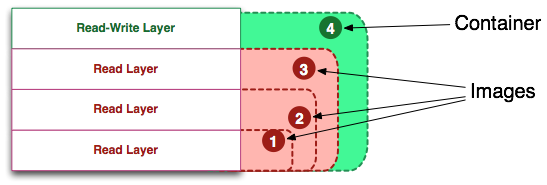
**Информация по работе с Docker**

Docker представляет собой стандартное решение для задач сборки, тестирования и поставки приложения клиенту, программное обеспечение для автоматизации развёртывания и управления приложениями в среде виртуализации на уровне операционной системы. Так или иначе задеплоить можно и без контейнера, для этого можно использовать следующие решения:

1. *bash-скрипт*, который делает сборку, создает виртуальное окружение для проекта, докачивает недостающие зависимости и запускает, в конечном счёте, код. Данный инструмент использовался до 2009, когда методология DevOps ещё только зарождалась в умах разработчиков, однако на практике не хватало инструментов по автоматизации.
2. *гипервизор и виртуальные машины*, которые позволяет создать изоляцию сетей и запустить приложение в кластерах. Сам по себе докер представляет собой виртуальную машину, где виртуализируется только пространство имён, в случае с виртуальной машиной виртуализация происходит и на уровне железа, и на уровне ядра системы, и на уровне процессов, и на уровне сетей. Виртуализация бывает полной, бывает паравиртуализация, однако что всегда отличает её от контейнеризации -- это большие затраты по техническим ресурсам. Существует значительная вычислительная нагрузка, необходимая для виртуализации железа гостевой ОС.

Контейнеры используют другой подход: они предоставляют схожий с виртуальными машинами уровень изоляции, но благодаря правильному задействованию низкоуровневых механизмов основной операционной системы делают это с в разы меньшей нагрузкой.

Подобно виртуальной машине докер запускает свои процессы в собственной, заранее настроенной операционной системе. Но при этом все процессы докера работают на физическом host сервере деля все процессоры и всю доступную память со всеми другими процессами, запущенными в host системе. Подход, используемый докером находится посередине между запуском всего на физическом сервере и полной виртуализацией, предлагаемой виртуальными машинами. Этот подход называется контейнеризацией.



Слои – это результат создания образов Docker. Каждый шаг в Docker-файле создает новый “слой”, который, по сути, представляет собой дифф изменений файловой системы с момента последнего шага + метаданные. Это можно представить как промежуточные изменения в гит git: вы добавляете изменение файла, затем еще одно, затем еще одно… **Слои необходимо добавлять таким образом, чтобы модифицировать как можно меньше файлов, поскольку в ином случае слои Docker займут больше места на диске.**

**Ошибка:**

RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone

RUN apt-get update -y

RUN apt-get install -y python3-pip python3.10 build-essential libgl1-mesa-glx ffmpeg libsm6 libxext6

**Приемлемый вариант:**

RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone &&

apt-get update -y &&

apt-get install -y python3-pip python3.10 build-essential libgl1-mesa-glx ffmpeg libsm6 libxext6

Кроме того, **важно учитывать последовательность слоёв при сборке**, дабы избегать ситуаций, когда необходимо пересобрать только кодовую часть, сам процесс создания контейнера запускается с загрузки зависимостей/их установки, или необходимо изменить только порт и entrypoint.

**Ошибка:**

EXPOSE 5000

COPY . .

CMD [ "run", "--host", "0.0.0.0"]

**Приемлемый вариант:**

COPY . .

EXPOSE 5000

CMD [ "run", "--host", "0.0.0.0"]

В идеальном случае желательно выносить все интерфейсы (volume, network) и сам запуск в файл docker-compose.yml.

| **Реализация в Docker** | **Реализация в docker-compose** |
| --- | --- |
| EXPOSE 5000 | expose:  -- "5000" |
| CMD [ "python3", "-m" , "flask", "run", "--host=0.0.0.0", "--port=5000"] | command: "flask run --host=0.0.0.0 --port=5000" |
| PORTS | ports:  - "5000:5000" |
| VOLUME ["/home/data/datasets"] | volume:  -- datasets:/home/data/datasets  ...  volumes:  datasets:  driver: local  driver\_opts:  o: bind  type: none  device: /home/data/datasets/ |

В идеальном варианте желательно не использовать сети докер, а создавать собственные, дабы была возможность запустить приложение в рамках одной машины в несколько экземпляров:

Сам по себе сервис:

web:

networks:

test:

ipv4\_address:

10.1.0.100

networks:

test:

driver: bridge

ipam:

driver: default

config:

- subnet: 10.1.0.0/24

Аналогично:

docker network create --subnet 10.5.0.0/24 local\_network

networks:

default:

external:

name: local\_network

Сам по себе сервис:

web:

networks:

default:

ipv4\_address: 10.5.0.11

| При необходимости, когда сети docker вообще не нужны, можно установить network\_mode: "host". |
| --- |

**Структура проекта:**

.

..

./scripts

./core/ # -- пакеты и модули, которые можно переиспользовать

./app.py

./deps/

./requirements.txt

./wheels/\*.whl

./utils/\*.tar.gz

Dockerfile

docker-compose.yml

Всё, что относится к коду, хранится в scripts, это то тот код, который мы разрабатываем. Всё, что относится к стороннему, но используемому в проекте софту, должно быть в ./deps. Сборка образа должна проходить в два этапа:

* копирование зависимостей и их установка (сборка зависимостей);
* копирование файлов ./scripts.

Это необходимо для того, чтобы упростить процесс запуска образа после внесённых изменений. Зависимости проекта могут передаваться в виде сборки.