QA
מסע

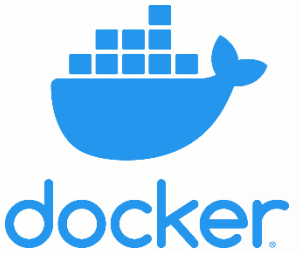
אלכס גורבצ'וב

АЛЕКСЕЙ ГОРБАЧЁВ

DOCKER

DOCKER

Docker

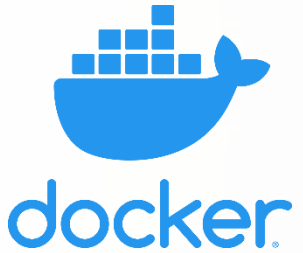


Что это и зачем нужен разработчику?

Докер - это инструмент, который позволяет разработчикам, системными администраторам и другим специалистам деплоить их приложения в песочнице (которые называются контейнерами), для запуска на целевой операционной системе, например, Linux. Ключевое преимущество Докера в том, что он позволяет пользователям упаковать приложение со всеми его зависимостями в стандартизированный модуль для разработки.

Стандарт в индустрии на сегодняшний день - это использовать виртуальные машины для запуска приложений. Виртуальные машины запускают приложения внутри гостевой операционной системы, которая работает на виртуальном железе основной операционной системы сервера.

Docker



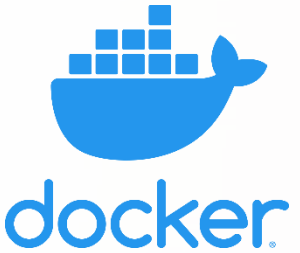
Что это и зачем нужен разработчику?

Виртуальные машины отлично подходят для полной изоляции процесса для приложения: никакие проблемы гостевой ОС не могут повлиять на софт, основной операционной системы.

Что такое контейнер?

Контейнеры по идее схожи с ВМ, однако используют иной подход благодаря которому имеют меньшую нагрузку, необходимую для виртуализации железа гостевой ОСrun-in user mode.

