нетология

DOCKER & DOCKER COMPOSE

Оксана Мельникова

ОКСАНА МЕЛЬНИКОВА

Software testing engineer

Оксана Мельникова

План занятия

1. Задача
2. Виртуализация
3. Контейнеризация
4. Docker
5. Docker Compose
6. 12 factor app
7. TestContainers
8. Итоги

Задача

Задача

Нам прислали два приложения на тестирование, но в этот раз разработчики поленились сделать тестовый режим, и сказали, что первое приложение написано для платформы Node.js, а второе — на Java, но для работы требует СУБД MySQL.

Естественно, возникает вопрос, как нам это всё установить и использовать?

Зависимости

Понятно, что есть самый очевидный вариант решения этой задачи: ничего страшного, скачиваем дистрибутив Node.js, MySQL (если сможем найти нужную версию), разбираемся с установкой, устанавливаем настраиваем.

Вопрос к аудитории: какие минусы у этого варианта?

Зависимости

Понятное дело, что минусов достаточно много, например, для СУБД:

1. Вы должны разобраться с тем, как установить и настроить базу данных;
2. Вы должны проследить за тем, чтобы при следующем тестировании, база данных была в «чистом состоянии», т.е. не должно сказываться предыдущее тестирование;
3. Вы должны обеспечить должную изоляцию, если тестируете несколько проектов или несколько сервисов, требующих этой СУБД;
4. Кроме того, нужно следить, чтобы ваша система не превратилась в «мусорку» от количества устанавливаемых сервисов.

Зависимости

А для платформы:

1. Вы должны разобраться с тем, как установить и настроить Node.js;
2. Вы должны проследить за тем, чтобы установить дополнительные пакеты для работы приложения (очень редко можно встретить приложения, не использующие внешних зависимостей);
3. Вы должны обеспечить должную изоляцию, т.е. если вы установили зависимости для одного проекта, то они не должны мешать другому проекту (или другой версии этого же проекта);
4. Кроме того, нужно следить, чтобы ваша система не превратилась в «мусорку» от количества устанавливаемых платформ и доп.пакетов (и решать проблемы с версиями платформы и этих пакетов).

И всё это занимает достаточное количество времени.

Варианты

Вариантов достаточно много, но давайте ключевые:

- требование тестовой инфраструктуры (т.е. вы требуете, чтобы наняли отдельного системного администратора, который бы занимался установкой СУБД на выделенные сервера);
- системы виртуализации;
- системы контейнеризации.

Первый вариант, конечно же, идеальный, но мы рассмотрим два других сценария.

Виртуализация

Виртуализация

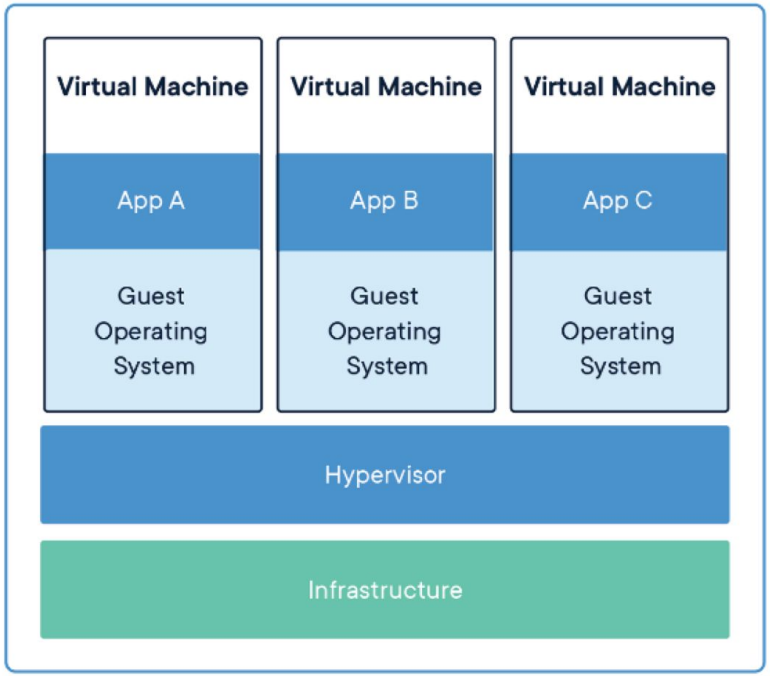
Системы виртуализации позволяют вам на одной физической машине эмулировать несколько виртуальных.

