Но как это сделать и что вообще такое контейнер? Давайте разбираться.

Виртуализация и контейнерная виртуализация (контейнеризация)

Прежде чем говорить о том, что такое **контейнеры и контейнеризация**, нужно разобраться в том, что такое **виртуализация**. Без объяснения этого понятия логика работы контейнеров будет непонятна.

Виртуализация — это технология, с помощью которой **на одном физическом устройстве** (компьютере или сервере) можно создать **несколько виртуальных компьютеров**. Иначе говоря, на компьютере с одной операционной системой можно запустить несколько других операционных систем или приложений.

ОС запускаются в **виртуальной среде**, что позволяет им работать на одном устройстве **изолированно** друг от друга. Операционную систему компьютера, на котором работает виртуальная среда, называют **хост-системой**, а операционную систему, которая запускается в этой виртуальной среде — **гостевой системой**. Хостовая и гостевая ОС могут иметь взаимоисключающие компоненты, но виртуальная среда позволяет им жить «дружно».

Чтобы создать на компьютере или сервере виртуальную среду, можно установить специальную программу — **виртуальную машину** (с помощью **VirtualBox** или **VMware**).

То есть виртуализация даёт возможность запуска нескольких операционных систем на одном физическом устройстве. Виртуализация использует ресурсы устройства (память, процессор, устройство ввода и вывода), но при этом работает как отдельный компьютер со своей операционной системой.

Виртуальные машины работают благодаря аппаратной виртуализации — это способ, при котором в создании изолированной виртуальной среды задействованы гостевые операционные системы.

В основе аппаратной виртуализации лежит работа гипервизора — он отвечает за изоляцию виртуальных сред и распределение ресурсов центральной машины.

Контейнерная виртуализация — это способ, при котором виртуальная среда запускается прямо из ядра хостовой операционной системы (то есть мы не устанавливаем другую ОС). В данном случае изоляцию ОС и приложений поддерживает контейнер. Он содержит специальный набор файлов, а также все зависимости запускаемого в нём приложения: код, библиотеки, инструменты и настройки. Всё это упаковано в отдельный образ, работу которого запускает движок (контейнерный движок), например, такой как *Docker*, *CRI-O*, *Railcar* и прочие.

Отличие контейнера от виртуальной машины

В случае с контейнерами у нас есть базовая аппаратная инфраструктура (железо компьютера), операционная система и движок, установленный на этой ОС. Движок управляет контейнерами, которые работают с библиотеками и зависимостями сами, в одиночку.

В случае виртуальной машины у нас есть ОС на базовом оборудовании (наше железо), затем гипервизор, такой как *ESXi*, а затем виртуальные машины. У каждой виртуальной машины внутри своя операционная система.

Эти накладные расходы приводят к более высокому использованию вычислительных ресурсов, поскольку приходится крутить несколько виртуальных операционных систем с их ядрами.

Виртуальные машины также потребляют больше дискового пространства, поскольку каждая виртуальная машина тяжелая и обычно имеет размер в гигабайтах, тогда как контейнеры легковесны и обычно имеют размер в мегабайтах. Это позволяет контейнерам загружаться быстрее и работать эффективнее.



Различия контейнеров и виртуальных машин. Источник: Курс Udemy Андрей Соколов - Docker для начинающих и чайников + практический опыт (2021)

При этом ничего не мешает работать с контейнерами на виртуальных машинах.

С таким подходом не нужно создавать определенную виртуальную машину под конкретное приложение. Задача этой виртуальной машины - запускать контейнеры.



Объединение контейнеров и виртуальных машин. Источник: Курс Udemy Андрей Соколов - Docker для начинающих и чайников + практический опыт

Таким образом, **контейнер** — это альтернатива и одновременно коллекция виртуальной машины. Существуют различные инструменты для работы с контейнерами, одним из которых является **Docker**.

Что такое Docker

Docker — это платформа для разработки, запуска и управления контейнерами. Докер позволяет создавать контейнеры, автоматизирует их запуск и управляет жизненным циклом.

Докер состоит из нескольких компонентов:

1. Docker-host (докер-хост)

Это компьютер, на котором работает докер: персональный компьютер, сервер или виртуальная машина в облаке.

2. Docker-daemon (докер-демон)

Это фоновый процесс ("душа докера"), который работает постоянно и ожидает команды. Все операции по управлению контейнерами отправляются именно в демон. Например: запустить или остановить контейнер, скачать образ. И уже на основе этих команд демон выполняет необходимые действия с контейнерами и образами. Поэтому докер-демон знает все о контейнерах, запущенных на одном хосте: сколько всего контейнеров, какие из них работают, где хранятся образы и так далее.

3. Docker-client (докер-клиент)

Это клиент, при помощи которого пользователи взаимодействуют с демоном и отправляют ему команды. Это может быть консоль, API или графический интерфейс. То есть это то, через что мы отправляем запросы докер-демону.

4. Docker-image (докер-образ)

Это неизменяемый образ, из которого разворачивается контейнер. Его можно рассматривать как набор файлов,

необходимых для запуска и работы приложения на другом хосте. Аналогия из мира установки ПО: образ — это компакт-диск, с которого устанавливается программа.

5. Docker-container (докер-контейнер)

Это уже развернутое (на основе образа) и запущенное приложение. Продолжая аналогию с установкой ПО, контейнер можно сравнить с уже установленной и работающей программой на ПК (которую установили с диска - образа).

6. Dockerfile (докер-файл)

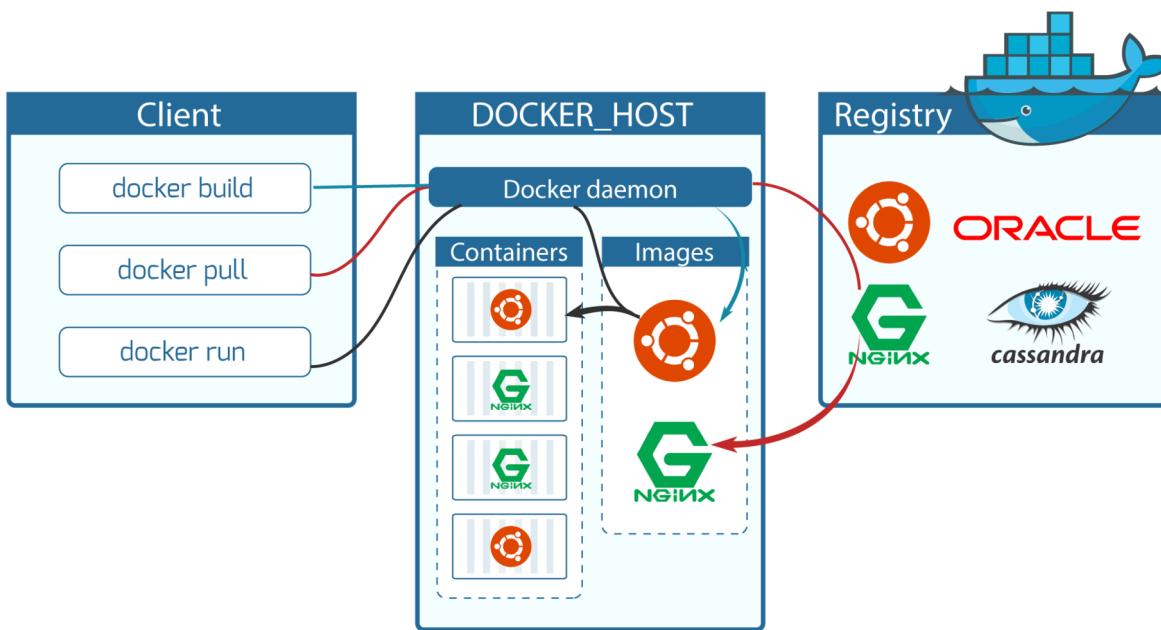
Это файл-инструкция для сборки образа.

7. Docker registry

Это репозиторий (место), в котором хранятся образы. Когда разработчики создают приложения, они размещают свои образы в этих репозиториях, откуда их могут скачать другие люди. Есть публичные репозитории, например, [Docker Hub](#). Можно создать свой закрытый репозиторий, для использования внутри компании или команды. Это своего рода [GitHub](#), только на котором хранится не программный код, докер-образы.

На [Docker Hub](#) собраны образы множества популярных программ или платформ: базы данных, веб-серверы, компиляторы, операционные системы и так далее.