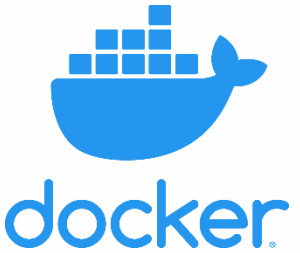
Docker



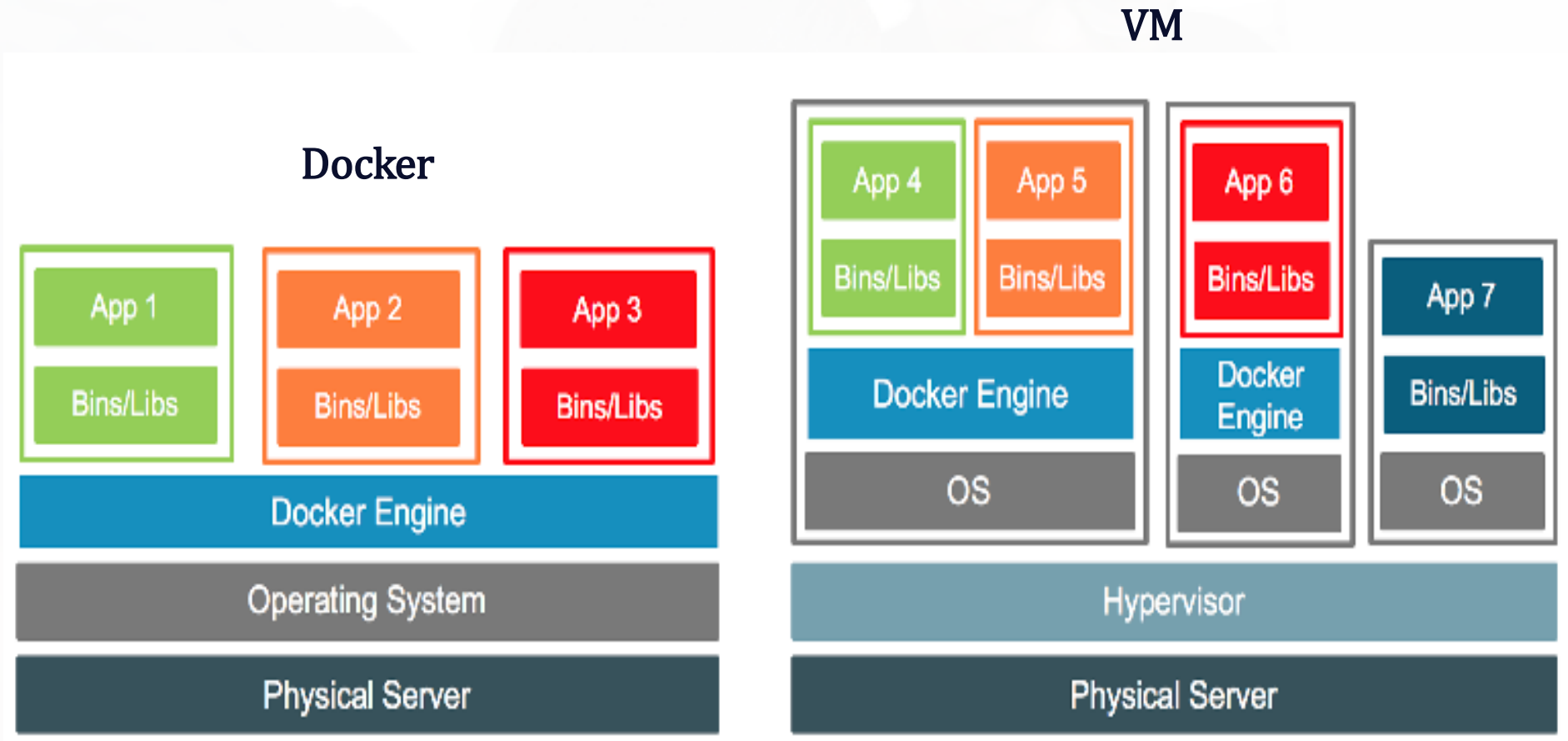
Плюсы использования Docker:

1. Изолированный запуск приложений в контейнерах.
2. Упрощение разработки, тестирования и деплоя приложений.
3. Отсутствие необходимости конфигурировать среду для запуска - она поставляется вместе с приложением - в контейнере.
4. Упрощает масштабируемость приложений и управление их работой с помощью систем оркестрации контейнеров.