Работа виртуальной машины, в большинстве случаев, ничем не отличается от работы физической, у неё также есть:

- процессор;
- ОЗУ;
- жёсткий диск;
- сетевые устройства;
- и т.д.

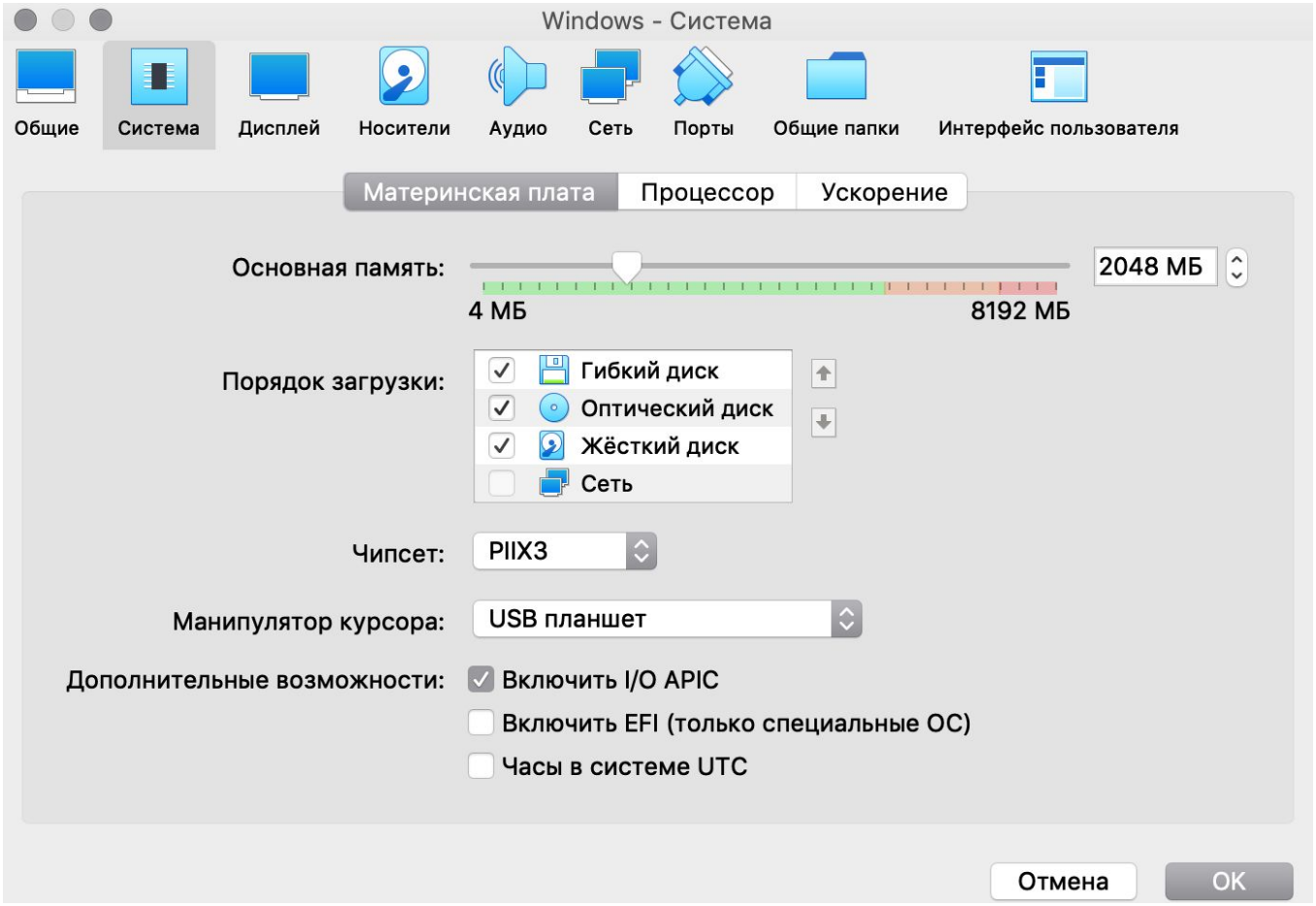
Ресурсы, естественно, берутся с физической (её ещё называю хостовой, а виртуальную машину — гостевой) машины.