DOCKER COMPONENTS



Компоненты Docker. Источник: <https://blog.knoldus.com/docker-components/>

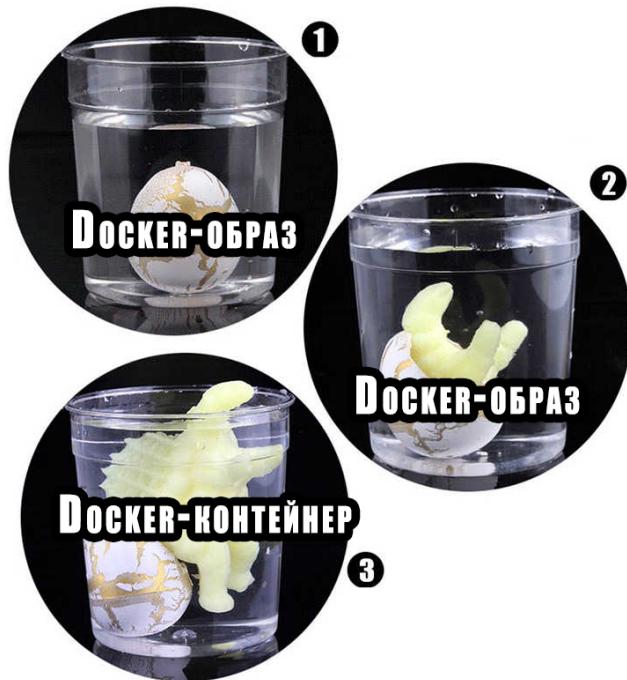
Чуть подробнее про Образы и контейнеры (Images и Container)

Docker-image (Образ) — шаблон с набором некоторых инструкций, предназначенных для создания контейнера.

Приложения (и всё что им нужно) упаковываются именно в образы, из которых потом уже создаются контейнеры.

Приведем аналогию на примере установки операционной системы. В дистрибутиве (образе) ОС есть все, что необходимо для ее установки. Но этот образ нельзя запустить, для начала его нужно «развернуть» в готовую ОС. Так вот, дистрибутив для установки ОС — это Docker image, а установленная и работающая ОС — это Docker container.

Docker-Container (Контейнер) — уже собранное, настроенное и запущенное (на основе образа) приложение в изолированной среде.



Развертывание контейнера из образа. Источник: https://aliexpress.ru/item/32819987922.html?sku_id=64743726793

Больше аналогий богу аналогий. Образ мы можем сравнить чертежом дома. Мы видимо из чего он состоит, как что и где расположено, каких размеров элементы и т.д. А контейнер - это уже построенный по чертежу дом, в котором живут люди (приложения).

Образы создаются путём редактирования других образов или с нуля с помощью специальных файлов - **Dockerfile**.

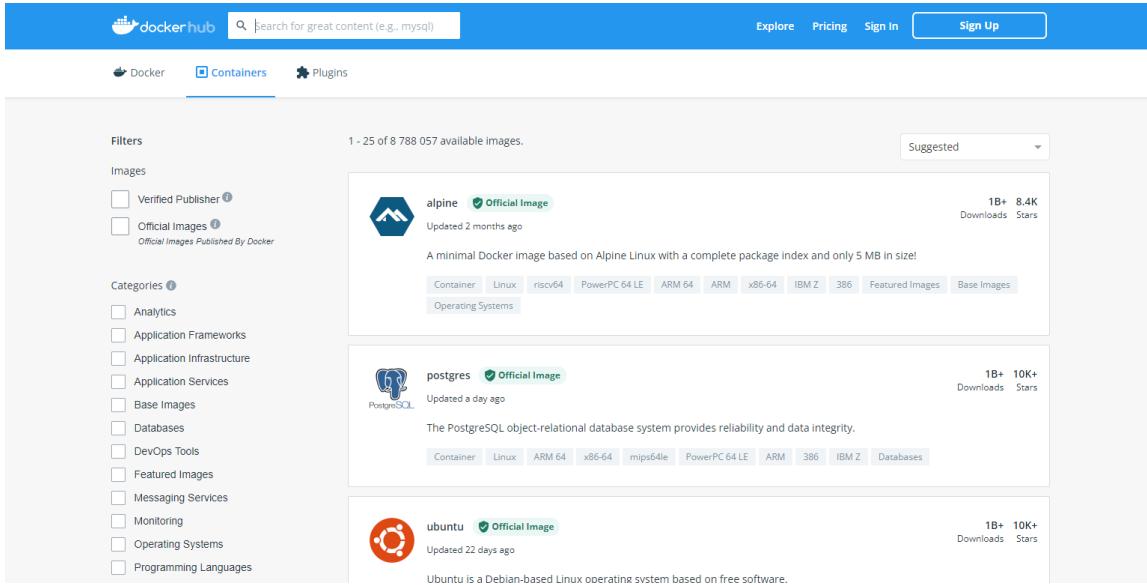
Dockerfile

Dockerfile — это инструкция для сборки образа. Это простой текстовый файл, содержащий по одной команде в каждой строке. В нем указываются все программы, зависимости, образы, и команды которые нужны для создания контейнера.

То есть это файл с построчным перечислением необходимых компонентов и последовательности действий, которые надо совершить. Подробнее с Dockerfile мы познакомимся дальше.

```
FROM python:3.7.2-alpine3.8
LABEL maintainer="pylounge@mail.ru"
# Устанавливаем зависимости
RUN apt-get install git
# Задаём текущую рабочую директорию
WORKDIR /usr/src/my_app_directory
# Копируем код из локального контекста в рабочую директорию образа
COPY .
# Задаём значение по умолчанию для переменной
ARG my_var=my_default_value
# Настраиваем команду, которая должна быть запущена в контейнере во время его выполнения
ENTRYPOINT ["python", "./app/my_script.py", "my_var"]
# Открываем порты
EXPOSE 8000
# Создаём том для хранения данных
VOLUME /my_volume
```

Созданные на основе Dockerfile образы, для дальнейшего распространения, загружают в Docker репозитории, например, Docker Hub.

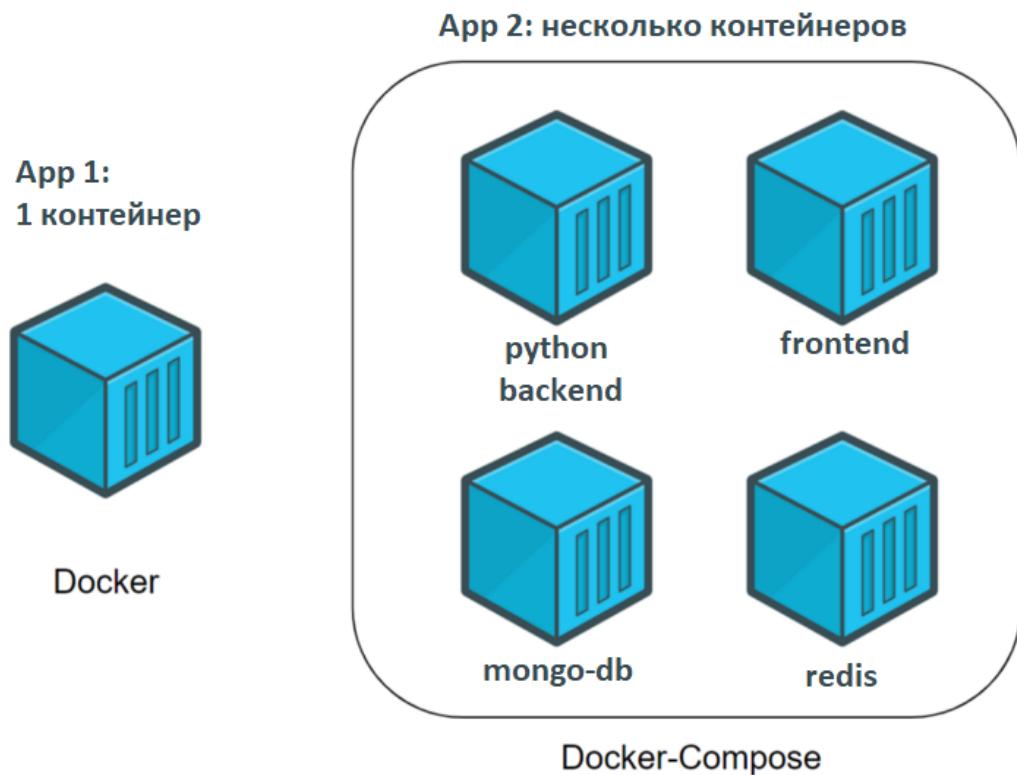


Docker Hub. Источник: <https://hub.docker.com/search?q=&type=image>

В определённый момент (когда образов и контейнеров становится много) и возможностей одного докера не хватает, на помощь ему приходят дополнительные инструменты в лице [Docker Compose](#) и [Docker Swarm/Kubernetes](#).