Docker



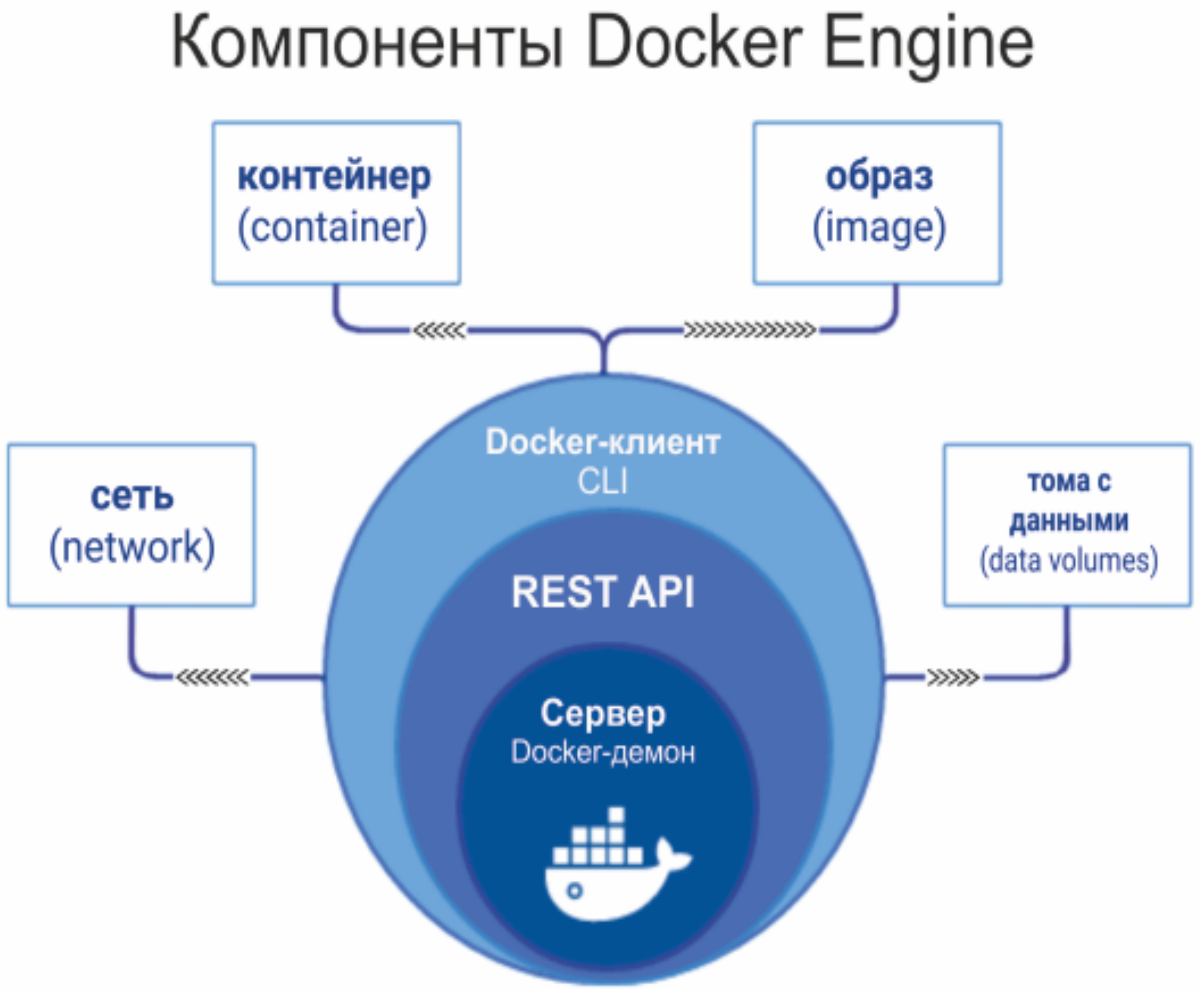
Чем отличается от Виртуальных Машин?



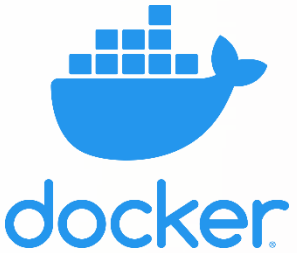
Docker

Из чего состоит?