Виртуализация



Изображение с сайта docker.com

Виртуализация



Виртуализация

Достаточно много как платных, так и бесплатных продуктов:

- VirtualBox;
- VMWare;
- Parallels;
- Hyper-V;
- и т.д.

Важно: большинство систем виртуализации требует x64 и поддержку виртуализации на уровне процессора (включается в BIOS), поэтому если у вас старый компьютер, то воспользоваться виртуализацией, скорее всего, не получится.

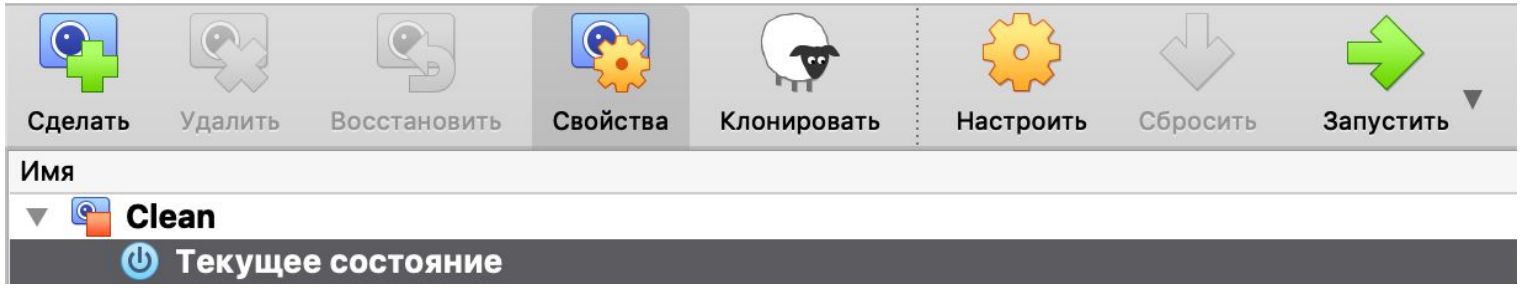
Виртуальные машины

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'а, 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.

Снапшоты


Сделать
Удалить
Восстановить
Свойства
Клонировать
Настроить
Сбросить
Запустить
Имя
Clean
Текущее состояние

Преимущества

Основные преимущества:

- экономия времени на настройку и установку;
- гибкость настройки;
- возможность отката изменений;
- виртуальную машину не жалко убить — вы в любой момент можете создать новую.

Недостатки

Раз есть преимущества, то должны быть и недостатки:

- всё равно в первый раз необходимо будет систему установить и настроить;
- тратятся ресурсы на работу гостевой ОС и системных приложений;
- виртуальные машины достаточно «много весят»: если вы создали диск на 80Гб, то образ виртуальной машины может хоть и не использовать весь диск, но будет весить несколько Гб (вплоть до 80).

VDS и другие

Виртуальные машины получили достаточно широкое распространение как для разработки и тестирования, так и для эксплуатации приложений: услуга VDS (Virtual Dedicated Server) является одной из самых популярных и предоставляется почти всеми провайдерами.

Вы покупаете виртуальную машину на мощностях провайдера и можете распоряжаться ею по собственному усмотрению.