Docker Compose

[Docker-compose](#) - инструмент, который позволяет разворачивать и настраивать несколько контейнеров одновременно. Например, для веб-приложения нужно развернуть стек LAMP: Linux, Apache, MySQL, PHP. Каждое из приложений — это отдельный контейнер. Но в этой ситуации нам нужны именно все контейнеры вместе, а не отдельно взятое приложение. Docker-compose позволяет развернуть и настроить все приложения одной командой, а без него пришлось разворачивать и настраивать каждый контейнер отдельно.



Docker-Compose запускает несколько контейнеров на одном компьютере/виртуальной машине. Источник: <https://ivan-shamaev.ru/docker-compose-tutorial-container-image-install/>

О работе с Docker Compose будет посвящён отдельный обстоятельный разговор.

Docker Swarm

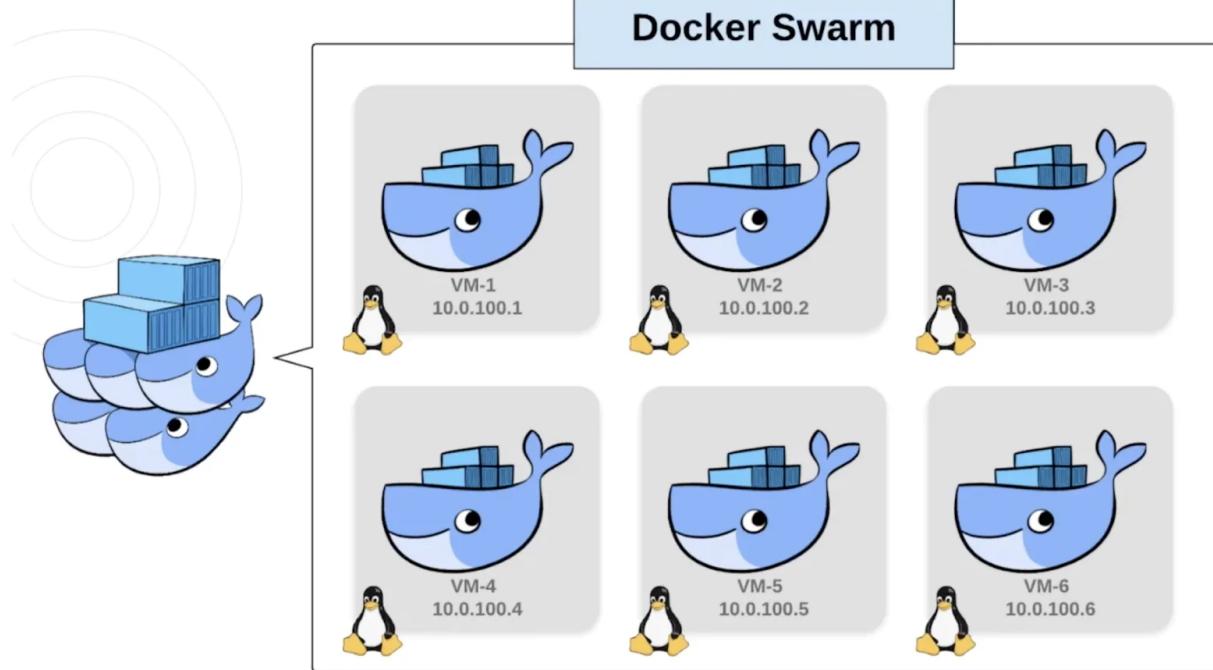
При работе с контейнерами не лишним будет ознакомиться с понятием оркестрации.

Оркестровка — это управление и координация взаимодействия между контейнерами. Контейнеры запускаются на хостах, а хосты объединяют в клUSTER.

У Docker есть стандартный инструмент оркестровки — Docker Swarm Mode, или просто Docker Swarm. Он поставляется «из коробки», довольно прост в настройке и позволяет создать простой кластер буквально за минуту.

Когда вы работаете в производственной среде, на сотнях докер-контейнеров будет работать несколько приложений.

Управление всеми этими контейнерами может быть большой головной болью. Вот где Docker Swarm поможет вам. Он легко управляет и управляет кластером, в котором работают несколько докер-контейнеров.



Docker Swarm запускает контейнер на нескольких виртуальных машинах. Источник: <https://testautomationu.applitools.com/scaling-tests-with-docker/chapter5.html>

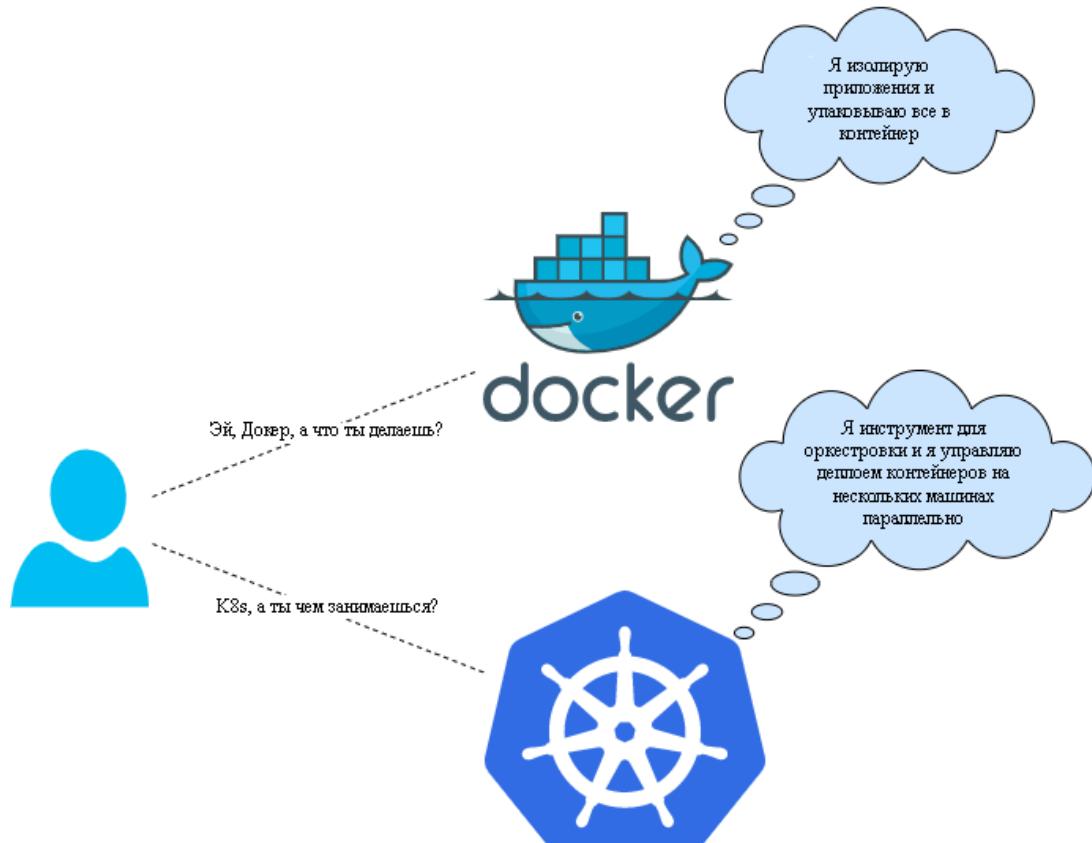
Внимание! Не стоит путать!!!

Docker Swarm Mode - это про **кластеры и оркестрацию**. Можно запустить много экземпляров одного контейнера на нескольких разных машинах.

Docker-compose - это просто про запуск нескольких (разных) контейнеров вместе на одной машине.