1. Образы
2. Контейнеры
3. Volumes
4. Networks

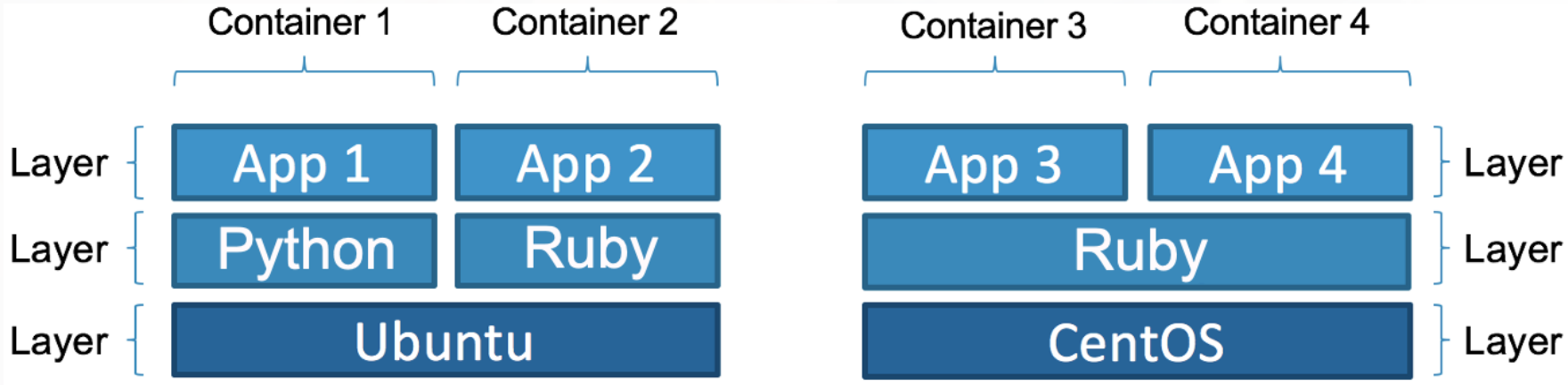


Docker

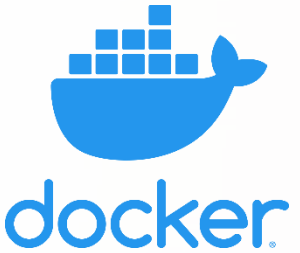


Образ

- Read-only шаблон с набором инструкций, предназначенных для создания контейнера
- Образ состоит из неизменяемых слоев, каждый из которых добавляет/удаляет/изменяет файлы из предыдущего слоя.
- Неизменяемость слоев позволяет их использовать совместно в разных образах.

