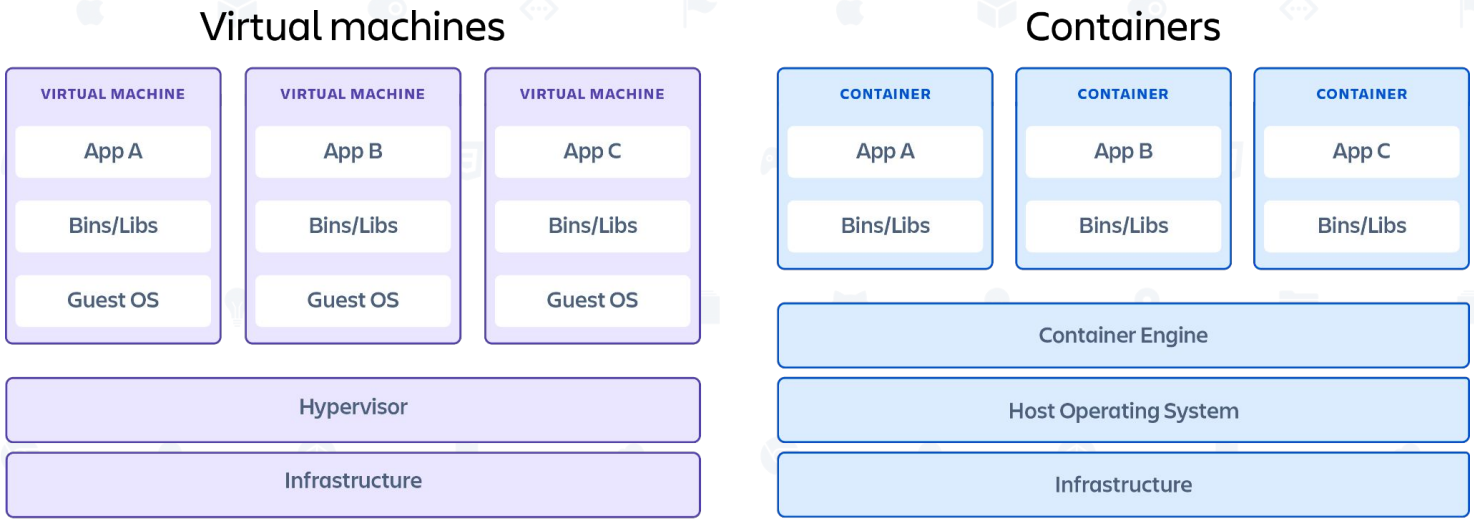
Lesson 38

09.03.2023

Контейнеры и виртуальные машины — очень похожие между собой технологии виртуализации ресурсов. Виртуализация — это процесс, при котором один системный ресурс, такой как оперативная память, ЦП, диск или сеть, может быть виртуализирован и представлен в виде множества ресурсов. Основное различие контейнеров и виртуальных машин заключается в том, что виртуальные машины виртуализируют весь компьютер вплоть до аппаратных уровней, а контейнеры — только программные уровни выше уровня операционной системы.