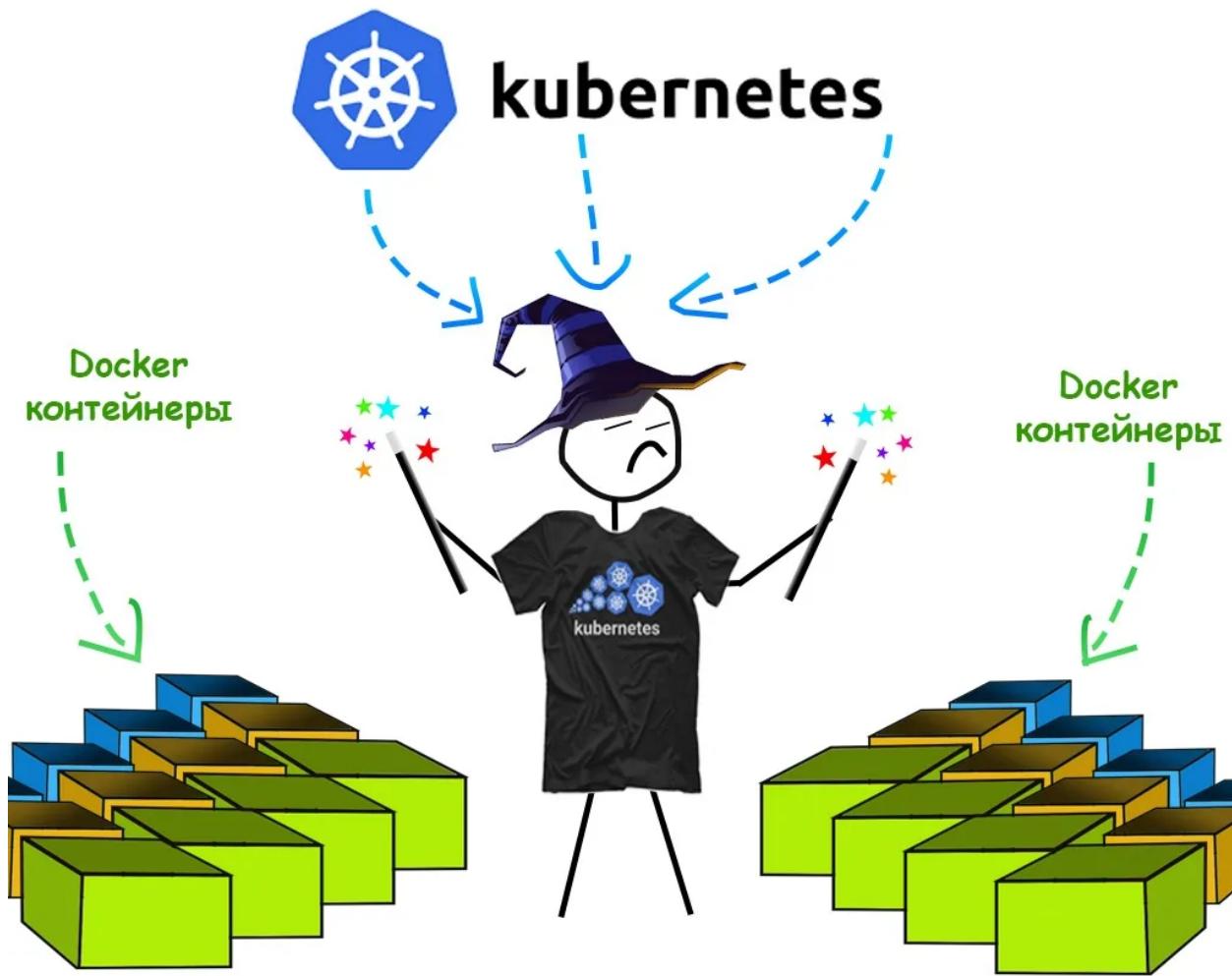
Kubernetes

Кроме стандартного **Docker Swarm** есть и другие инструменты оркестровки, например **Kubernetes**. Это сложная система, которая позволяет построить отказоустойчивую и масштабируемую платформу для управления контейнерами. Он умеет работать не только с контейнерами Docker, но и с другими контейнерами: rkt, CRI-O.



Разница между Docker и Kubernetes. Источник: <https://techrocks.ru/2020/06/11/difference-between-docker-and-kubernetes/>

Kubernetes похож на Docker Swarm. Однако служба Kubernetes более обширна, чем Docker Swarm, и предназначена для эффективной координации кластеров узлов в масштабировании в процессе производства.



Источник: <https://bor64.com/2019/03/12/docker-shmoker-i-kubernetes-chto-jeto-i-zachem/>

Установка Docker

Пакет установки Docker, доступен в [официальном репозитории Ubuntu](#), однако оттуда может быть установлена не самая последняя версия. Чтобы точно загрузить самую актуальную версию, необходимо устанавливать Docker из [официального репозитория Docker](#).

Для этого необходимо добавить новый источник пакета, ключ GPG от Docker, а затем уже установить пакет через пакетный менеджер.

Установка Docker для ОС Linux (Ubuntu 20.04)

1. Обновите существующий список пакетов Ubuntu:

```
sudo apt-get update  
sudo apt-get upgrade
```

2. Установите пакеты, которые позволяют `apt-get` работать с репозиториями через `HTTPS`:

```
sudo apt-get install \  
ca-certificates \<
```

```
curl \
gnupg \
lsb-release
```

3. Добавьте ключ GPG для официального репозитория Docker:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

1. Добавьте репозиторий **Docker** в источники `apt-get`:

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
```

5. Обновите базу данных пакетов и добавьте в нее пакеты Docker из добавленного репозитория:

```
sudo apt-get update
```

Список добавленных в Ubuntu репозиториев можно посмотреть в файле `/etc/apt/sources.list` или с помощью команды `sudo grep -rhE ^deb /etc/apt/sources.list*`

6. Установите Docker через менеджер `apt-get`:

```
sudo apt install docker-ce
```

7. Убедимся, что установка прошла успешно, проверив статус службы:

```
sudo systemctl status docker
```

Результат должен быть примерно следующий (может разнится от системы к системе):