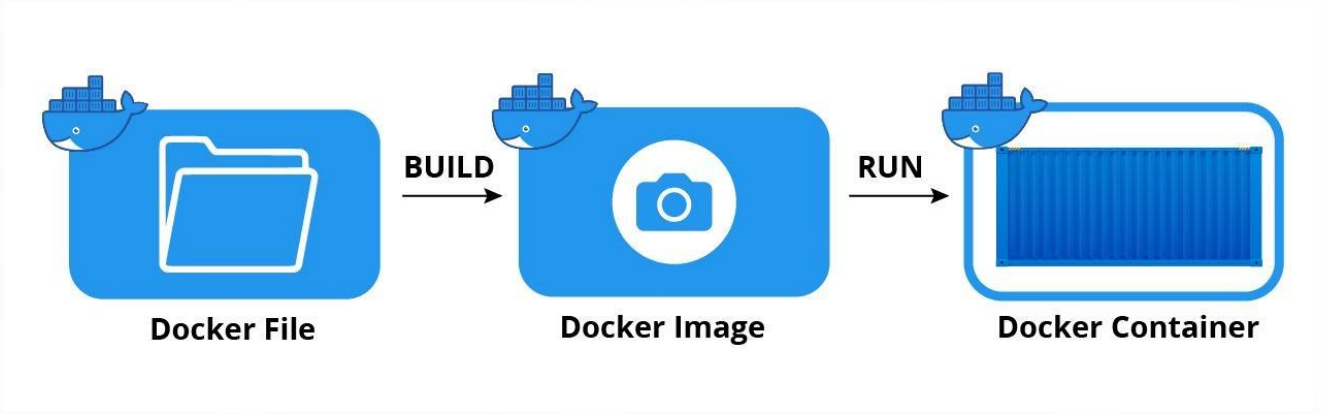


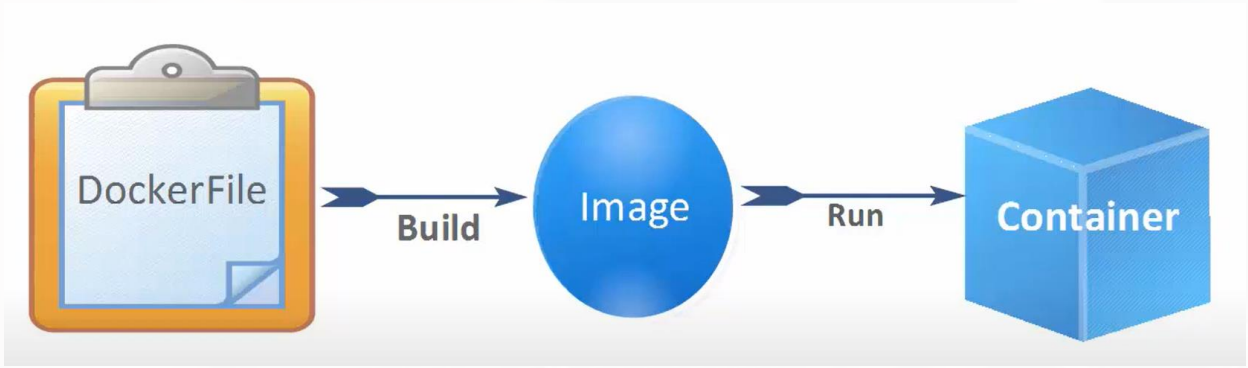
Docker



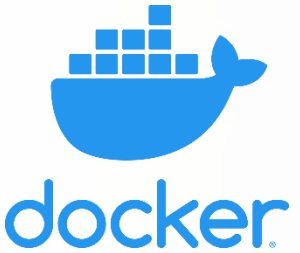
Контейнеры

Контейнер - это запущенный образ.





Docker



Установка Docker

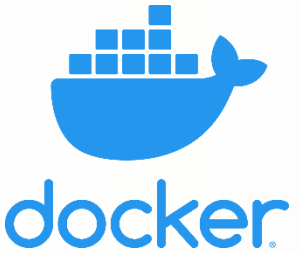
Скачиваем установщик Docker (Docker Desktop Installer) с Docker Hub.

Установка Docker Desktop включает Docker Engine, Docker CLI client, Docker Compose, Notary, Kubernetes и Credential Helper. Контейнеры и образы, созданные с помощью Docker Desktop, используются всеми учетными записями пользователей на компьютерах, на которых он установлен. Это связано с тем, что все учетные записи Windows используют одну и ту же виртуальную машину для создания и запуска контейнеров.

Запускаем установщик **Docker Desktop Installer.exe** и ожидаем пока он скачает все необходимые компоненты.

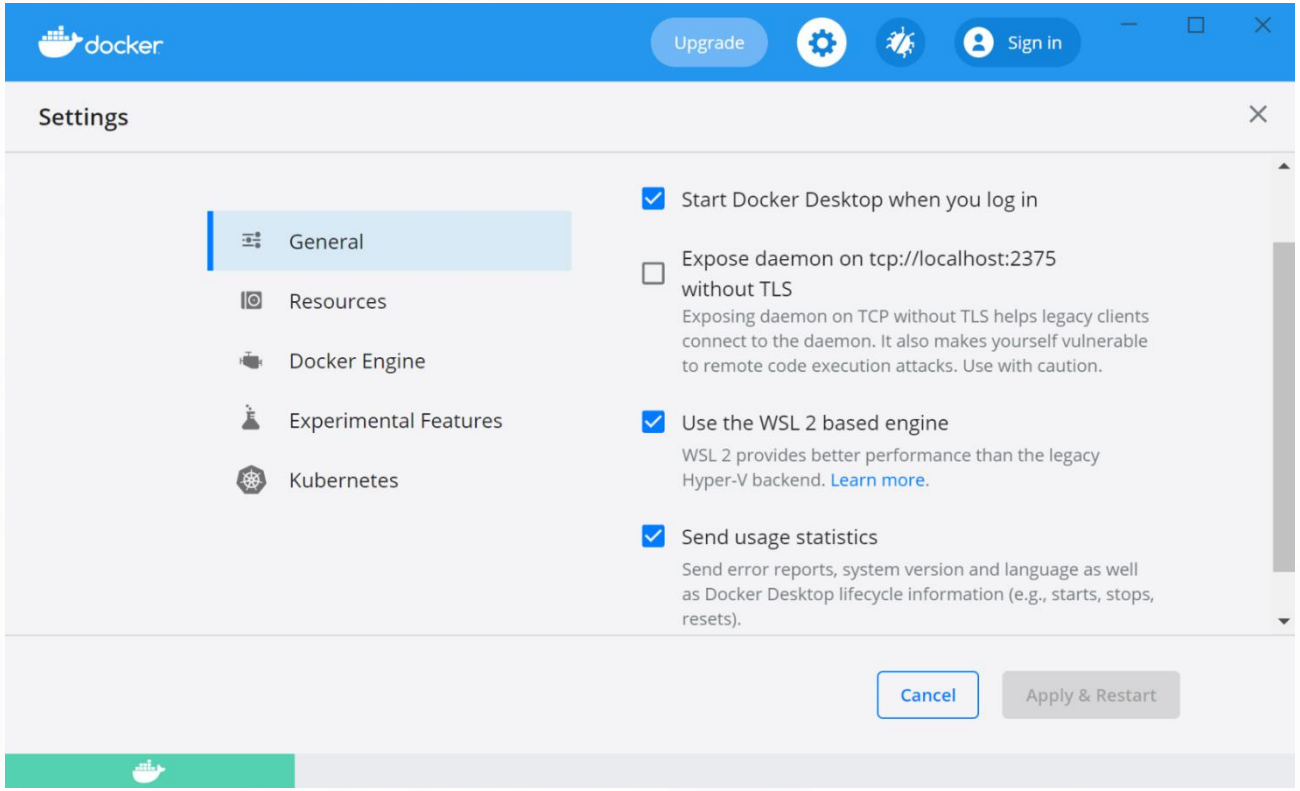
После установки система потребует перезагрузки. Перезагружаемся и входим в систему.

