VIRTUALISATION & DOCKER

Виртуальные машины — незаменимый помощник тестировщика, т.к. позволяют вам:

- создать виртуальную машину с нужными характеристиками: протестировать работу приложения в условиях ограниченных ресурсов;
- установить нужный набор ОС и софта на виртуальную машину: при этом он (набор) никак не будет «замусоривать» и влиять* на вашу хостовую систему;
- установить множество виртуальных машин на одной физической: вы сможете промоделировать работу систему «по сети»;
- создавать снапшоты (моментальные снимки состояния машины): представьте, что у вас есть система-аналог Git'a, в которой можно создавать состояния, возвращаться к ним и т.д.
- возможность быстрого клонирования: один раз настроенную машину не нужно настраивать заново, просто создаёте столько копий, сколько нужно
- и это не полный перечень возможностей.

Примечание*: виртуальные машины не идеальны и содержат уязвимости, позволяющие выйти за пределы гостевой машины.

VDS (Virtual Dedicated Server)

Вы покупаете виртуальную машину на мощностях провайдера и можете

распоряжаться ею по собственному усмотрению.

Продукты, предоставляющими возможности виртуализации:

- Windows 10 Pro и выше — Hyper-V (входит в состав компонентов ОС)
- Windows, Linux, MacOS — VirtualBox

Системы контейнеризации существуют достаточно давно, но наибольшее

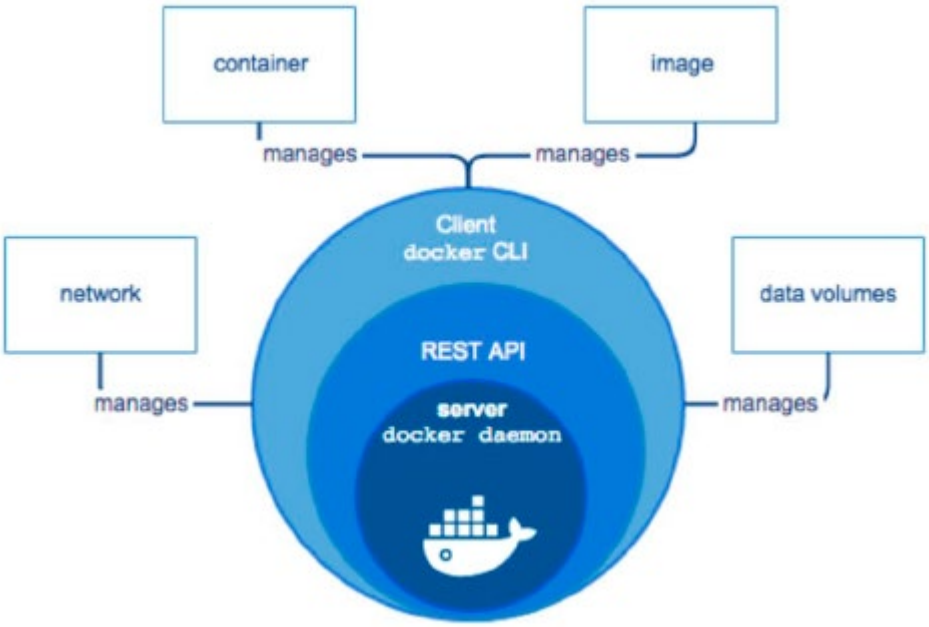
распространение в настоящий момент получил Docker.

Docker имеет клиент-серверную архитектуру (т.е. вы вполне можете подключаться к серверной части, установленной не на вашей машине).

- Image – образ, содержащий всю необходимую информацию для запуска приложения;
- Container – экземпляр запущенного образа.

Image состоит из двух частей:

- Снапшот файловой системы;
- Команда запуска.(Одна,например, java -jar app.jar

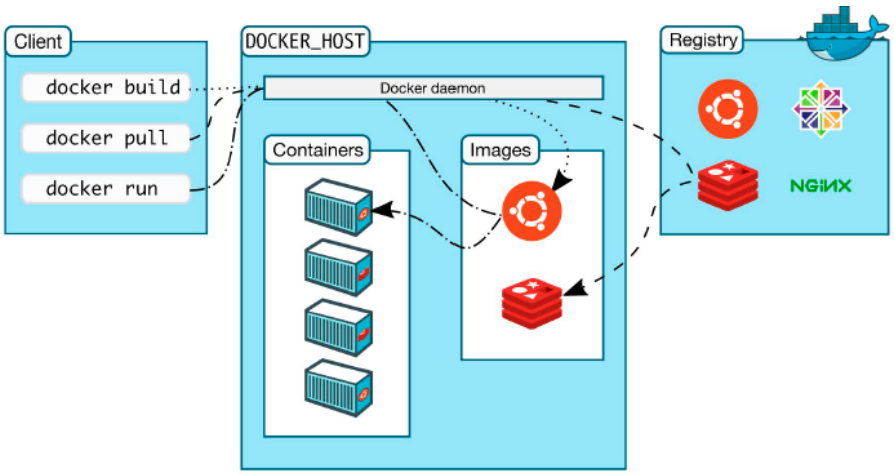


Запуск контейнера

```
# скачиваем образ с Docker Hub
$ docker image pull hello-world

Using default tag: latest
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:c3b4ada4687bbaa170745b3e4dd8ac3f194ca95b2d0518b417fb47e5879d9b5f
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest

# создаём из образа контейнер
$ docker container create --name first hello-world
# id контейнера
b4aa96b47729dbef34eee79341038733dbc1821c2b9e7b35f8915a5d2b1f7252

$ docker container start first
# или
$ docker container start b4aa96
```

Образы хранятся в реестре (публичные — в Docker Hub), соответственно, по шагам:

1. скачивается образ (если ещё не был скачан);
2. из образа создаётся контейнер;
3. контейнер запускается.



Поскольку это очень частый сценарий, есть отдельная команда, позволяющая сделать всё сразу:

```
$ docker container run --name first hello-world

docker: Error response from daemon: Conflict.
The container name "/first" is already in use by container "b4aa96..."

$ docker container run --name second hello-world
```

Т.е. вы не можете создать контейнер с тем же именем.

```
# все существующие контейнеры
$ docker container ls --all
CONTAINER ID  IMAGE        COMMAND   CREATED         STATUS      NAMES
d9e517b8c11c  hello-world  "/hello"  3 minutes ago   Exited (0)  second
b4aa96b47729  hello-world  "/hello"  14 minutes ago  Exited (0)  first

# только работающие в данный момент
$ docker container ls

$ docker image ls --all
REPOSITORY   TAG     IMAGE ID      CREATED        SIZE
hello-world  latest  fce289e99eb9  10 months ago  1.84kB
```

Команды удаления образов, контейнеров, переименования, вы можете найти с помощью справки:

```
$ docker container --help

$ docker image --help
```

Большинство образов содержат документацию по запуску, например, для

MySQL вы переходите на hub.docker.com и в поиске вбиваете MySQL.

Большая часть сервисов, упакованных в Docker образы, предоставляют

следующий набор возможностей для конфигурации:

- указание специфичных для сервисов настроек — через переменные окружения;
- указание портов для binding'a — порт, на котором работает сервис внутри контейнера,

привязывается к порту хостовой ОС*;

- указания каталога для постоянного хранения данных (Volumes) — каталог внутри контейнера привязывается к каталогу хостовой ОС (т. е. уничтожение контейнера не ведёт к уничтожению данных).

Вы уже должны знать про переменные окружения в ОС

Переменная окружения (**переменная** среды́, англ. environment variable) — текстовая **переменная операционной системы**, хранящая какую-либо информацию — например, данные о настройках системы или сведения о текущем пользователе.

В случае Docker образов они передаются при запуске контейнера через командную строку с флагом -e.

Например, для образа MySQL*:

- MYSQL_ROOT_PASSWORD;
- MYSQL_RANDOM_ROOT_PASSWORD;
- MYSQL_DATABASE;
- MYSQL_USER;
- MYSQL_PASSWORD;

Перечень допустимых переменных окружения их предназначения определяется автором образа.

```
$ docker container run -d \
  -e MYSQL_RANDOM_ROOT_PASSWORD=yes \
  -e MYSQL_DATABASE=app \
  -e MYSQL_USER=app \
  -e MYSQL_PASSWORD=9mREsvXDs9Gk89Ef \
  mysql
$ docker container ls
$ docker container stop <id>
```

Обратите внимание на флаг `-d`: некоторые контейнеры (`hello-world`) запускают команды, которые отрабатывают и завершают свою работу (работа контейнера завершается вместе с завершением этой команды).

Другие же — запускают команды, задача которых «постоянно работающий сервис» и флаг `-d` позволяет запустить такие сервисы, не «заняв» нашу консоль.

Контейнер можно остановить через `docker container stop <id>`

Сервис, работающий в контейнере (если хочет использовать сеть) должен слушать определённый порт. Флаг -p определяет binding портов (привязка порта контейнера к порту хоста):

```
$ docker container run -d \
  -e MYSQL_RANDOM_ROOT_PASSWORD=yes \
  -e MYSQL_DATABASE=app \
  -e MYSQL_USER=app \
  -e MYSQL_PASSWORD=9mREsvXDs9Gk89Ef \
  -p 3000:3306 \
  mysql
$ docker container ls
$ docker container stop <id>
```

Первым всегда пишется порт хоста, а через двоеточие — порт контейнера. Хотя обычно стараются, чтобы они совпадали, т.е.: -p 3306:3306

Не всегда нужно привязывать порт контейнера к порту хостовой машины. Например, если мы хотим организовать сетевое взаимодействие между двумя контейнерами, то bind'ить порты к хостовой машине не нужно.