```
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset>
   Active: active (running) since Mon 2022-01-24 10:53:46 MSK; 2min 10s ago
     TriggeredBy: ● docker.socket
   Docs: https://docs.docker.com
      Main PID: 4479 (dockerd)
        Tasks: 7
       Memory: 27.6
      CGroup: /system.slice/docker.service
              └─4479 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/cont>
```

Здесь следует обратить внимание на строчку `Active: active (running)` сигнализирующую о том, что служба **docker** сейчас работает.

После установки Docker будет доступна Docker-служба (демон-процесс) и утилита командной строки `docker`, с помощью которой и будет происходить дальнейшее взаимодействие.

Установка Docker для ОС Windows 10 (на основе WSL)

Для запуска Docker на Windows необходимо активировать WSL2.

WSL2 - это подсистема Windows для Linux позволяет разработчикам запускать среду GNU/Linux с большинством программ командной строки, служебных программ и приложений непосредственно в Windows без каких-либо изменений и необходимости использовать традиционную виртуальную машину или двойную загрузку.

Подробный процесс установки WSL описан на сайте Microsoft <https://docs.microsoft.com/ru-ru/windows/wsl/install-win10>

Либо наглядно в этом ролике:

https://www.youtube.com/watch?v=NlObauL_XT4

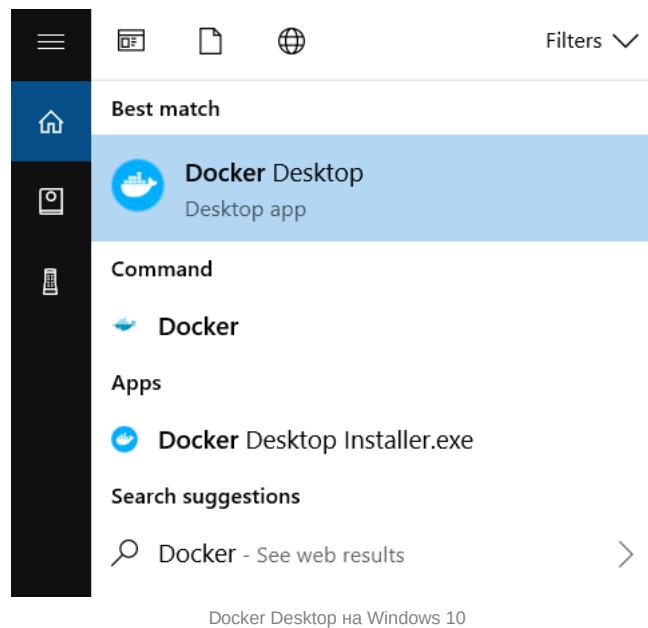
WSL На Windows 10 | Установка И Настройка | Как Установить Linux В Windows 10

Теперь установим **Docker Desktop WSL 2 backend**, идем по ссылке <https://hub.docker.com/editions/community/docker-ce-desktop-windows/> Скачиваем и устанавливаем **Docker Desktop for Windows** (stable).

Запускаем загруженный файл и производим обычную установку приложения Windows.

При установке убедитесь что установлена галочка на **Enable WSL 2 Windows Features**.

После установки следуйте инструкциям и перелогиньтесь в Windows. Докер запуститься при следующем входе в Windows, иногда в первый раз может понадобится довольно длительное время.



Docker Desktop на Windows 10

Основы работы с Docker

Практика 1. Установка Nextcloud

Nextcloud (ответвление проекта ownCloud) — напоминающий **Dropbox** файлообменный сервер для централизованного хранения персонального контента, в том числе документов и изображений. Отличие Nextcloud в том, что весь его функционал реализован с открытым исходным кодом. Также Nextcloud возвращает вам возможность управления и защиты важных данных, не требуя использования стороннего облачного сервиса для хостинга.

Контейнеры Docker получают из **образов Docker**. По умолчанию Docker загружает эти образы из **Docker Hub**, реестр Docker, контролируемые Docker, т.е. компанией, реализующей проект Docker. Любой может размещать свои образы Docker на Docker Hub, поэтому большинство приложений и дистрибутивов Linux, которые вам потребуются, хранят там свои образы.

Для начала необходимо найти **официальный образ Nextcloud** на сайте **Docker Hub**.

The screenshot shows the Docker Hub interface for the 'nextcloud' image. At the top, there's a search bar with 'Search for great content (e.g., mysql)'. Below it, a navigation bar includes 'Explore', 'Pricing', 'Sign In', and 'Sign Up'. The main content area features a large icon of a ship with a stack of blocks, followed by the text 'nextcloud' and 'Official Image'. A star icon indicates it's popular. Below this, a tagline reads 'A safe home for all your data'. A download button shows '500M+'. A horizontal bar lists supported architectures: Container, Linux, ARM, ARM 64, IBM Z, x86-64, mips64le, PowerPC 64 LE, 386, Application Frameworks, Application Infrastructure, and Official Image. On the right, there's a 'Copy and paste to pull this image' section with the command 'docker pull nextcloud' and a 'View Available Tags' link.

Использование `docker` подразумевает передачу ему цепочки опций (флагов) и команд, за которыми следуют аргументы. Синтаксис имеет следующую форму:

```
docker [option] [command] [arguments]
```

Чтобы просмотреть все доступные команды, введите:

```
docker
```

Полный список команд выглядит следующим образом:

```

Output
attach      Attach local standard input, output, and error streams to a running container
build       Build an image from a Dockerfile
commit      Create a new image from a container's changes
cp          Copy files/folders between a container and the local filesystem
create      Create a new container
diff        Inspect changes to files or directories on a container's filesystem
events     Get real time events from the server
exec        Run a command in a running container
export      Export a container's filesystem as a tar archive
history    Show the history of an image
images     List images
import     Import the contents from a tarball to create a filesystem image
info        Display system-wide information
inspect    Return low-level information on Docker objects
kill        Kill one or more running containers
load        Load an image from a tar archive or STDIN
login      Log in to a Docker registry
logout    Log out from a Docker registry
logs       Fetch the logs of a container
pause      Pause all processes within one or more containers
port       List port mappings or a specific mapping for the container
ps         List containers
pull       Pull an image or a repository from a registry
push       Push an image or a repository to a registry
rename    Rename a container
restart   Restart one or more containers
rm        Remove one or more containers
rmi      Remove one or more images
run       Run a command in a new container
save      Save one or more images to a tar archive (streamed to STDOUT by default)
search   Search the Docker Hub for images
start    Start one or more stopped containers
stats    Display a live stream of container(s) resource usage statistics

```

```
stop      Stop one or more running containers
tag       Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
top       Display the running processes of a container
unpause   Unpause all processes within one or more containers
update    Update configuration of one or more containers
version   Show the Docker version information
wait     Block until one or more containers stop, then print their exit codes
```

Чтобы просмотреть параметры, доступные для конкретной команды, введите: `docker docker-command --help`

После того как мы нашли образ (в нашем случае образ так и называется **nextcloud**), можно загрузить его на свой компьютер с помощью команды `pull`.

Загрузим официальный образ **nextcloud** на свой компьютер из Docker Hub:

```
docker pull nextcloud
```

Чтобы просмотреть образы, которые были загружены на ваш компьютер, введите:

```
docker images
```

На данный момент у нас имеется единственный образ **nextcloud**, который мы загрузили на предыдущем этапе:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nextcloud	latest	6dfcd73f8597	4 days ago	969MB

Образы, которые вы используете для запуска контейнеров, можно изменить или использовать для создания новых образов, которые в дальнейшем можно загрузить на Docker Hub, чтобы другие разработчики могли их использовать.

После того как образ будет загружен, вы сможете запустить контейнер (созданный на основе этого образа) с помощью команды `run`.

```
docker run nextcloud
```

Кроме того, после команды `run` можно передавать дополнительные опции и параметры, влияющие на запуск контейнера.

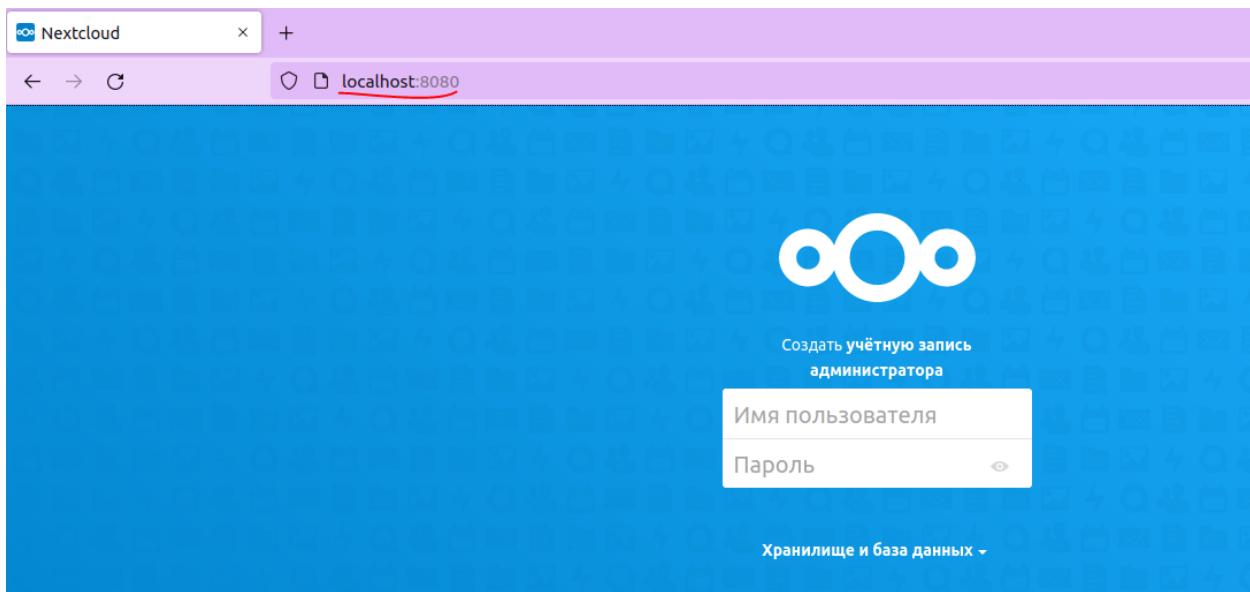
Например, запустим контейнер с **nextcloud** предварительно указав, чтобы сервис был доступен на порту `8080`:

```
docker run -d -p 8080:80 nextcloud
```

Команда выше использует несколько флагов. Немного информации о них:

1. `-d` - запустить контейнер в автономном режиме (в фоновом режиме);
2. `-p 8080:80` - сопоставить порт 8080 хоста с портом 80 в контейнере.

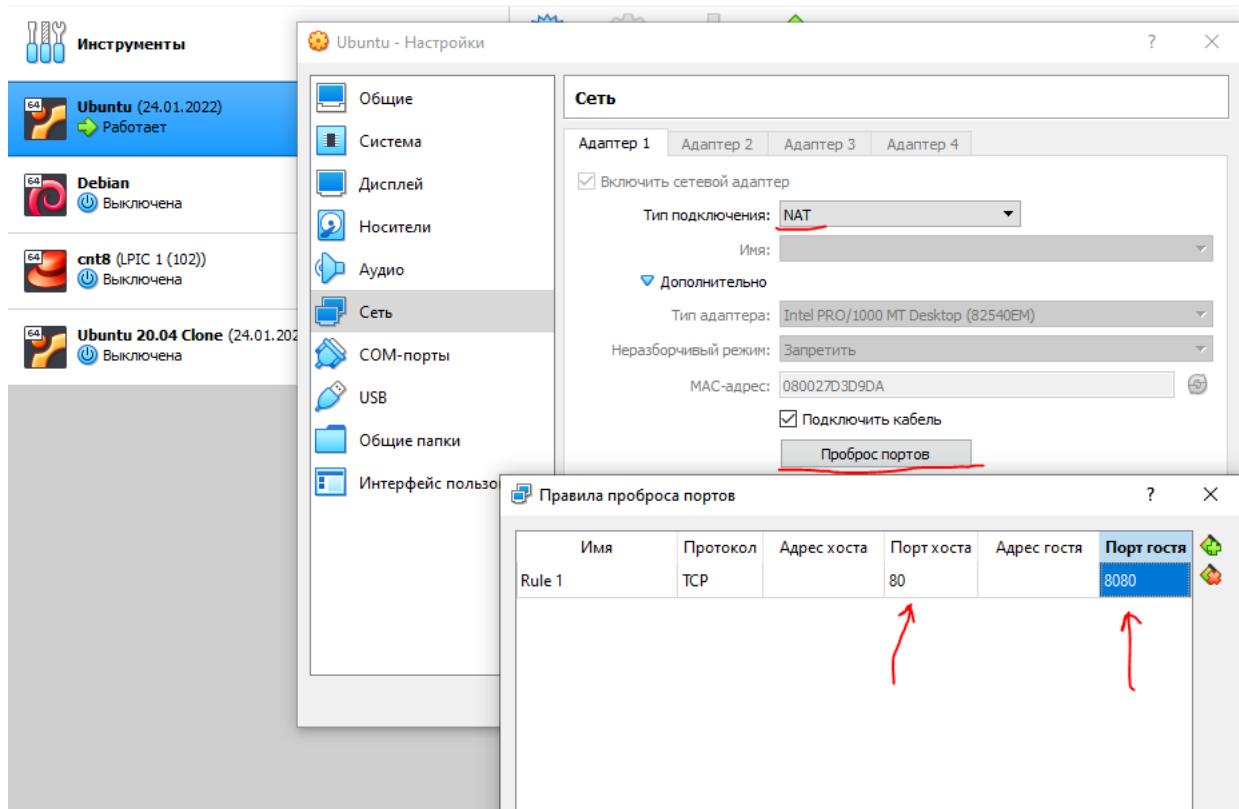
Теперь можно получить доступ к Nextcloud по адресу <http://localhost:8080/> (или http://ip_адрес:8080) из хост-системы:



Доступ к Nextcloud через запрос в браузере

IP-адрес компьютера можно посмотреть с помощью команды `ifconfig` (или `ip addr`) на ОС семейства Linux или с помощью команды `ipconfig` на Windows.

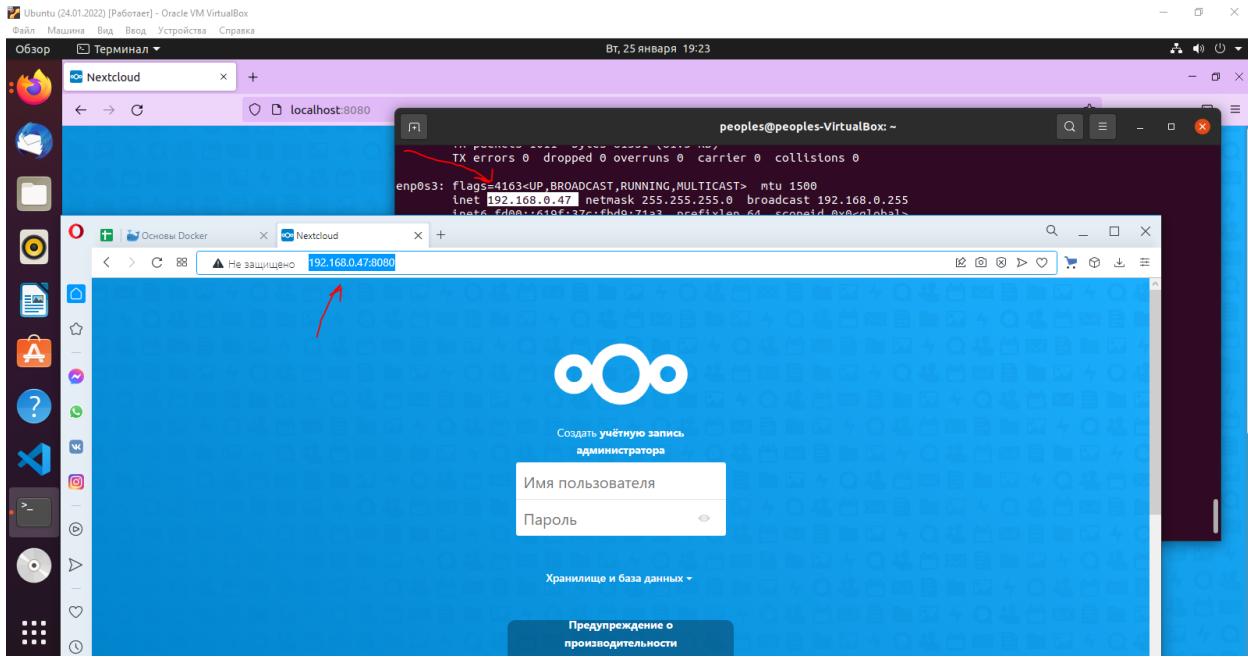
Если в качестве хост-системы для Docker вы используете виртуальную машину *VirtualBox* с Типом подключения сети - **NAT**, то для доступа к Nextcloud из основной ОС (хост-ос), то необходимо настроить *Проброс портов* как на скриншоте ниже:



Проброс портов виртуальной машины

Затем в адресной строке браузера хост-машины ввести <http://localhost:80>.

Если Тип подключения сети VirtualBox - **Сетевой мост**, то
http://ip_адрес_виртуальной_машины:8080



Доступ к Nextcloud из хост-ос по IP-адресу виртуальной машины

Таким образом контейнер с **nextcloud** успешно настроен и запущен.

Чтобы просмотреть **активные (запущенные в данный момент) контейнеры**, используйте следующую команду `docker ps` :

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
e176ec2e0cd6	nextcloud	"/entrypoint.sh apache2"	23 minutes ago	Up 23 minutes	0.0.0.0:8080->80/tcp, :::8080->80/tcp	eloquent_almeida

Чтобы просмотреть все контейнеры — активные и неактивные, воспользуйтесь командой `docker ps` с флагом `a` :

```
docker ps -a
```

Чтобы остановить (выключить) запущенный контейнер, используйте команду `docker stop` с **идентификатором контейнера (CONTAINER ID)** или **именем контейнера (NAMES)**.

В нашем случае остановить контейнер **nextcloud** через **CONTAINER ID** можно следующим образом:

```
docker stop e176ec2e0cd6
```

А через **NAMES** так:

```
docker stop eloquent_almeida
```

Аналогичным образом запустить остановленный контейнер можно с помощью команды `docker start` с указанием **идентификатором контейнера (CONTAINER ID)** или **именем контейнера (NAMES)**:

```
docker start e176ec2e0cd6
docker start eloquent_almeida
```

Если контейнер больше вам не нужен, то удалите его с помощью команды `docker rm`, снова добавив идентификатор контейнера или его имя.