ORACLE
VirtualBox


vmware


Microsoft
Hyper-V


docker

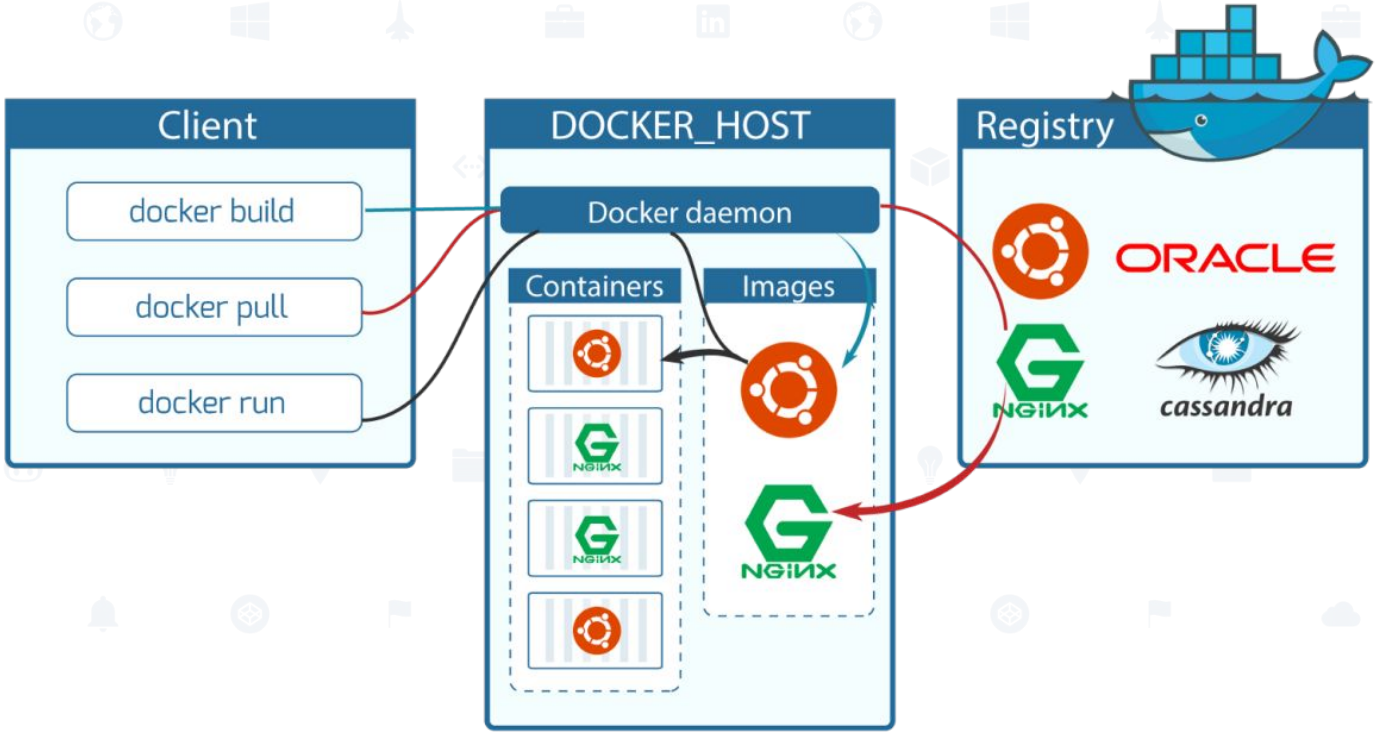
Docker — программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации, контейнеризатор приложений.

В своем ядре docker позволяет запускать практически любое приложение, безопасно изолированное в контейнере. Безопасная изоляция позволяет вам запускать на одном хосте много контейнеров одновременно. Легковесная природа контейнера, который запускается без дополнительной нагрузки гипервизора, позволяет вам добиваться больше от вашего железа

Платформа и средства контейнерной виртуализации могут быть полезны в следующих случаях:

- упаковывание вашего приложения (и так же используемых компонент) в docker контейнеры;
- раздача и доставка этих контейнеров вашим командам для разработки и тестирования;
- выкладывания этих контейнеров на ваши продакшены, как в дата центры так и в облака.

DOCKER COMPONENTS



Docker-демон

Как показано на диаграмме, демон запускается на хост-машине. Пользователь не взаимодействует с сервером на прямую, а использует для этого клиент.

Docker-клиент

Docker-клиент, программа docker — главный интерфейс к Docker. Она получает команды от пользователя и взаимодействует с docker-демоном.

Внутри docker-а

Чтобы понимать, из чего состоит docker, вам нужно знать о трех компонентах:

- образы (images)
- реестр (registries)
- контейнеры

Образы

Docker-образ — это read-only шаблон. Например, образ может содержать операционку Ubuntu с Apache и приложением на ней. Образы используются для создания контейнеров. Docker позволяет легко создавать новые образы, обновлять существующие, или вы можете скачать образы созданные другими людьми. Образы — это компонента сборки docker-а.

Реестр

Docker-реестр хранит образы. Есть публичные и приватные реестры, из которых можно скачать либо загрузить образы. Публичный Docker-реестр — это Docker Hub. Там хранится огромная коллекция образов. Как вы знаете, образы могут быть созданы вами или вы можете использовать образы созданные другими. Реестры — это компонента распространения.

Контейнеры

Контейнеры похожи на директории. В контейнерах содержится все, что нужно для работы приложения. Каждый контейнер создается из образа. Контейнеры могут быть созданы, запущены, остановлены, перенесены или удалены. Каждый контейнер изолирован и является безопасной платформой для приложения. Контейнеры — это компонента работы.

Файл Dockerfile

Файл Dockerfile содержит набор инструкций, следуя которым Docker будет собирать образ контейнера. Этот файл содержит описание базового образа, который будет представлять собой исходный слой образа. Среди популярных официальных базовых образов можно отметить python, ubuntu, alpine.