Q: А что если я хочу подключиться к контейнеру и выполнить там какую-токоманду? Это возможно?

A: Да, для этого используется команда docker container exec -it<id> sh, где:

- it — флаги интерактивного режима;
- sh — запускаемая команда.

Контейнеры стартуют в так называемом неинтерактивном режиме: т.е. вы можете видеть то, что выводиться в поток вывода, но не можете никак с этим взаимодействовать (а иногда это оказывается нужным).

Кстати, флаги -it работают и в команде docker container run. Выйти можно с помощью клавиш Ctrl + D.

Q: А где же будут храниться данные? Ведь задача базы данных хранить данные, а мы говорим, что контейнеры должны быть иммутабельны.

Q: Но как я узнаю, какие порты нужно bind'ить?

A: Только из документации.

Но в хороших образах разработчики это явно указывают:

```
$ docker container ls
... PORTS
... 3306/tcp, 33060/tcp

# после binding'a

$ docker container ls
... PORTS
... 33060/tcp,0.0.0.0:3000->3306/tcp
```

A: Хранение данных будет осуществляться в хостовой системе с помощью механизма Volumes.

Фактически, контейнер живёт своей жизнью, а данные все хранятся в хостовой системе.

Для указания каталога, в котором всё будет хранится используется флаг `-v`:

```
$ docker container run -d \
  -e MYSQL_RANDOM_ROOT_PASSWORD=yes \
  -e MYSQL_DATABASE=app \
  -e MYSQL_USER=app \
  -e MYSQL_PASSWORD=9mREsvXDs9Gk89Ef \
  -p 3000:3306 \
  -v "$PWD/data":/var/lib/mysql
  mysql
$ docker container ls
$ docker container stop <id>
```

`$PWD` – это текущий каталог в Linux. В Windows нужно использовать `%cd%` для CMD, `${PWD}`.

Настоятельно рекомендуем вам освежить в памяти руководство по терминалу из курса по Git.

Docker Compose — инструмент, позволяющий запускать мультиконтейнерные приложения.

Но даже для приложений, использующих один контейнер, он позволяет здорово сэкономить время — мы можем сохранять всю конфигурацию в файле формата yml:

```yaml
version: '3.7'
services:
  mysql:
    image: mysql:8.0.18
    ports:
      - '3306:3306'
    volumes:
      - ./data:/var/lib/mysql
    environment:
      - MYSQL_RANDOM_ROOT_PASSWORD=yes
      - MYSQL_DATABASE=app
      - MYSQL_USER=app
      - MYSQL_PASSWORD=9mREsvXDs9Gk89Ef
```

Запуск всех сервисов осуществляется с помощью команды:

```
$ docker-compose up
```

А остановка (в том же каталоге) с помощью команды:

```
$ docker-compose down
```

Удаление остановленных контейнеров:

```
$ docker-compose rm
```

Полная справка по параметрам командной строки находится на официальной странице.

Q: Хорошо, а что если я хочу определённую версию сервиса?

A: Для этого служат теги, вы можете найти все поддерживаемые теги на странице образа (по умолчанию всегда используется `latest` — т.е. `mysql:latest`):

Например, можно указать: `mysql:8.0.18`.

Осталась последняя часть: подключить наше приложение к БД.

Разработчики предоставили нам следующую инструкцию:

```
Рядом с jar'ником нужно положить файл application.properties,
в котором прописать строку подключения в формате:

spring.datasource.url=jdbc:mysql://localhost:3306/db
spring.datasource.username=user
spring.datasource.password=password
```

- `mysql` – тип БД;
- `localhost` – хост БД;
- `3306` – порт;
- `db` – имя БД;
- `user` – пользователь;
- `password` – пароль.

Прописываем соответствующие значения и запускаем: `java -jar db-api.jar`

Удостоверяемся, что при заходе на страницу /api/cards в формате JSON выдаётся список карт:

```
[
  {
     "id":1,
     "name":"Альфа-Карта Premium",
     "description":"Альфа-Карта вернёт ваши деньги",
     "imageUrl":"/alfa-card-premium.png"
  },
  ...
]
```

Про развёртывание Node.js приложения мы поговорим в рамках ДЗ.

В рамках современных подходов разработки был сформулирован термин, который содержит «лучшие» подходы — 12 factor app.

Ключевое: указанные подходы позволяют не только быстро разрабатывать и разворачивать приложения, но и, самое главное для нас, удобно тестировать и интегрировать весь процесс в CI/CD.

Давайте ознакомимся с ключевыми моментами на примере русскоязычного перевода: https://12factor.net/ru/.

Библиотека TestContainers позволяет вам прямо из тестов создавать

контейнеры и управлять ими