```
docker rm e176ec2e0cd6
```

Выполнить “вход” внутрь контейнера для внесения в него изменений (редактирование файлов конфигурации, установка программ и т.д.) можно с помощью команды `exec` и комбинации флагов `-it`:

```
docker exec -it e176ec2e0cd6 bash
```

Мы оказываемся в терминале среды контейнера. Добавим в рабочую директорию какой-нибудь текстовый файл, например, `1.txt` и установим текстовый редактор `vim`:

```
root@9125fed6dffff:/var/www/html# touch 1.txt
root@9125fed6dffff:/var/www/html# apt update
root@9125fed6dffff:/var/www/html# apt install vim
```

Выйдем из из интерактивного режима с помощью команды `exit`.

Любые изменения, которые внесены внутри контейнера, применяются **только** к контейнеру.

После запуска образа Docker можно создавать, изменять и удалять файлы так же, как и с помощью виртуальной машины. Эти изменения будут применяться только к данному контейнеру. Можно запускать и останавливать контейнер, но после того как он будет удалён с помощью команды `docker rm`, изменения будут утрачены навсегда.

Чтобы внести изменения в контейнер необходимо выполнить:

```
//docker commit -m "Комментарий" -a "Кто изменил" container_id repository/new_image_name
```

Переключатель `-m` используется в качестве сообщения о внесении изменений, которое помогает вам и остальным узнать, какие изменения внесли, в то время как `-a` используется для указания автора. `container_id` — это тот самый идентификатор, который отмечали ранее, когда запускали интерактивную сессию Docker. Если вы не создавали дополнительные репозитории на Docker Hub, `repository`, как правило, является вашим именем пользователя на Docker Hub.

Например, для пользователя `peoples` с идентификатором контейнера `e176ec2e0cd6bash` команда будет выглядеть следующим образом:

```
docker commit -m "Add txt file and install vim" -a "pylounge" e176ec2e0cd6bash nextcloud/nextcloud_with_txt
```

Когда вы *вносите* изменения в образ, новый образ сохраняется локально на компьютере.

Список образов Docker теперь будет содержать новый образ, а также старый образ, из которого он получен:

```
peoples@peoples-VirtualBox:~$ sudo docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
nextcloud/nextcloud_with_txt    latest   4e025d8225a9  12 seconds ago  969MB
nextcloud            latest   6dfcd73f8597  4 days ago    969MB
```

В данном примере `nextcloud_with_txt` является новым образом, который был получен из образа `nextcloud` на Docker Hub. Разница в размере отражает внесенные изменения. В данном примере изменение состояло в том, что добавлен файл `1.txt` и установлен текстовый редактор `vim`.

Вы также можете создавать образы из `Dockerfile`, что позволяет автоматизировать установку программного обеспечения в новом образе (о чём далее).

Запускаем контейнер с обновлённым образом:

```
docker run -d -p 8080:80 nextcloud_with_txt
```

Загрузка образов Docker в репозиторий Docker (Docker Hub)

После создания нового образа его можно загрузить на Docker Hub или в другой реестре Docker. Чтобы добавить образ на Docker Hub или любой другой реестр Docker, у вас должна быть там учетная запись.

Чтобы загрузить свой образ необходимо выполнить вход в Docker Hub:

```
docker login -u docker-registry-username
```

Будет предложено использовать для аутентификации пароль Docker Hub. Если вы указали правильный пароль, аутентификация должна быть выполнена успешно.

Если ваше имя пользователя в реестре Docker отличается от локального имени пользователя, которое указывалось при создании образа, потребуется пометить ваш образ именем пользователя в реестре:

```
docker tag peoples/nextcloud_with_txt docker-registry-username/nextcloud_with_txt
```

Затем можно загрузить образ с помощью следующей команды:

```
docker push pylounge/nextcloud_with_txt
```

После добавления образа в реестр он должен отображаться в панели вашей учетной записи.

Теперь можно использовать `docker pull pylounge/nextcloud_with_txt`, чтобы загрузить образ на новый компьютер и использовать его для запуска нового контейнера.

Практика 2: Создание Docker образа с приложением на Python

Создание Flask приложения

Для начала создадим простое Flask-приложение. Необходимо создать пустой каталог для нашего проекта, в котором будут содержаться все файлы проекта. Внутри каталога создать виртуальное окружение и активировать его:

```
peoples@peoples-VirtualBox:~$ mkdir pylounge_docker_project
peoples@peoples-VirtualBox:~$ cd pylounge_docker_project/
peoples@peoples-VirtualBox:~/pylounge_docker_project$ sudo apt-get install python3.8-venv
peoples@peoples-VirtualBox:~/pylounge_docker_project$ python3 -m venv env
peoples@peoples-VirtualBox:~/pylounge_docker_project$ source env/bin/activate
```

Установим Flask:

```
(env) peoples@peoples-VirtualBox:~/pylounge_docker_project$ python -m pip install flask
```

Создадим отдельную директорию, в которой будет находиться само Flask-приложение:

```
(env) peoples@peoples-VirtualBox:~/pylounge_docker_project$ mkdir pylounge_flask_app
```

Сохраните список установленных в окружении пакетов в `requirements.txt` в каталоге приложения:

```
(env) peoples@peoples-VirtualBox:~/pylounge_docker_project$ pip freeze > pylounge_flask_app/requirements.txt
```

Создайте файл `app.py` в каталоге `pylounge_flask_app`, который будет содержать простое python Flask Web-приложение:

```
sudo nano pylounge_flask_app/app.py
```

И добавим туда следующий программный код на языке Python:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from flask import Flask
app = Flask(__name__)

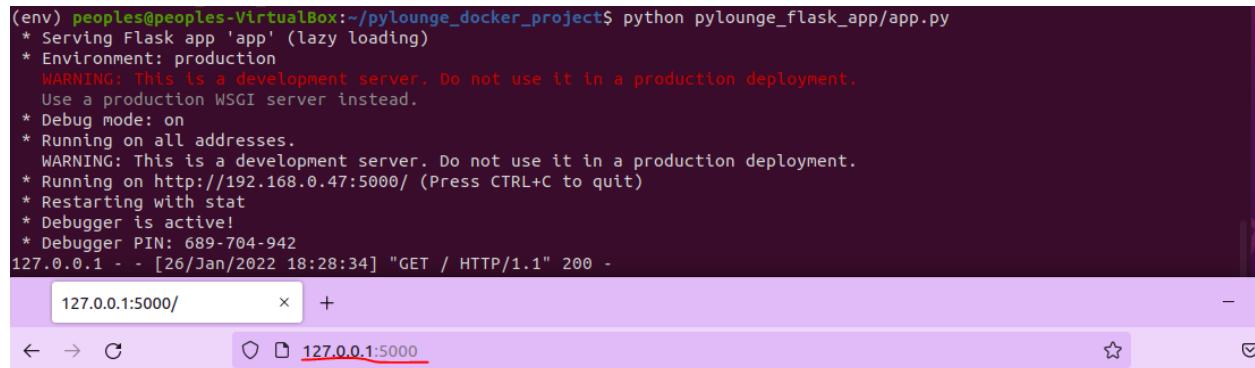
@app.route('/')
def hello_world():
    return 'Hello from PyLounge and Docker!'

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

Проверим работоспособность приложения, запустив его командой:

```
python pylounge_flask_app/app.py
```

В адресной строке веб-браузера введём <http://127.0.0.1:5000/>



Работа Flask приложения

Если отобразилась веб страница с текстом *Hello from PyLounge and Docker!* значит сделано всё правильно.

Упаковка Flask приложения в контейнер Dockerfile

Для создания Docker образа (Docker-image) используется специальный **Dockerfile**.

Dockerfile — это текстовый файл, в котором описан рецепт создания образа Docker. Рецепт состоит из инструкций, которые выполняются последовательно. Они содержат информацию об операционной системе, выбранной платформе, фреймворках, библиотеках, инструментах, которые нужно установить.

Структура Dockerfile

- **FROM** — задаёт родительский (главный) образ;
- **LABEL** — добавляет метаданные для образа. Хорошее место для размещения информации об авторе;
- **ENV** — создаёт переменную окружения;
- **RUN** — запускает команды, создаёт слой образа. Используется для установки пакетов и библиотек внутри контейнера;
- **COPY** — копирует файлы и директории в контейнер;

- `ADD` — делает всё то же, что и инструкция `COPY`. Но ещё может распаковывать локальные `.tar` файлы;
- `CMD` — указывает команду и аргументы для выполнения внутри контейнера. Параметры могут быть переопределены. Использоваться может только одна инструкция `CMD`;
- `WORKDIR` — устанавливает рабочую директорию для инструкции `CMD` и `ENTRYPOINT`;
- `ARG` — определяет переменную для передачи Docker'у во время сборки;
- `ENTRYPOINT` — предоставляет команды и аргументы для выполняющегося контейнера. Суть его несколько отличается от `CMD`, о чём мы поговорим ниже;
- `EXPOSE` — открывает порт;
- `VOLUME` — создаёт точку подключения директории для добавления и хранения постоянных данных.

Создадим Dockerfile в директории `pylounge_flask_app`:

```
sudo nano pylounge_flask_app/Dockerfile
```

Содержимое Dockerfile файла в нашем случае будет следующим:

```
FROM ubuntu:latest
MAINTAINER Maxim Melnikov 'pylounge@mail.ru'
RUN apt-get update -qy
RUN apt-get install -qy python3.8 python3-pip python3.8-dev
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
CMD ["python3","app.py"]
```

Описание инструкций Dockerfile

Инструкция	Описание
<code>FROM ubuntu:latest</code>	В качестве базового образа будет использоваться latest версия образа Ubuntu (то есть мы создаем наш образ на основе образа ubuntu из Docker Hub)
<code>MAINTAINER Maxim Melnikov 'pylounge@mail.ru'</code>	Справочная информация об авторе образа. Не обязательный параметр
<code>RUN apt-get update -qy</code>	Обновить информацию о репозиториях внутри контейнера
<code>RUN apt-get install -qy python3.8 python3-pip python3.8-dev</code>	Установить внутрь контейнера пакеты: python3.8 python3-pip, python3.8-dev
<code>COPY . /app</code>	Скопировать содержимое текущей директории «.» в директорию /app внутри образа. Внимание: текущей директорией в процессе сборки будет считаться директория, содержащая Dockerfile, т.е. в нашем случае pylounge_flask_app/
<code>WORKDIR /app</code>	Сменить рабочую директорию внутри контейнера. Все команды далее будут запускаться внутри директории /app внутри контейнера
<code>RUN pip install -r requirements.txt</code>	Установить зависимости, сохраненные вами в requirements.txt через pip freeze. Данная команда установить Flask и все, что необходимо для его запуска внутри контейнера
<code>CMD ["python3","app.py"]</code>	Интерпретатору python будет передан дополнительный аргумент app.py. Другими словами, во время запуска контейнера последней инструкцией будет выполнена команда python3 app.py из директории /app

Создание образа контейнера на основе Dockerfile

Для создания собственного контейнера, содержащего созданное Flask приложение, на основе также созданного нами Dockerfile, находясь внутри директории `pylounge_docker_project` необходимо выполнить команду `build`:

```
sudo docker build -t pylounge_flask_app:v1 pylounge_flask_app/
```

Ключ `-t` предназначен для того, чтобы присвоить вашему образу метку (label) «`pylounge_flask_app`» и его версию «`v1`». Метка и версия могут быть произвольными.

В процессе выполнения команды как описано в Dockerfile будет произведена загрузка последнего образа `ubuntu`, внутри него установлены все необходимые зависимости, создана директория `/app`, в которую будет помещено содержимое директории `pylounge_flask_app/`, установлены все зависимости из файла `requirements.txt`, а сам образ настроен на запуск вашего Flask приложения из директории `/app`, находящейся внутри образа.

После успешного выполнения команды в списке образов появится только что созданный образ `pylounge_flask_app:v1`:

```
docker images

REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
pylounge_flask_app  v1       0079c1c47c56  About a minute ago  458MB
nextcloud/nextcloud_with_txt  latest   4e025d8225a9  23 hours ago   969MB
nextcloud            latest   6dfcd73f8597  5 days ago    969MB
ubuntu              latest   d13c942271d6  2 weeks ago   72.8MB
```

Для запуска Docker контейнера из подготовленного образа необходимо выполнить команду:

```
sudo docker run -it -p 5000:5000 pylounge_flask_app:v1
// либо
sudo docker run -d -p 5000:5000 pylounge_flask_app:v1
```

Ключ `-d` предназначен для запуска вашего контейнера в фоновом режиме (необходим для того, чтобы вернуть управление терминалу, в котором вы работаете).

- `i` интерактивный режим
- `t` подключает виртуальный терминал

Ключ `-p` заставит **Docker Machine** пробрасывать подключения, приходящие на порт 5000 внешнего адреса Docker Machine на порт 5000 контейнера, на котором будет слушать подключения ваше Flask приложение.

Проверим, что контейнер успешно запущен:

```
docker ps

CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
a09f0f91dbe5      pylounge_flask_app:v1   "python3 app.py"   4 seconds ago     Up 2 seconds      0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   quirky.
```

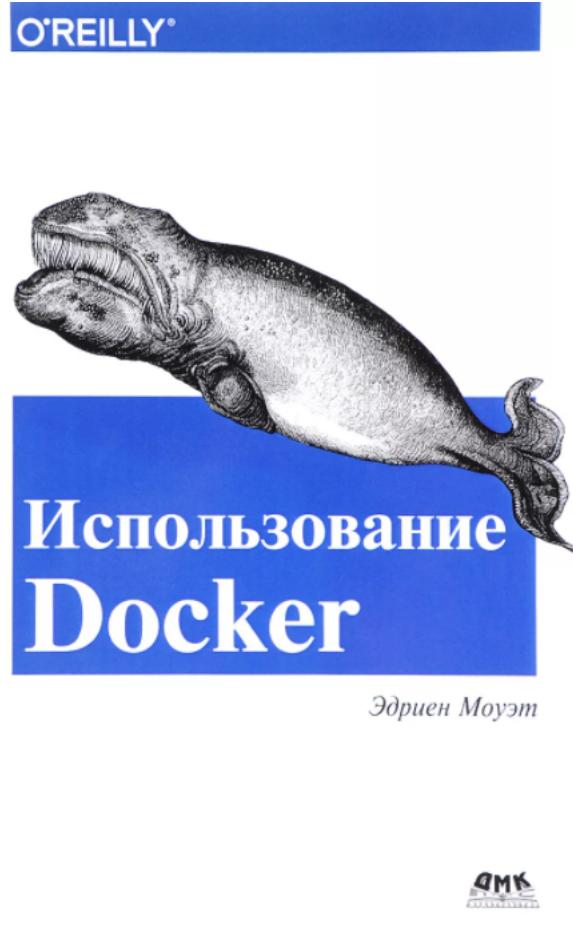
После чего в браузере перейдём по адресу <http://127.0.0.1:5000/>

Таким образом наш контейнер готов. Теперь мы можем разворачивать наше приложения на различных компьютерах, не боясь «ада зависимостей», а также выложить образ на Docker Hub.

Полезные ресурсы

- Документация Docker: <https://docs.docker.com>
- Туториал DigitalOcean по Docker: <https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04-ru>
- Курс Владилена Минина по Docker: <https://www.youtube.com/watch?v=n9uCgUzfeRQ>
- Плейлист Docker уроки от А до Я: <https://www.youtube.com/playlist?list=PLD5U-C5KK50XMCBKY0U-NLzglcRHrOwAg>
- Идеальный разбор Docker и связанных с ним технологий на Дока: <https://doka.guide/tools/docker/>
- Docker Hub: <https://hub.docker.com>
- Шпаргалка по Docker: <https://gist.github.com/wtw24/66265a5707d5feb7ed51f570db94157>

- Цикл статей по Docker на Habr: <https://habr.com/ru/company/ruvds/blog/438796/>
- Простой и наглядный гайд по Docker, представленный в игровой форме: <https://badcode.ru/docker-tutorial-dlia-novichkov-rassmatrivaem-docker-tak-iesli-by-on-by-lighrovoi-pristavkoj/>
- Книга “Использование Docker” Моут Эдриен



Заключение

Таким образом, мы разобрались по понятием контейнеризации, установили Docker, поработали с образами, контейнерами, а также создали собственный образ. После знакомства с основами, можно переходить к более продвинутым темам, таким как Docker-Compose, Docker Swarm, Kubernetes, Ansible и Vagrant.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. <https://wiki.merionet.ru>
2. <https://habr.com>
3. <https://techrocks.ru>
4. <https://docs.microsoft.com/ru-ru/>
5. <https://www.digitalocean.com/community/tutorials>
6. <https://www.docker.com>
7. <https://selectel.ru>

8. <https://tproger.ru/>
9. <https://www.reg.ru>
10. <https://eternalhost.net/blog>
11. <https://mcs.mail.ru/blog/>
12. <https://proglab.io>