В образ контейнера, поверх базового образа, можно добавлять дополнительные слои. Делается это в соответствии с инструкциями из Dockerfile. Например, если Dockerfile описывает образ, который планируется использовать для решения задач машинного обучения, то в нём могут быть инструкции для включения в промежуточный слой такого образа библиотек NumPy, Pandas и Scikit-learn.

И, наконец, в образе может содержаться, поверх всех остальных, ещё один тонкий слой, данные, хранящиеся в котором, поддаются изменению. Это — небольшой по объёму слой, содержащий программу, которую планируется запускать в контейнере.

Docker Compose

Docker Compose — это инструмент, который упрощает развёртывание приложений, для работы которых требуется несколько контейнеров Docker. Docker Compose позволяет выполнять команды, описываемые в файле docker-compose.yml. Эти команды можно выполнять столько раз, сколько потребуется. Интерфейс командной строки Docker Compose упрощает взаимодействие с многоконтейнерными приложениями. Этот инструмент устанавливается при установке Docker.

Дюжина инструкций Dockerfile

1. FROM — задаёт базовый (родительский) образ.
2. LABEL — описывает метаданные. Например — сведения о том, кто создал и поддерживает образ.
3. ENV — устанавливает постоянные переменные среды.
4. RUN — выполняет команду и создаёт слой образа. Используется для установки в контейнер пакетов.
5. COPY — копирует в контейнер файлы и папки.
6. ADD — копирует файлы и папки в контейнер, может распаковывать локальные .tar-файлы.
7. CMD — описывает команду с аргументами, которую нужно выполнить когда контейнер будет запущен. Аргументы могут быть переопределены при запуске контейнера. В файле может присутствовать лишь одна инструкция CMD.
8. WORKDIR — задаёт рабочую директорию для следующей инструкции.
9. ARG — задаёт переменные для передачи Docker во время сборки образа.
10. ENTRYPOINT — предоставляет команду с аргументами для вызова во время выполнения контейнера. Аргументы не переопределяются.
11. EXPOSE — указывает на необходимость открыть порт.
12. VOLUME — создаёт точку монтирования для работы с постоянным хранилищем

Dockerfile_app

```
FROM openjdk:17
MAINTAINER oStepurko
EXPOSE 8080
COPY target/ExhangerMarket-spring-boot.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]

```

Dockerfile_mysql

```
FROM mysql:latest
MAINTAINER oStepurko
EXPOSE 3306
ENV MYSQL_DATABASE=exhangeMarket \
    MYSQL_ROOT_PASSWORD=rootroot

ADD init.sql /docker-entrypoint-initdb.d
```