Мы рекомендуем вам самостоятельно ознакомиться с продуктами, предоставляющими возможности виртуализации:

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

Контейнеризация

Контейнеризация

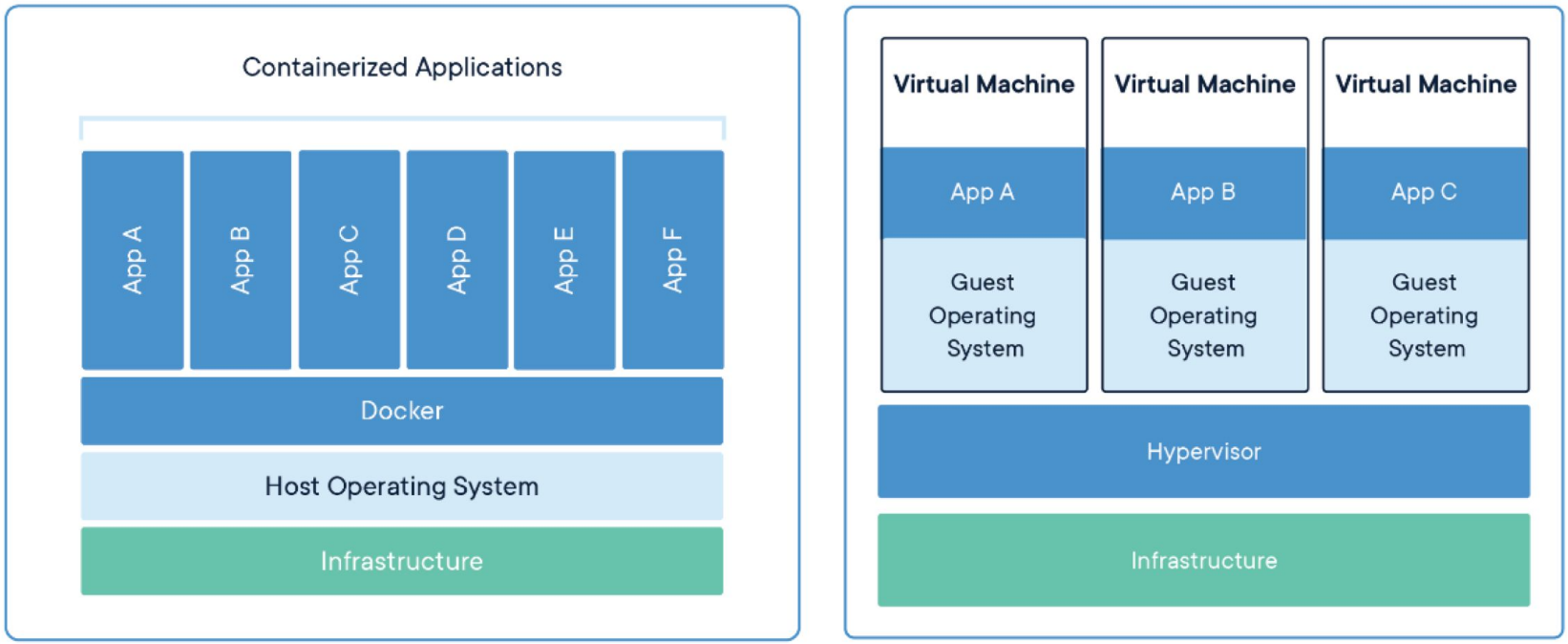
Несмотря на то, что виртуализация — это здорово, сообщество искало пути создания более лёгкого формата, обеспечивающего почти те же самые возможности.

Так появилась контейнеризация приложений — упаковка всех необходимых для работы приложения зависимостей* в единый образ, который можно запустить на определённой ОС.

Ключевое: отказались от возможности запуска другой ОС, за счёт этого приобрели выигрыш в производительности и «весе».

Примечание*: мы специально выбрали два примера — с СУБД и платформой. Чаще всего СУБД в мире контейнеризации не входит в зависимости приложения и функционирует как отдельный сервис.

Контейнеризация



Изображение с сайта docker.com

Контейнеризация

Контейнеры использую ядро хостовой ОС: приложение, написанное для ядра Windows, не запустится на Linux.

Но обёртки позволяют запускать Linux-контейнеры на Windows/Mac (создаётся легковесная виртуальная машина с Linux и уже в ней выполняются контейнеры).

Docker

Docker

Системы контейнеризации существуют достаточно давно, но наибольшее распространение в настоящий момент получил Docker.

В первую очередь, это связано с тем, что именно Docker предложил удобную систему поставки преднастроенных контейнеров и открытую инфраструктуру их распространения.

Docker

Представьте, что у вас есть Gradle вместе с репозиториями, но теперь там лежат не библиотеки, а готовые контейнеры с приложениями — базы данных, сервисы, платформы и т.д.

И вы можете буквально одной командой скачать их и запустить (как мы делали с Gradle — блок `dependencies` в `build.gradle`).

Кроме того, вы можете на базе уже существующих контейнеров создавать свои (как мы на базе библиотеки JUnit, Selenide, Akita и других — строили свои авто-тесты).

Docker

Docker достаточно требователен к машине, на которую вы собираетесь его устанавливать — **Windows 10 Pro (и выше), либо Linux/MacOS**.

Если же у вас Windows 7 или Windows 10 (но не Pro), то Docker вы установить не сможете, но сможете «попробовать» его с рядом ограничений, установив Docker Toolbox.

В приложениях к ДЗ вы найдёте подробные инструкции по установке.

Важно: Docker так же требует x64 и поддержку виртуализации на уровне процессора (включается в BIOS), поэтому если у вас старый компьютер, то воспользоваться Docker'ом, скорее всего, не получится.

Docker Playground

Даже если ваша система не удовлетворяет требованиям Docker, вы можете поэкспериментировать с помощью площадки Play with Docker.

К нему даже можно подключиться по SSH, если вы умеете им пользоваться*.

Если такая потребность возникнет — пишите в Slack.

Help

Приступим к работе: все команды будут выполняться в командной строке (в IDEA встроен уже терминал: Alt + F12).

У Docker прекрасная системой справки, которая вызывается по команде:

```
$ docker help

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
  --config string      Location of client config files (default "/root/.docker")
  ...

Management Commands:
  ...
  container   Manage containers
  image       Manage images
  ...

Run 'docker COMMAND --help' for more information on a command.
```

System

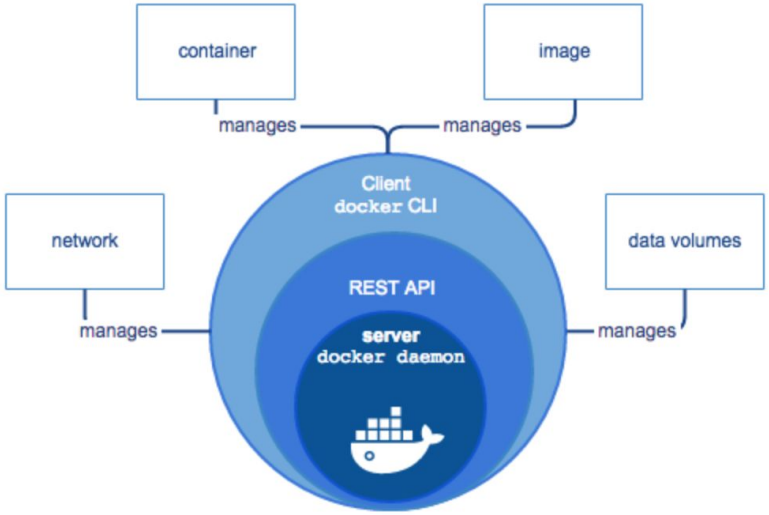
Смотрим версию:

```
$ docker system info

Client: Docker Engine - Community
  Version:           19.03.4
  API version:       1.40
  Built:             Thu Oct 17 23:44:48 2019
  OS/Arch:           darwin/amd64

Server: Docker Engine - Community
  Engine:
    Version:          19.03.4
    API version:      1.40 (minimum version 1.12)
    Built:            Thu Oct 17 23:50:38 2019
    OS/Arch:          linux/amd64
  containerd:
    Version:          v1.2.10
  runc:
    Version:          1.0.0-rc8+dev
  docker-init:
    Version:          0.18.0
```

Архитектура

Q: Что это всё значит? Client/Server и т.д., мы же ставили просто Docker.

A: Docker имеет клиент-серверную архитектуру (т.е. вы вполне можете подключаться к серверной части, установленной не на вашей машине). В целом, это выглядит вот так:



Изображение с сайта docker.com

Container, Image, Volumes, Network

Q: Хорошо, а что такое `Container`, `Image`, `Volume`, `Network`?

A: Хороший вопрос, давайте начнём по-порядку:

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

Если выстраивать ассоциации, то это как «класс» и «объект» в Java: из образа вы создаёте контейнер.

Image

Image состоит из двух частей:

- Снапшот файловой системы;
- Команда запуска.

Q: Только одна команда?

A: Чаще всего да, стараются делать контейнеры, ответственные за запуск только одной команды (например, `java -jar app.jar`).

Q: А почему снапшот? Разве мы не можем туда что-то писать?

A: Писать вы можете, но опять-таки, стараются делать контейнеры иммутабельными (не хранящими состояние), чтобы их проще было уничтожать и создавать заново.

Первый запуск

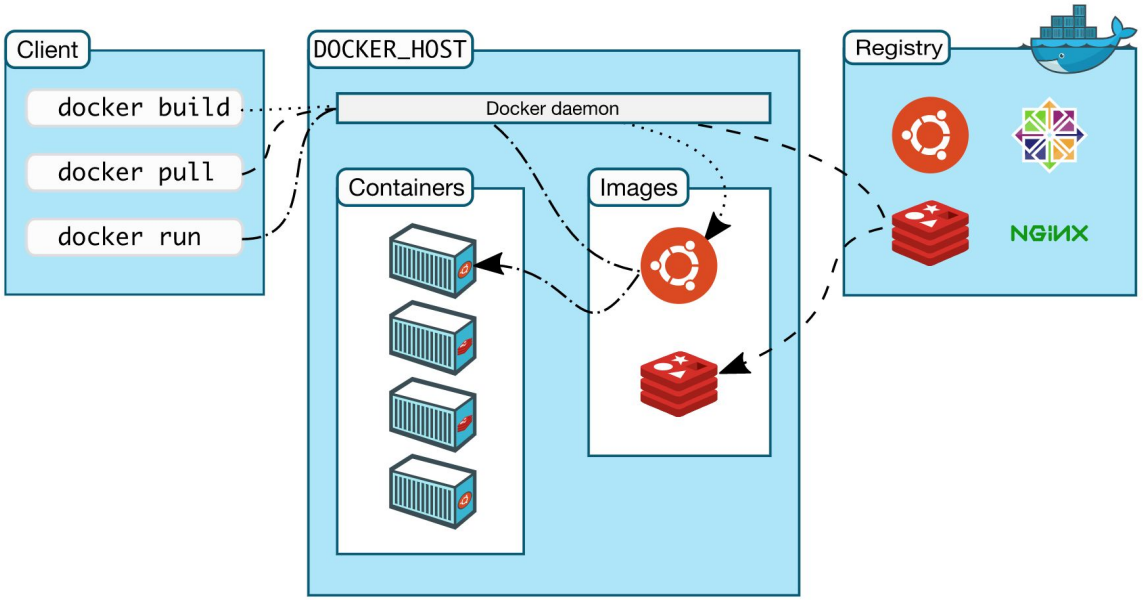
```
# скачиваем образ с Docker Hub
$ docker image pull hello-world

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
```

Важно: механика такая же, как в Git — вы можете использовать id (писать целиком не обязательно), либо давать логические имена (`--name`).

Инфраструктура

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.



run

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.

Управление

Для управления контейнерами и образами служат специальные команды:

```
# 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
```

Синтаксис команд

В старых руководствах другой синтаксис: `docker run` `docker ps`.

Docker развивался достаточно быстро и некоторые части (например, команды) проектировались стихийно.

Сейчас команда Docker работает над наведением порядка и старые команды хоть и работают, но в скором будущем могут быть отключены.

Документация

Большинство образов содержат документацию по запуску, например, для MySQL вы переходите на hub.docker.com и в поиске вбиваете MySQL:

Official Image

й учётной записи.

Старайтесь использовать именно их, т.к. бывают ещё не официальные и не факт, что их содержимому можно доверять.

Документация

На странице образа будет представлена документация по запуску и доступным опциям (если авторы образа об этом позаботились):

О СУБД

СУБД (Системы Управления Базами Данных) появились достаточно давно и изначально были рассчитаны на большое количество пользователей и разграничение прав доступа.

Под пользователями в данном случае понимаются субъекты, манипулирующие объектами, хранящимися в БД.

Например, сервис, планирующий использовать БД в качества хранилища данных подключается от имени определённого пользователя к определённой БД.

При этом и пользователь, и БД должны быть заранее созданы и у пользователя должны быть права на чтение/запись/модификацию БД.

Кроме того, если это сетевая БД, то нужно знать к какому порту и хосту подключаться.

О СУБД

Возникает вопрос, а как же сконфигурировать СУБД, если мы используем иммутабельные образа?

Конфигурация

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'а — 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).

Примечание:* это простейший случай.

Переменные окружения

Вы уже должны знать про переменные окружения в ОС.

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;
- и другие.

Перечень допустимых переменных окружения их предназначения определяется автором образа.

Переменные окружения

```
$ 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>`

Binding портов

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.

Binding портов

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
```

Подключение к контейнеру

Q: А что если я хочу подключиться к контейнеру и выполнить там какую-то команду? 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`.

Volumes

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

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

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

Volumes

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

Docker Compose

Каждый раз заполнять все параметры командной строки очень неудобно.

Можно, конечно, сохранить их в файлик и даже написать скрипт.

А если нужно запустить одновременно несколько контейнеров? Тоже скрипт?

Об этой проблеме уже позаботились и предоставили нам инструмент Docker Compose.

Docker Compose

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
```

Docker Compose

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.

Docker Compose

Q: Но почему синтаксис так отличается от командной строки Docker?

A: Изначально инструмент Docker Compose разрабатывался другой организацией и только впоследствии был выкуплен (вместе с организацией).

Tags

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`.

Возвращаемся к задаче

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.

Возвращаемся к задаче

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.

12 factor app

Тестирование и CI/CD

А теперь давайте подумаем: есть практика, когда весь сервис, его зависимости и большая часть контролируемых внешних сервисов, представляют из себя Docker контейнеры.

Насколько удобным это делает тестирование?

Тестирование и CI/CD

Тестирование становится максимально удобным: вам нужно уметь работать с Docker'ом, а вся настройка будет описана с помощью Docker Compose.

Вам лишь нужно следить за тем, чтобы на вашей хостовой машине нужные порты были свободны (либо заменить их) и существовали нужные каталоги.

12 factor app

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/.

TestContainers

TestContainers

Поскольку контейнеры — достаточно легковесная и удобная штука, логично предположить, что уже кто-то додумался их «вкрутить прямо в тесты».

Библиотека TestContainers позволяет вам прямо из тестов создавать контейнеры и управлять ими.

К сожалению, поддержка на системах, отличных от Linux, не даёт нам включить рассмотрение этой библиотеки в наш курс.

Итоги

Итоги

Сегодня мы рассмотрели Docker и Docker Compose.

Эти инструменты очень часто используются в разработке, тестировании и продакшене.

Советуем вам обратить особое внимание на навыки работы с ними.

Мы рассмотрели только малую часть возможностей, поэтому рекомендуем вам продолжить самостоятельно ознакамливаться с ними.

Домашнее задание

Давайте посмотрим ваше домашнее задание.

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

нетология

Задавайте вопросы и напишите отзыв о лекции!

ОКСАНА МЕЛЬНИКОВА

Оксана Мельникова