Docker

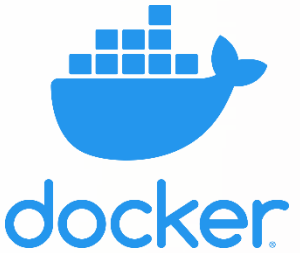


Настройка и запуск приложения

Входим в систему и ждем запуска всех служб Docker. Когда все службы будут запущены, мы увидим в трее классический значок Docker - это значит что служба установлена и запущена. Далее можно запустить приложение Docker Desktop. Далее можно изменить настройки Docker при необходимости:



Docker

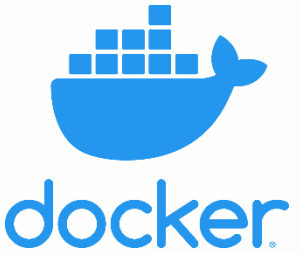


Настройка и запуск приложения

Далее управление Docker выполняется через Powershell.
Проверяем версию и выполняем тестовый запуск контейнера:

```
PS C:\Users\vtkac> docker version
Client: Docker Engine - Community
 Cloud integration: 1.0.7
 Version:           20.10.2
 API version:       1.41
 Go version:        go1.13.15
 Git commit:        2291f61
 Built:             Mon Dec 28 16:14:16 2020
 OS/Arch:           windows/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.2
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.13.15
  Git commit:       8891c58
  Built:            Mon Dec 28 16:15:28 2020
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.4.3
  GitCommit:        269548fa27e0089a8b8278fc4fc781d7f65a939b
 runc:
  Version:          1.0.0-rc92
  GitCommit:        ff819c7e9184c13b7c2607fe6c30ae19403a7aff
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
PS C:\Users\vtkac>
```

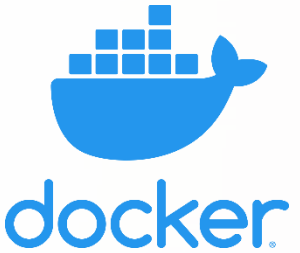
Docker



Основные команды Docker:

docker build -t *app-name*:latest . - *build from Dockerfile*
docker images - *show all images*
docker ps -a - *show all containers*
docker run -it --name *container_name app-name*:latest
docker run --rm -p 5000:5000 *app-name*:latest - *run container with port*
docker search *img_name*
docker pull *app-name*:latest
docker kill *container_name - kill container*
docker rm *container_name - delete container*
docker rmi *img_name - delete image*
docker push *username/app-name :tagname*
docker tag *old_user/old_app-name username/app-name:tagname*
docker login *-u "username" -p "password"*

Docker



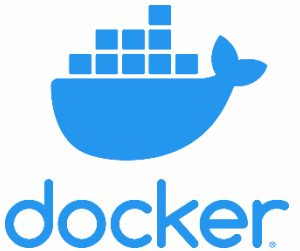
Пример

docker run --rm –p 8888:80 nginx

Связывает локальный порт хостовой машины с портом приложения в контейнере

docker run --rm nginx
docker run --rm –p 8888:80 nginx

Docker



Что делает Docker при запуске образа?

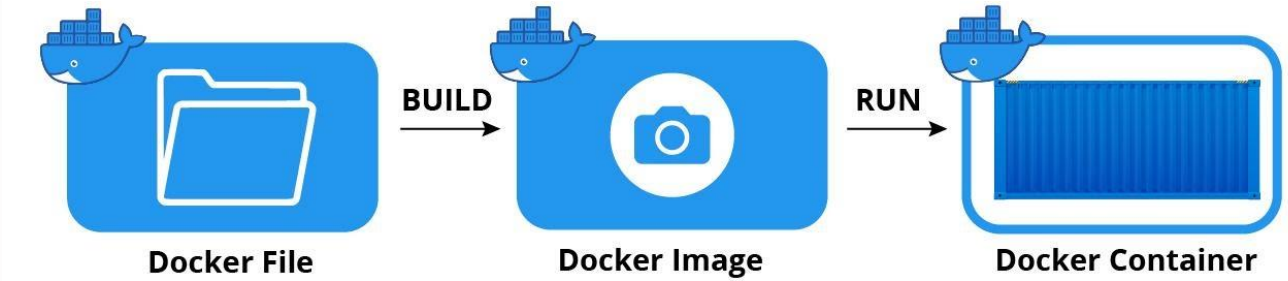
docker run --rm –p 8888:80 nginx

- скачивает образ
- создает контейнер
- инициализирует файловую систему и монтирует Read-only образ
- инициализирует сеть/мост
- запускает указанный процесс
- обрабатывает и выдает вывод приложения

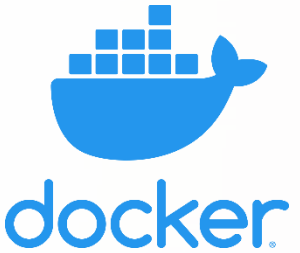
Docker

Dockerfile

Dockerfile - инструкции для подготовки образа



Docker



Dockerfile Пример

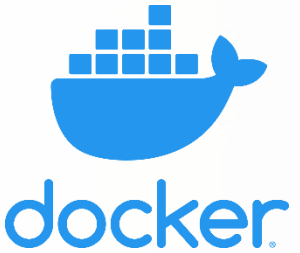
docker build -t *app-name*:latest .

```
ARG VERSION = 20.04
FROM ubuntu:${VERSION}

RUN apt-get update -y

CMD ["bash"]
```

```
ARG VERSION = latest
FROM ubuntu:${VERSION}

RUN apt-get update -y

CMD ["bash"]
```

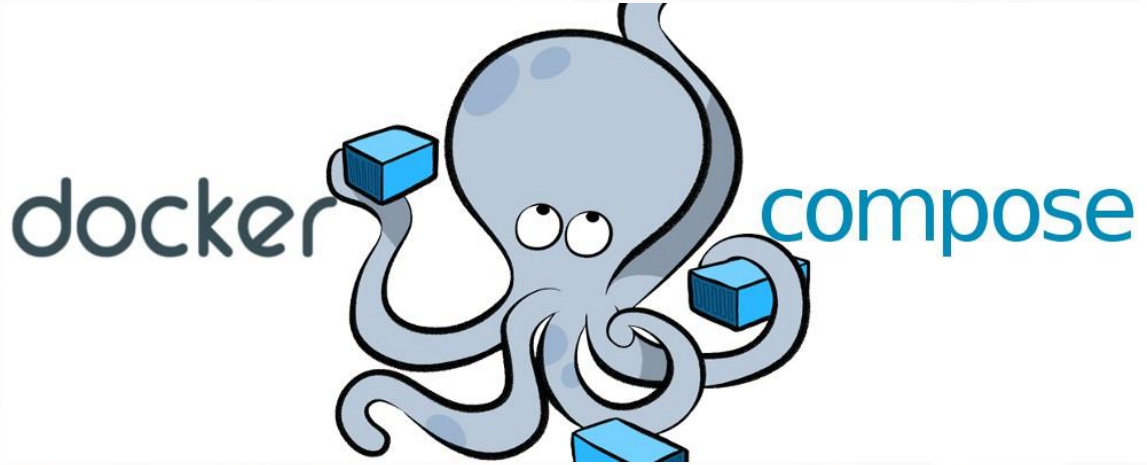
```
FROM python:3.7

RUN mkdir /app
WORKDIR /app
ADD . /app/
RUN pip install -r requirements.txt

EXPOSE 8080
CMD ["python", "/app/app.py"]
```

Docker

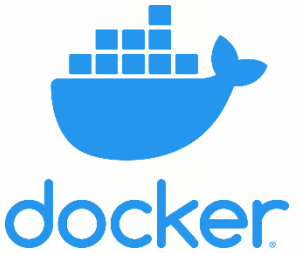


Docker Compose

Docker Compose – инструментальное средство, предназначенное для решения задач, связанных с развёртыванием проектов



Docker

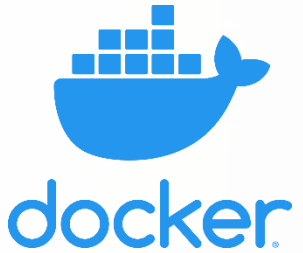


Example

1. Create a directory for the project
2. Create a file app.py:

```
import time
import redis
from flask import Flask

app = Flask(__name__)
cache = redis.Redis(host='redis', port=6379)

def get_hit_count():
    retries = 5
    while True:
        try:
            return cache.incr('hits')
        except redis.exceptions.ConnectionError as exc:
            if retries == 0:
                raise exc
            retries -= 1
            time.sleep(0.5)

@app.route('/')
def hello():
    count = get_hit_count()
    return 'Hello World! I have been seen {} times.\n'.format(count)
```

Docker



Example

3. Create a file requirements.txt:

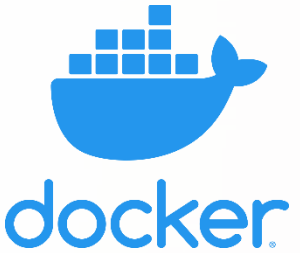
```
flask
redis
```

4. Create a Dockerfile:

```
FROM python:3.7-alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev linux-headers
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
EXPOSE 5000
COPY . .
CMD ["flask", "run"]
```

- Build an image starting with the Python 3.7 image.
- Set the working directory to `/code`.
- Set environment variables used by the `flask` command.
- Install gcc and other dependencies
- Copy `requirements.txt` and install the Python dependencies.
- Add metadata to the image to describe that the container is listening on port 5000
- Copy the current directory `.` in the project to the workdir `.` in the image.
- Set the default command for the container to `flask run`.

Docker



Example

5. Create a file docker-compose.yml (compose file defines two services: **web** and **redis**):

```
version: "3.9"
services:
  web:
    build: .
    ports:
      - "5000:5000"
  redis:
    image: "redis:alpine"
```

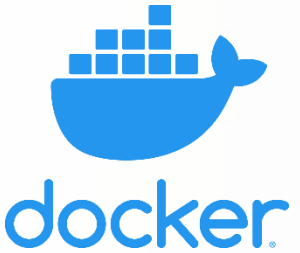
Web service

The web service uses an image that's built from the Dockerfile in the current directory. It then binds the container and the host machine to the exposed port, 5000. This example service uses the default port for the Flask web server, 5000.

Redis service

The redis service uses a public Redis image pulled from the Docker Hub registry.

Docker



Example

6. Build and run your app with Compose:
   From your project directory, start up your application by running
   docker-compose up

   Compose pulls a Redis image, builds an image for your code, and starts the services you defined. In this case, the code is statically copied into the image at build time.

7. Enter **http://localhost:5000/ (or http://127.0.0.1:5000)** in a browser to see the application running
8. Refresh the page.
9. Run **docker image ls** in terminal.

Thanks for your time ☺