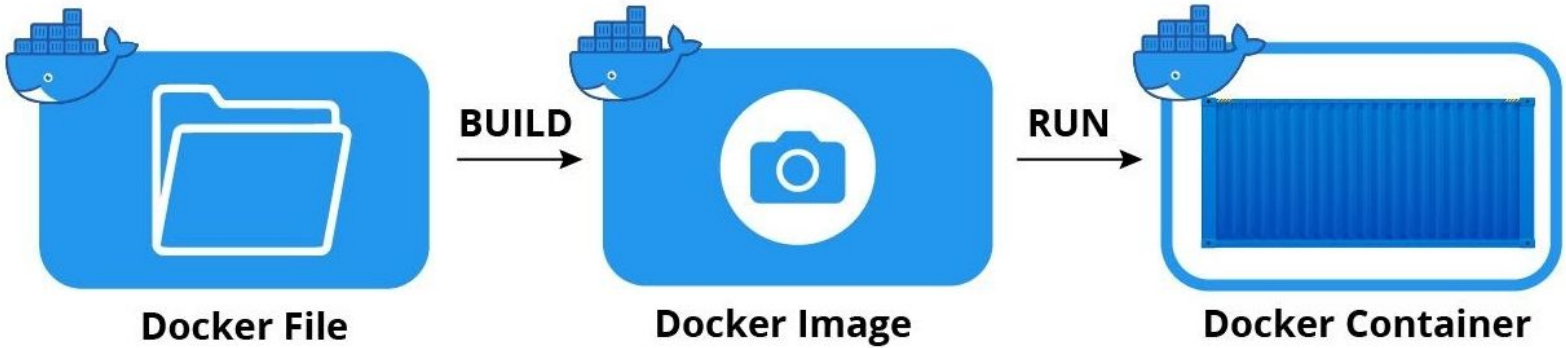
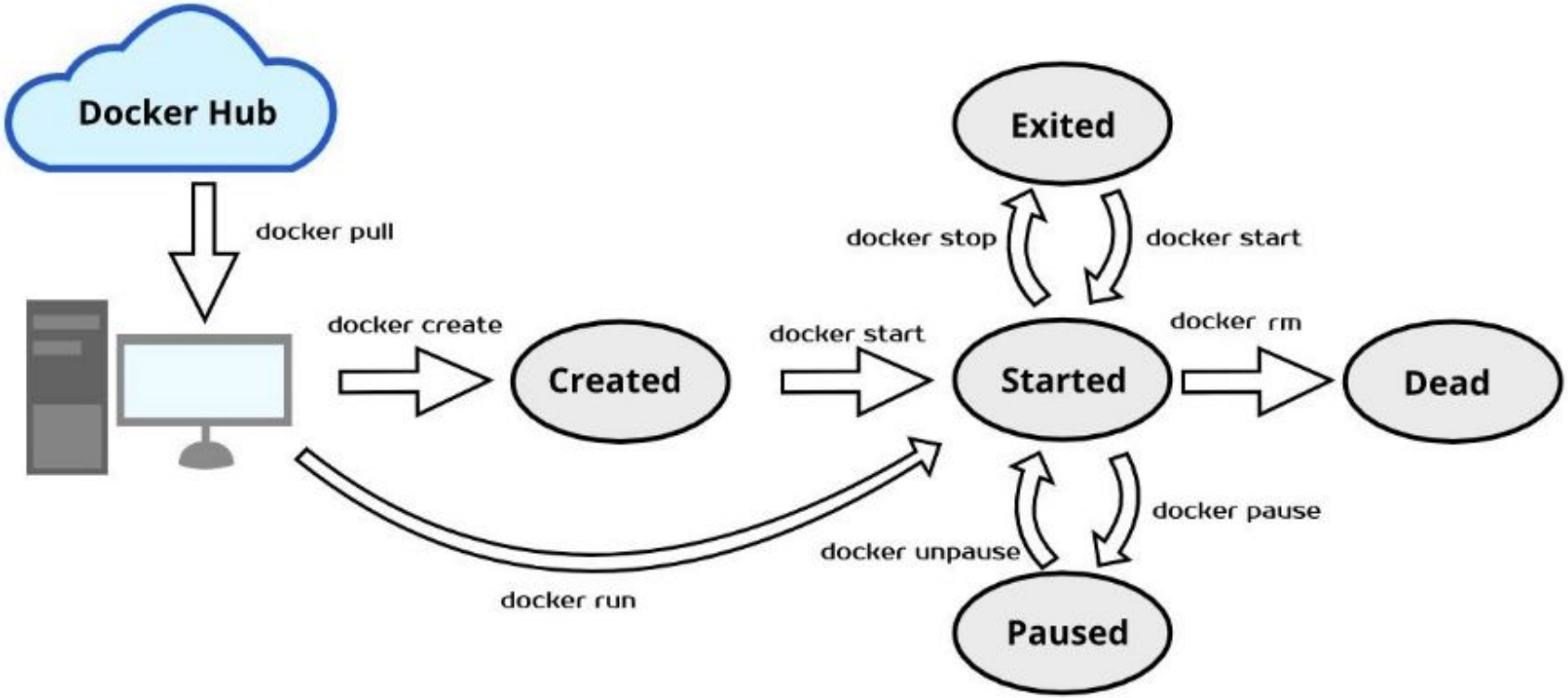

BUILD
RUN
Docker File
Docker Image
Docker Container

Схема Lifecycle of Docker Container



Create container

Создайте контейнер, чтобы в дальнейшем запустить его с нужным образом.

```
docker create --name <container-name> <image-name>
```

Run docker container

Запустите контейнер докера с требуемым образом и указанной командой / процессом. Флаг **-d** используется для запуска контейнера в фоновом режиме.

```
docker run -it -d --name <container-name> <image-name> bash
```

Pause container

Используется для приостановки процессов, запущенных внутри контейнера.

```
docker pause <container-id/name>
```

Unpause container

Используется для возобновления процессов внутри контейнера.

```
docker unpause <container-id/name>
```

Start container

Запустите контейнер, если он находится в остановленном состоянии.

```
docker start <container-id/name>
```

Stop container

Остановить контейнер и процессы, запущенные внутри контейнера:

```
docker stop <container-id/name>
```

Чтобы остановить все запущенные контейнеры докеров

```
docker stop $(docker ps -a -q)
```

Restart container

Используется для перезапуска контейнера, а также процессов, работающих внутри контейнера.

```
docker restart <container-id/name>
```

Kill container

Мы можем убить работающий контейнер.

```
docker kill <container-id/name>
```

Destroy container

Лучше уничтожать контейнер, только если он находится в остановленном состоянии, вместо того, чтобы принудительно уничтожать работающий контейнер.

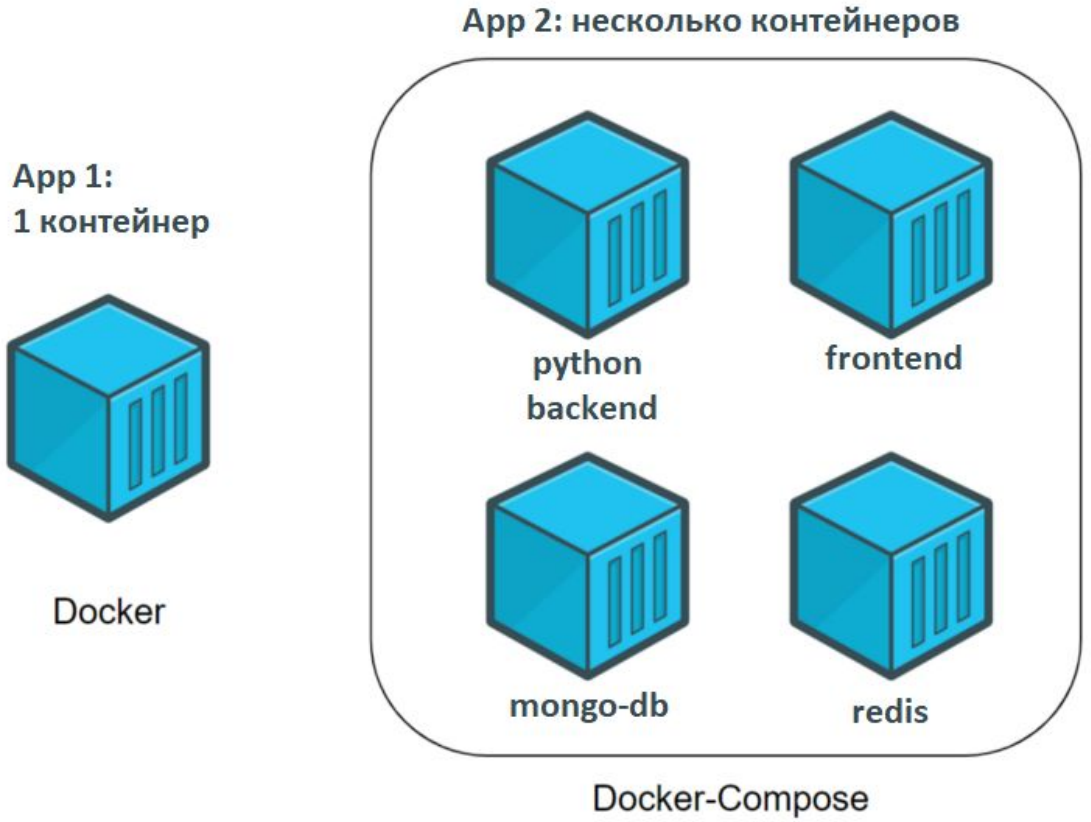
```
docker rm <container-id/name>
```

Чтобы удалить все остановленные контейнеры докеров

```
docker rm $ (docker ps -q -f status = exited)
```

Что такое Docker Compose?

Docker Compose — это инструмент, который упрощает запуск приложений, состоящих из нескольких контейнеров.



Docker Compose позволяет записывать команды в docker-compose.yml файл для повторного использования. Интерфейс командной строки Docker Compose (cli) упрощает взаимодействие с вашим многоконтейнерным приложением.

docker-compose.yml

```yaml
version: "3.7"
services:
  ex-market:
    depends_on:
      - mysqldb
    command: sh -c './wait-for mysqldb:3306 -- npm start'
    build:
      context: .
      dockerfile: Dockerfile_app
    ports:
      - 8080:8080
    links:
      - "mysqldb:app_db"
  mysqldb:
    build:
      context: .
      dockerfile: Dockerfile_mysql
    ports:
      - 3306:3306
    env_file: .env
    volumes:
      - myapp:/home/node/app
volumes:
  myapp:
```

Схема работы Docker-Compose: