Docker Compose

Введение

> Прежде чем приступить к работе с Docker Composer необходимо ознакомиться с Docker и понятием контейнерной виртуализации.

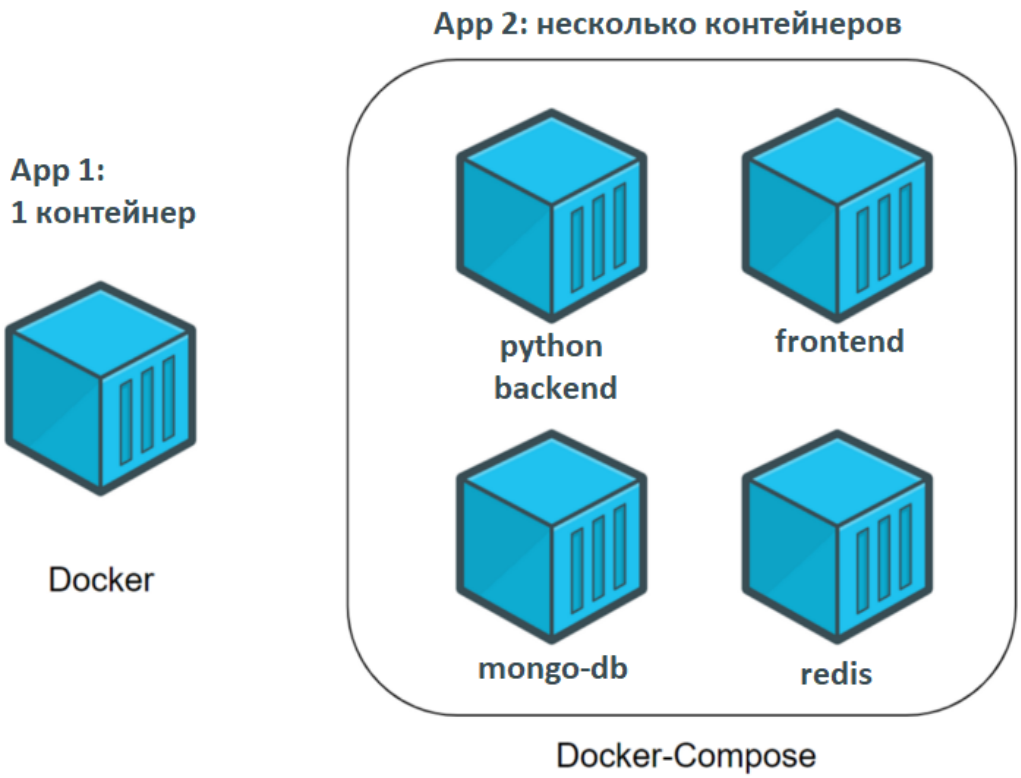
Docker - это то, что он позволяет избавиться от необходимости установки на компьютер различных сервисов. К их числу можно отнести и веб-сервер Apache или Nginx, базы данных и прочие компоненты инфраструктуры приложения. Вся инфраструктура прописана в конфигурационном файле docker-compose.yml и запускается одной командой вместе с вашим приложением. Все что нужно разработчику работающему с докером, это по сути сам докер и любимая среда разработки и ВСЕ!

В определённый момент (когда образов и контейнеров становится много) и возможностей одного Docker не хватает, на помощь ему приходят дополнительные инструменты в лице ***Docker Compose*** и ***Docker Swarm***/***Kubernetes*****.**

В этом руководстве размерёмся с тем, что такое Docker Compose, как с ним работать, а также рассмотрим пример развертки локального окружения состоящего из связки LEMP: ***Linux + Nginx + PHP + MariaDB(MySql) + phpMyAdmin*** с помощью Docker Compose. Данная связка очень популярна и может удовлетворить ряд стандартных потребностей рядового разработчика.

Что такое Docker Composer и зачем он нужен

Docker-compose - инструмент, который позволяет разворачивать и настраивать несколько контейнеров одновременно. Например, для веб-приложения нужно развернуть стек ***LAMP***: ***Linux, Apache, MySQL, PHP***. Каждое из приложений — это отдельный контейнер. Но в этой ситуации нам нужны именно все контейнеры вместе, а не отдельно взятое приложение. *Docker-compose* позволяет развернуть и настроить все приложения одной командой, а без него пришлось разворачивать и настраивать каждый контейнер отдельно.



Docker-Compose запускает несколько контейнеров на одном компьютере/виртуальной машине. Источник: https://ivan-shamaev.ru/docker-compose-tutorial-container-image-install/

Важно понимать, что ***Docker-compose*** - нужен для запуска *нескольких (разных) контейнеров вместе на одном компьютере* .

Также как Docker для создания контейнеров использует специальный файл - `Dockerfile`, Docker Compose для своей работы использует специальный файл конфигурации `docker-compose.yml`. Файл `docker-compose.yml` - этой файл который будет содержать инструкции, необходимые для запуска и настройки контейнеров, их взаимодействия и координации. Тоесть в `.yml` вы описываете какие у вас будут запускаться контейнеры, их настройки, то как они будут между собой взаимодействовать и так далее.

YML Файлы

YAML — это язык для хранения информации в формате понятном человеку. Его название расшифровывается как, «Ещё один язык разметки».

AML обычно применяют для создания конфигурационных файлов в программах типа Инфраструктура как код (Iac), или для управления контейнерами в работе DevOps.

Чаще всего с помощью YAML создают протоколы автоматизации, которые могут выполнять последовательности команд записанные в YAML-файле. Это позволяет вашей системе быть более независимой и отзывчивой без дополнительного внимания разработчика.

> Кроме того, YAML легко интегрировать, благодаря поддержке Python (используя PyYAML библиотеку, Docker или Ansible) и других популярных технологий.

Синтаксис YAML

В языке есть несколько базовых концепций, которые позволяют обрабатывать большинство данных.

Пары ключ-значение

Большинство данных в YAML-файле хранятся в виде пары ключ-значение, где ключ — это имя пары, а значение — связанные данные.

Скаляры и маппинг

Скаляр представляет собой одно значение, которому соответствует имя.

YAML поддерживает стандартные типы: int и float, boolean, string и null.

Они могут быть представлены в разных видах: шестнадцатеричном, восьмеричном или экспоненциальном. Также существуют специальные типы для математических сущностей, такие как: бесконечность, -бесконечность и NAN.

```
integer: 25
hex: 0x12d4 #равно 4820
octal: 023332 #равно 9946
float: 25.0
exponent: 12.3015e+05 #равно 1230150.0
boolean: Yes
string: "25"
infinity: .inf # преобразуется в бесконечность
neginf: -.Inf #преобразуется в минус бесконечность
not: .NAN #Not a Number
null: ~
```

Строки

Строка — это коллекция символов, которая может содержать слово или предложение. Можно использовать либо |, для отдельных строк, либо >, для параграфов.

Кавычки в YAML не нужны.

```
str: Hello World
data: |
   Это
   Отдельные
   Строки
data: >
   Это
   один параграф
   текста
```

Последовательности

Последовательности — это структуры данных похожие на списки или массивы, которые хранят несколько значений под одним ключом. Они определяются с помощью отступов или [].

```
shopping:
- milk
- eggs
- juice
```

Однострочные последовательности выглядят лаконичнее, но хуже читаются.

```
shopping: [milk, eggs, juice]
```

Словари

Словари — это коллекции пар ключ-значение, которые хранятся под одним ключом. Они позволяют разделить данные на логические категории.

```
Employees:
- dan:
    name: Dan D. Veloper
    job: Developer
    team: DevOps
- dora:
   name: Dora D. Veloper
   job: Project Manager
   team: Web Subscriptions
```

Словари могут содержать более сложные структуры, что позволяет хранить сложные реляционные данные.

Что ещё может YAML?

- Anchors (якоря)
- Templates (шаблоны)
- Взаимодействие с Docker, Ansible и т. д.
- Расширенные последовательности и маппинг.
- Расширенные типы данных (timestamp, null и т. д.)

```
# Файл docker-compose должен начинаться с тега версии.
# Мы используем "3" так как это - самая свежая версия на момент написания этого кода.

version: "3"

# Следует учитывать, что docker-composes работает с сервисами. 1 сервис = 1 контейнер.
# Сервисом может быть клиент, сервер, сервер баз данных...
# Раздел, в котором будут описаны сервисы, начинается с 'services'.

services:

  # Будут созданы клиентское и серверное приложения.
  # Это означает, что нам нужно два сервиса.
  # Первый сервис (контейнер): сервер. Назвать его можно так, как нужно разработчику.
  # Понятное название сервиса помогает определить его роль.
  # Здесь мы, для именования соответствующего сервиса, используем ключевое слово 'server'.

  server:

    # Ключевое слово "build" позволяет задать
    # путь к файлу Dockerfile, который нужно использовать для создания образа,
    # который позволит запустить сервис.
    # Здесь 'server/' соответствует пути к папке сервера,
    # которая содержит соответствующий Dockerfile.

    build: server/

    # Команда, которую нужно запустить после создания образа.
    # Следующая команда означает запуск "python ./server.py".

    command: python ./server.py

    # Вспомните о том, что в качестве порта в 'server/server.py' указан порт 1234.
    # Если нужно обратиться к серверу со своего компьютера (находясь за пределами контейнера),
    # следует организовать перенаправление этого порта на порт компьютера.
    # Сделать это поможет ключевое слово 'ports'.
    # При его использовании применяется следующая конструкция: [порт компьютера]:[порт контейнера]
    # В данном случае нужно использовать порт компьютера 1234 и организовать его связь с портом
    # 1234 контейнера (так как именно на этот порт сервер ожидает поступления запросов).

    ports:
      - 1234:1234

  # Второй сервис (контейнер): клиент.
  # Этот сервис назван 'client'.

  client:
    # Здесь 'client/ соответствует пути к папке, которая содержит
```

```
    # файл Dockerfile для клиентской части системы.

    build: client/

    # Команда, которую нужно запустить после создания образа.
    # Следующая команда означает запуск "python ./client.py".

    command: python ./client.py

    # Ключевое слово 'network_mode' используется для описания типа сети.
    # Тут указывается, что контейнер может обращаться к 'localhost' компьютера.

    network_mode: host

    # Ключевое слово 'depends_on' позволяет указывать, должен ли сервис,
    # прежде чем запуститься, ждать, когда будут готовы к работе другие сервисы.
    # Нужно, чтобы сервис 'client' дождался бы готовности к работе сервиса 'server'.

    depends_on:
      - server
```

> *При добавлении информации в файл docker-compose.yml нужно **обязательно сохранять форматирование**, данное в примере. Это прежде всего касается отступов перед блоками. Если это расстояние нарушить, будет выведена ошибка.*

Установка Docker Compose

Чтобы получить самую последнюю стабильную версию Docker Compose, загружать его необходимо из официального репозитория Github.

Для начала проверьте, какая последняя версия доступна на странице релизов. На момент написания настоящего документа наиболее актуальной стабильной версией является версия `2.2.3`.

Следующая команда загружает версию `2.2.3` и сохраняет исполняемый файл в каталоге `/usr/local/bin/docker-compose`, в результате чего данное программное обеспечение будет глобально доступно под именем `docker-compose`:

```
sudo curl -L "https://github.com/docker/compose/releases/download/v2.2.3/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-c
```

Затем необходимо задать правильные разрешения, чтобы сделать команду `docker-compose` исполняемой:

```
sudo chmod +x /usr/local/bin/docker-compose
# либо
sudo chmod 777 /usr/local/bin/docker-compose
```

Чтобы проверить успешность установки, выполним следующую команду:

```
docker-compose --version
```

Установка Docker Compose успешно выполнена. Далее поговорим про файл `docker-compose.yml` и запустить контейнерную среду с помощью этого инструмента.

Практика 1: Установка LEMP с помощью Docker Compose

Наша сборка будет включать классический LEMP стек:

- PHP;
- Composer;
- MariaDB (форк MySQL);

- Nginx;
- phpMyAdmin.

Использовать будем следующие docker образы с Docker Hub:

- PHP;
- Nginx;
- MariaDB;
- PhpMyAdmin.

Структура проекта

Создадим папку `project`, где будет лежать наш проект. В ней нужно будет создать следующий набор папок и файлов:

- **www** — в этой папке будут лежать файлы наших проектов, по директории на каждый проект (сайт)
- **mysql** — в этой папке будут храниться файлы наших баз данных
- **logs** — здесь будет собирать логи из разных образов
- **hosts** — здесь будут храниться файлы конфигурации nginx для наших проектов
- **images** — папка с нашими образами

```
mkdir project
cd project
mkdir www mysql logs hosts images
```

Теперь необходимо создать папку с нашим сайтом для проверки сборки. В папке `www` создадим папку с нашим проектом (первым сайтом) , например `hello`.

```
mkdir www/hello.test
```

В ней создадим файл `index.php` со следующим содержанием:

```
touch www/hello.test/index.php
sudo nano www/hello.test/index.php
```

```
<h1>Hello, mate!</h1>
<h4>Attempting MySQL connection from php</h4>
<?php
$host = 'mysql';
$user = 'root';
$pass = 'secret';
$conn = new mysqli($host, $user, $pass);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected to MySQL successfully!";
?>
```

Далее создадим в папке с нашими образами папку `php`, а в ней файлы `php.ini` (можем в нем определять свои значения конфига) и `Dockerfile` (тут будут настройки нашего образа):

```
sudo mkdir images/php
sudo touch images/php/php.ini
sudo touch images/php/Dockerfile
sudo touch docker-compose.yml
```

Таки образом у на получается следующая структура:

```
tree ~/project

/home/pylounge/project
├── docker-compose.yml
├── hosts
├── images
│   └── php
│       ├── Dockerfile
│       └── php.ini
├── logs
├── mysql
└── www
    └── hello.test
        └── index.php
```

Собираем наш PHP образ (Dockerfile)

Официальный образ PHP с Docker Hub не включает в себя никаких дополнительных модулей, для того чтобы их включить, мы соберем собственный образ на основе официального с помощью Dockerfile:

```
sudo nano images/php/Dockerfile
```

В файл `images/php/Dockerfile` пропишем следующее:

```
# Для начала указываем исходный образ, он будет использован как основа
FROM php:7.4-fpm

# RUN выполняет идущую за ней команду в контексте нашего образа.
# В данном случае мы установим некоторые зависимости и модули PHP.
# Для установки модулей используем команду docker-php-ext-install.
# На каждый RUN создается новый слой в образе, поэтому рекомендуется объединять команды.
RUN apt-get update && apt-get install -y \
        curl \
        wget \
        git \
        libfreetype6-dev \
        libonig-dev \
        libpq-dev \
        libjpeg62-turbo-dev \
        libmcrypt-dev \
        libpng-dev \
        libzip-dev \
    && pecl install mcrypt-1.0.3 \
    && docker-php-ext-install -j$(nproc) iconv mbstring mysqli pdo_mysql zip \
    && docker-php-ext-configure gd --with-freetype --with-jpeg \
    && docker-php-ext-install -j$(nproc) gd \
    && docker-php-ext-enable mcrypt

# Куда же без composer'a.
RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer

# Добавим свой php.ini, можем в нем определять свои значения конфига
ADD php.ini /usr/local/etc/php/conf.d/40-custom.ini

# Указываем рабочую директорию для PHP
WORKDIR /var/www

# Запускаем контейнер
# Из документации: The main purpose of a CMD is to provide defaults for an executing container. These defaults can include an executable,
# or they can omit the executable, in which case you must specify an ENTRYPOINT instruction as well.
CMD ["php-fpm"]
```

Работа с файлом docker-compose.yml

Docker Compose упрощает жизнь если у вас больше одного контейнера. С помощью одного, а иногда нескольких файлов, мы описываем какие контейнеры запускать, их настройки и связи между контейнерами. Начиная

со второй версии docker compose поддерживает наследование и можно с его помощью описывать разные конфигурации для разных окружений. В файл `docker-compose.yml` пропишем следующее:

```
sudo nano docker-compose.yml
```

```
version: '2'
services:
    nginx:
      # используем последний стабильный образ nginx
        image: nginx:latest
        # маршрутизируем порты
        ports:
            - "8000:80"
        # монтируем директории, слева директории на основной машине, справа - куда они монтируются в контейнере
        volumes:
            - ./hosts:/etc/nginx/conf.d
            - ./www:/var/www
            - ./logs:/var/log/nginx
        # nginx должен общаться с php контейнером
        links:
            - php
    php:
        # у нас свой образ для PHP, указываем путь к нему и говорим что его надо собрать
        build: ./images/php
        # этот образ будет общаться с mysql
        links:
            - mysql
        # монтируем директорию с проектами
        volumes:
            - ./www:/var/www
    mysql:
        image: mariadb
        ports:
            - "3306:3306"
        volumes:
            - ./mysql:/var/lib/mysql
        # задаем пароль для root пользователя
        environment:
            MYSQL_ROOT_PASSWORD: secret
    pma:
      # используем последний стабильный образ phpmyadmin
        image: phpmyadmin/phpmyadmin
        restart: always
        links:
            - mysql:mysql
        ports:
            - 8183:80
        environment:
            # прописываем название нашего MySQL хоста
            PMA_HOST: mysql
            MYSQL_USERNAME: root
            MYSQL_ROOT_PASSWORD: secret
```

Конфигурация сервера nginx для проектов

Мы уже создали тестовый проект(сайт) `hello`, создадить для этого проекта nginx конфиг. В папке `hosts` создадим файл `hello.conf`:

```
sudo nano hosts/hello.conf
```

```
server {
    index index.php;
    server_name hello.test;
    error_log  /var/log/nginx/error.log;
    access_log /var/log/nginx/access.log;
    root /var/www/hello.test;

    location ~ \.php$ {
```

```
        try_files $uri =404;
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        fastcgi_pass php:9000;
        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }
}
```

Конфиг nginx для docker контейнеров ничем не отличается от обычного конфига для сайта. Стоит лишь обратить внимание на директиву `fastcgi_pass`, где мы используем не путь к unix-сокету, а адрес `php:9000`. Здесь присутствует немного магии docker'а: php - это хост по которому доступен наш php контейнер внутри контейнера `nginx`, а 9000 - порт, по которому можно достучаться до fpm-сокета. Тут просто стоит запомнить.

Запуск Docker Compose

> Внимание! Хост для доступа к MySQL — `mysql`. Не `localhost`, а именно `mysql`. Пароль к базе мы указали в файле `docker-compose.yml` - *MYSQL_ROOT_PASSWORD: secret.* Таким образом, хость - `mysql`, пароль - `secret`.

Минимальная конфигурация для нашей локальной разработки готовы. Осталось только запустить ее и проверить, работает ли оно. Переходим в корень нашего проекта, где лежит файл `docker-compose.yml`. Выполняем команду:

```
cd ~/project
docker-compose up -d
```

`docker-compose up -d` загрузит необходимые образы Docker, создаст контейнер для службы и запустит контейнерную среду в фоновом режиме (флаг `-d`). Docker Compose будет вначале искать заданный образ в локальной системе, и если не найдет его, загрузит его из Docker Hub.

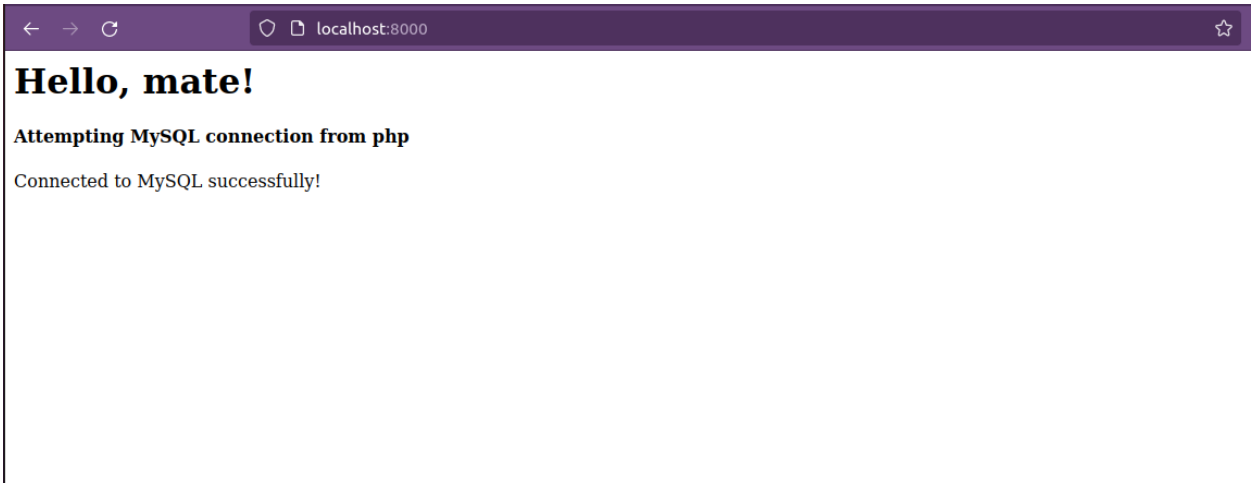
Ждем. Первый запуск он самый долгий, потому что докеру нужно скачать образы и собрать наш образ php.

В самом конце мы увидим:

```
[+] Running 5/5
 ⠿ Network project_default    Created                                                    0.2s
 ⠿ Container project-mysql-1  Started                                                    1.4s
 ⠿ Container project-php-1    Started                                                    2.9s
 ⠿ Container project-pma-1    Started                                                    2.9s
 ⠿ Container project-nginx-1  Started
```

Они сообщают нам, что все 5 контейнеров поднялись и работают.

Проверим. Открываем браузер и переходим по адресу `http://localhost:8000/`



Результат работы

Теперь среда запущена в фоновом режиме. Для проверки активности контейнера используйте следующую команду:

```
sudo docker ps

CONTAINER ID   IMAGE                  COMMAND                  CREATED              STATUS              PORTS
60d6a7721cb0   nginx:latest           "/docker-entrypoint.…"   About a minute ago   Up About a minute   0.0.0.0:8000->80/tcp, :::8000->80/
4b6bdd18a14e   phpmyadmin/phpmyadmin  "/docker-entrypoint.…"   About a minute ago   Up About a minute   0.0.0.0:8183->80/tcp, :::8183->80/
4acc4e0e07fc   project_php            "docker-php-entrypoi…"   About a minute ago   Up About a minute   9000/tcp
83e5cf2b4e74   mariadb                "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0.0:3306->3306/tcp, :::3306->3
```

Эта команда покажет информацию о работающих контейнерах и их состоянии, а также о действующей переадресации портов.

Заданный в файле `docker-compose.yml` общий том синхронизирует файлы в папке `www/hello` с корневым каталогом документов контейнера. Если вы внесете любые изменения в файл `index.php`, они будут автоматически отражены в контейнере и появятся в браузере после перезагрузки страницы.

Итоговая структура проекта:

```
pylounge@ubuntu:~/project$ tree ~/project
/home/pylounge/project
├── docker-compose.yml
├── hosts
│   └── hello.conf
├── images
│   └── php
│       ├── Dockerfile
│       └── php.ini
├── logs
│   ├── access.log
│   └── error.log
├── mysql [error opening dir]
└── www
    └── hello.test
        └── index.php
```

Основные команды Docker Compose

Мы рассмотрели процедуру настройки файла `docker-compose.yml` и запуск среды с помощью команды `docker-compose up`.

Посмотрим, как использовать команды Docker Compose для управления контейнерной средой и взаимодействия с ней.

Чтобы посмотреть журналы контейнера Nginx, используйте команду `logs`:

```
docker-compose logs
```

▼ Лог файл Nginx (нажми, чтобы посмотреть)

project-mysql-1 | 2022-02-06 05:01:49+00:00 [Note] [Entrypoint]: Entrypoint script for MariaDB Server 1:10.6.5+maria~focal started.
project-mysql-1 | 2022-02-06 05:01:50+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
project-mysql-1 | 2022-02-06 05:01:50+00:00 [Note] [Entrypoint]: Entrypoint script for MariaDB Server 1:10.6.5+maria~focal started.
project-mysql-1 | 2022-02-06 5:01:50 0 [Note] mariadbd (server 10.6.5-MariaDB-1:10.6.5+maria~focal) starting as process 1 ...
project-mysql-1 | 2022-02-06 5:01:50 0 [Note] InnoDB: Compressed tables use zlib 1.2.11
project-mysql-1 | 2022-02-06 5:01:50 0 [Note] InnoDB: Number of pools: 1
project-mysql-1 | 2022-02-06 5:01:50 0 [Note] InnoDB: Using crc32 + pclmulqdq instructions
project-mysql-1 | 2022-02-06 5:01:50 0 [Note] mariadbd: O_TMPFILE is not supported on /tmp (disabling future attempts)
project-mysql-1 | 2022-02-06 5:01:50 0 [Note] InnoDB: Using Linux native AIO
project-mysql-1 | 2022-02-06 5:01:50 0 [Note] InnoDB: Initializing buffer pool, total size = 134217728, chunk size = 134217728
project-mysql-1 | 2022-02-06 5:01:50 0 [Note] InnoDB: Completed initialization of buffer pool
project-mysql-1 | 2022-02-06 5:01:51 0 [Note] InnoDB: 128 rollback segments are active.
project-mysql-1 | 2022-02-06 5:01:51 0 [Note] InnoDB: Creating shared tablespace for temporary tables
project-mysql-1 | 2022-02-06 5:01:51 0 [Note] InnoDB: Setting file './ibtmp1' size to 12 MB. Physically writing the file full; Please wait ...
project-mysql-1 | 2022-02-06 5:01:51 0 [Note] InnoDB: File './ibtmp1' size is now 12 MB.
project-mysql-1 | 2022-02-06 5:01:51 0 [Note] InnoDB: 10.6.5 started; log sequence number 42365; transaction id 14
project-mysql-1 | 2022-02-06 5:01:51 0 [Note] InnoDB: Loading buffer pool(s) from /var/lib/mysql/ib_buffer_pool
project-mysql-1 | 2022-02-06 5:01:51 0 [Note] Plugin 'FEEDBACK' is disabled.
project-mysql-1 | 2022-02-06 5:01:51 0 [Warning] You need to use --log-bin to make --expire-logs-days or --binlog-expire-logs-seconds work.
project-mysql-1 | 2022-02-06 5:01:51 0 [Note] Server socket created on IP: '0.0.0.0'.
project-mysql-1 | 2022-02-06 5:01:51 0 [Note] Server socket created on IP: '::'.
project-mysql-1 | 2022-02-06 5:01:51 0 [Warning] 'proxies_priv' entry '@% root@83e5cf2b4e74' ignored in --skip-name-resolve mode.
project-mysql-1 | 2022-02-06 5:01:51 0 [Note] InnoDB: Buffer pool(s) load completed at 220206 5:01:51
project-mysql-1 | 2022-02-06 5:01:51 0 [Note] mariadbd: ready for connections.
project-mysql-1 | Version: '10.6.5-MariaDB-1:10.6.5+maria~focal' socket: '/run/mysqld/mysqld.sock' port: 3306 mariadb.org binary distribution
project-mysql-1 | 2022-02-06 5:02:40 0 [Note] mariadbd (initiated by: unknown): Normal shutdown
project-mysql-1 | 2022-02-06 5:02:40 0 [Note] InnoDB: FTS optimize thread exiting.
project-mysql-1 | 2022-02-06 5:02:40 0 [Note] InnoDB: Starting shutdown...
project-mysql-1 | 2022-02-06 5:02:40 0 [Note] InnoDB: Dumping buffer pool(s) to /var/lib/mysql/ib_buffer_pool
project-mysql-1 | 2022-02-06 5:02:40 0 [Note] InnoDB: Buffer pool(s) dump completed at 220206 5:02:40
project-mysql-1 | 2022-02-06 5:02:40 0 [Note] InnoDB: Removed temporary tablespace data file: "./ibtmp1"
project-mysql-1 | 2022-02-06 5:02:40 0 [Note] InnoDB: Shutdown completed; log sequence number 42377; transaction id 15
project-mysql-1 | 2022-02-06 5:02:40 0 [Note] mariadbd: Shutdown complete
project-mysql-1 |
project-mysql-1 | 2022-02-06 05:03:02+00:00 [Note] [Entrypoint]: Entrypoint script for MariaDB Server 1:10.6.5+maria~focal started.
project-mysql-1 | 2022-02-06 05:03:02+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
project-mysql-1 | 2022-02-06 05:03:02+00:00 [Note] [Entrypoint]: Entrypoint script for MariaDB Server 1:10.6.5+maria~focal started.
project-mysql-1 | 2022-02-06 5:03:02 0 [Note] mariadbd (server 10.6.5-MariaDB-1:10.6.5+maria~focal) starting as process 1 ...
project-mysql-1 | 2022-02-06 5:03:03 0 [Note] InnoDB: Compressed tables use zlib 1.2.11
project-mysql-1 | 2022-02-06 5:03:03 0 [Note] InnoDB: Number of pools: 1
project-mysql-1 | 2022-02-06 5:03:03 0 [Note] InnoDB: Using crc32 + pclmulqdq instructions
project-mysql-1 | 2022-02-06 5:03:03 0 [Note] mariadbd: O_TMPFILE is not supported on /tmp (disabling future attempts)
project-mysql-1 | 2022-02-06 5:03:03 0 [Note] InnoDB: Using Linux native AIO

project-mysql-1 | 2022-02-06 5:03:03 0 [Note] InnoDB: Initializing buffer pool, total size = 134217728, chunk size = 134217728
project-mysql-1 | 2022-02-06 5:03:03 0 [Note] InnoDB: Completed initialization of buffer pool
project-mysql-1 | 2022-02-06 5:03:03 0 [Note] InnoDB: 128 rollback segments are active.
project-mysql-1 | 2022-02-06 5:03:03 0 [Note] InnoDB: Creating shared tablespace for temporary tables
project-mysql-1 | 2022-02-06 5:03:03 0 [Note] InnoDB: Setting file './ibtmp1' size to 12 MB. Physically writing the file full; Please wait ...
project-mysql-1 | 2022-02-06 5:03:03 0 [Note] InnoDB: File './ibtmp1' size is now 12 MB.
project-mysql-1 | 2022-02-06 5:03:03 0 [Note] InnoDB: 10.6.5 started; log sequence number 42377; transaction id 14
project-mysql-1 | 2022-02-06 5:03:03 0 [Note] InnoDB: Loading buffer pool(s) from /var/lib/mysql/ib_buffer_pool
project-mysql-1 | 2022-02-06 5:03:03 0 [Note] Plugin 'FEEDBACK' is disabled.
project-mysql-1 | 2022-02-06 5:03:03 0 [Warning] You need to use --log-bin to make --expire-logs-days or --binlog-expire-logs-seconds work.
project-mysql-1 | 2022-02-06 5:03:03 0 [Note] InnoDB: Buffer pool(s) load completed at 220206 5:03:03
project-mysql-1 | 2022-02-06 5:03:03 0 [Note] Server socket created on IP: '0.0.0.0'.
project-mysql-1 | 2022-02-06 5:03:03 0 [Note] Server socket created on IP: '::'.
project-mysql-1 | 2022-02-06 5:03:03 0 [Warning] 'proxies_priv' entry '@% root@83e5cf2b4e74' ignored in --skip-name-resolve mode.
project-mysql-1 | 2022-02-06 5:03:03 0 [Note] mariadbd: ready for connections.
project-mysql-1 | Version: '10.6.5-MariaDB-1:10.6.5+maria~focal' socket: '/run/mysqld/mysqld.sock' port: 3306 mariadb.org binary distribution
project-nginx-1 | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
project-nginx-1 | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
project-nginx-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
project-nginx-1 | 10-listen-on-ipv6-by-default.sh: info: /etc/nginx/conf.d/default.conf is not a file or does not exist
project-nginx-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
project-nginx-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
project-nginx-1 | /docker-entrypoint.sh: Configuration complete; ready for start up
project-nginx-1 | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
project-nginx-1 | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
project-nginx-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
project-nginx-1 | 10-listen-on-ipv6-by-default.sh: info: /etc/nginx/conf.d/default.conf is not a file or does not exist
project-nginx-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
project-nginx-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
project-nginx-1 | /docker-entrypoint.sh: Configuration complete; ready for start up
project-pma-1 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.18.0.4. Set the 'ServerName' directive globally to suppress this message
project-pma-1 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.18.0.4. Set the 'ServerName' directive globally to suppress this message
project-php-1 | [06-Feb-2022 05:01:51] NOTICE: fpm is running, pid 1
project-php-1 | [06-Feb-2022 05:01:51] NOTICE: ready to handle connections
project-php-1 | 172.18.0.5 - 06/Feb/2022:05:01:54 +0000 "GET /index.php" 200
project-php-1 | 172.18.0.5 - 06/Feb/2022:05:02:05 +0000 "GET /index.php" 200
project-php-1 | 172.18.0.5 - 06/Feb/2022:05:02:06 +0000 "GET /index.php" 200
project-php-1 | 172.18.0.5 - 06/Feb/2022:05:02:06 +0000 "GET /index.php" 200
project-php-1 | [06-Feb-2022 05:02:39] NOTICE: Finishing ...
project-php-1 | [06-Feb-2022 05:02:39] NOTICE: exiting, bye-bye!
project-php-1 | [06-Feb-2022 05:03:03] NOTICE: fpm is running, pid 1
project-php-1 | [06-Feb-2022 05:03:03] NOTICE: ready to handle connections
project-php-1 | 172.18.0.5 - 06/Feb/2022:05:03:06 +0000 "GET /index.php" 200
project-pma-1 | [Sun Feb 06 05:01:51.878868 2022] [mpm_prefork:notice] [pid 1] AH00163: Apache/2.4.52 (Debian) PHP/8.0.15 configured -- resuming normal operations
project-pma-1 | [Sun Feb 06 05:01:51.878935 2022] [core:notice] [pid 1] AH00094: Command line: 'apache2 -D FOREGROUND'
project-pma-1 | [Sun Feb 06 05:02:38.804986 2022] [mpm_prefork:notice] [pid 1] AH00170: caught SIGWINCH, shutting down

gracefully
project-pma-1 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.18.0.3. Set the 'ServerName' directive globally to suppress this message
project-pma-1 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.18.0.3. Set the 'ServerName' directive globally to suppress this message
project-pma-1 | [Sun Feb 06 05:03:03.718789 2022] [mpm_prefork:notice] [pid 1] AH00163: Apache/2.4.52 (Debian) PHP/8.0.15 configured -- resuming normal operations
project-pma-1 | [Sun Feb 06 05:03:03.718853 2022] [core:notice] [pid 1] AH00094: Command line: 'apache2 -D FOREGROUND'

Чтобы приостановить работу среды без изменения текущего состояния контейнеров, используйте команду:

```
docker-compose pause
```

Чтобы возобновить работу после приостановки, используйте команду:

```
docker-compose unpause
```

Команда `stop` останавливает выполнение контейнера, но не уничтожает данные, связанные с вашими контейнерами:

```
docker-compose stop
```

Если вы хотите удалить контейнеры, сети и тома, связанные с контейнерной средой, используйте команду `down`:

```
docker-compose down
```

Обратите внимание, что при этом не будет удален базовый образ, используемый Docker Compose для запуска нашей среды (в нашем примере `nginx:latest`). Так, при повторном запуске среды с помощью команды `docker-compose up` процесс будет намного быстрее, поскольку образ уже находится в вашей системе.

Если вы хотите удалить из системы базовый образ, используйте команду:

```
docker image rm nginx:latest
```

Практика 2: Запуск Python проекта через Docker Compose

Зачастую при собеседовании на позицию разработчика тестовое задание просят сделать доступным для запуска через docker-compose. Попробуем проделать подобную операцию. Создадим простое flask-приложение, которое взаимодействует с MongoDB.

Создадим структуру проекта и виртуальное окружение:

```
mkdir test_project
cd test_project
python3 -m venv venv
source venv/bin/activate
mkdir flask_app
```

Установим и запишем необходимые зависимости. В нашем случае это библиотеки `flask` и `pymongo`:

```
pip install flask
pip install pymongo
```

```
pip freeze > flask_app/requirements.txt
```

Создайте файл `app.py` в каталоге *flask_app*, который будет содержать простое python Flask Web-приложение:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import os
from flask import Flask, redirect, url_for, request, render_template
from pymongo import MongoClient

app = Flask(__name__)

client = MongoClient('mongodb', 27017)
db = client.tododb

@app.route('/')
def todo():

 _items = db.tododb.find()
 items = [item for item in _items]

 return render_template('todo.html', items=items)


@app.route('/new', methods=['POST'])
def new():

 item_doc = {
   'task': request.form['task'],
   'description': request.form['description']
 }
 db.tododb.insert_one(item_doc)

 return redirect(url_for('todo'))

if __name__ == "__main__":
 app.run(host='0.0.0.0', debug=True)
```

> Важно отметить, что в строке подключения в качестве имени хоста используется “mongodb“. Это имя хоста получает контейнер, запускаемый Docker Compose по имени запускаемого сервиса (mongodb). Контейнер web знает о контейнере mongodb, т.к. в его описании присутствует параметр links, указывающий на этот сервис (см. Обновление docker-compose.yml).

Создадим Dockerfile для запуска flask-приложения:

```
nano flask_app/Dockerfile
```

```
FROM ubuntu:latest
MAINTAINER Maxim Melnikov 'pylounge@mail.ru'
RUN apt-get update -qy
RUN apt-get install -qy python3.8 python3-pip python3.8-dev
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
CMD ["python3","app.py"]
```

Также в директории *flask_app* создадим директорию *templates*, в которой необходимо создать шаблон страницы `todo.html`, использующийся в вашем приложении:

```
mkdir flask_app/templates
nano flask_app/templates/todo.html
```

```
<code class="hljs javascript"><h1>Новая запись</h1>
<form action="/new" method="POST">
<table border="0">
<tbody><tr>
 <td>Дело</td>
 <td><input type="text" name="task"></td>
</tr><tr>
 <td>Описание</td>
 <td><input type="text" name="description"></td>
</tr>
<tr>
 <td></td><td><input type="submit" value="Сохранить"></td>
</tr>
</tbody></table>
</form>
<hr>
<h1>Список дел</h1>
<theader>
</theader>
<table border="1">
<tbody><tr>
 <td><b>Дело</b></td>
 <td><b>Описание</b></td>
</tr>
{% for item in items %}
</tbody><tbody><tr>
 <td>{{ item.task }} </td>
 <td>{{ item.description }}</td>
</tr>
{% endfor %}
</tbody>
</table>
</code>
```

Теперь необходимо создать основной файл конфигурации `docker-compose.yml`:

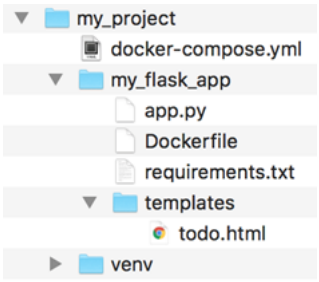
```
nano test_project/docker-compose.yml
```

```
version: '2'
services:
    web:
        build: ./flask_app
        ports:
           - "5000:5000"
        volumes:
           - ./flask_app:/app
        links:
            - mongodb
    mongodb:
           image: mongo:3.6
```

> Параметр links – это инструкция для Docker, которая заставит его обновить файл /etc/hosts контейнера web, внеся туда информацию об IP-адресе контейнера mongodb.

> Сервис mongodb, запускается из образа mongo версии 3.6. В сервис web, добавим параметр links, указывающий на сервис mongodb, чтобы контейнер web знал о контейнере mongodb после запуска.

В итоге проект должен иметь похожую структуру:

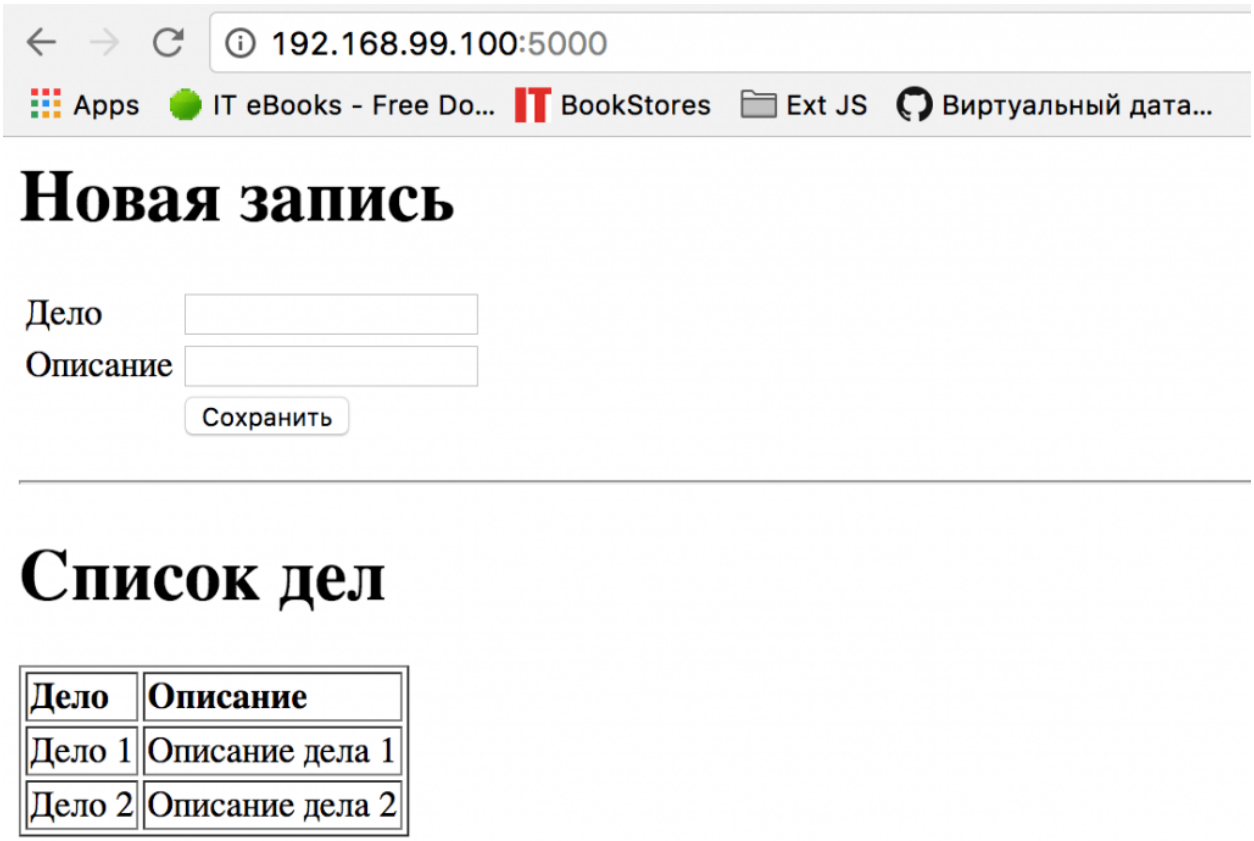


Структура проекта

Соберём и запустим приложение:

```
docker-compose up -d --build
```

После успешного выполнения этой команды в браузере переходим по адресу http://localhost:5000/ или http://127.0.0.1:5000/, либо если вы имеете публичный IP-адрес, то http://ваш_ip:5000/ .
Будет запущено приложение, состоящее уже из двух контейнеров: web с простым Flask приложением и mongodb с СУБД.



Приложение “Список дел” на языке Python

Аналогичным образом можно собрать и запустить Python-проект с любым количеством зависимостей.

Заключение

Таким образом, рассмотрели процедуры установки Docker Compose и настройки контейнерной среды на базе образа веб-сервера Nginx. Собрали и запустили сборку стека LEMP и запустили тестовый сайт на сервере из контейнеров.

Также мы увидели, как можно управлять этой средой с помощью команд Compose.

После освоения Docker и Docker Compose можно переходить к разбору такого понятия как оркестрация и инструментов оркестрации: Docker Swarm и Kubernates.

Список использованных источников

1. https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-compose-on-ubuntu-20-04-ru
2. https://habr.com/ru/company/ruvds/blog/450312/
3. https://webdevkin.ru/posts/raznoe/docker#subtitle4
4. https://mariadb.com/kb/en/setting-up-a-lamp-stack-with-docker-compose/
5. https://totaku.ru/ustanovka-lemp-s-pomoshchiu-dockera/
6. https://www.cloudreach.com/en/technical-blog/containerize-this-how-to-use-php-apache-mysql-within-docker-containers/
7. https://itisgood.ru/2019/02/05/kak-zapuskat-kontejnery-s-pomoshhju-docker-compose/
8. https://habr.com/ru/post/349704/
9. https://miac.volmed.org.ru/wiki/index.php/Docker-compose_настройка_для_сайта_NGINX_%2B_MYSQL_%2B_PHP-FPM
10. https://habr.com/ru/post/460173/
11. https://blog.sylo.space/guide-to-install-nginx-php-mariadb-phpmyadmin-in-docker/
12. https://tproger.ru/translations/yaml-za-5-minut-sintaksis-i-osnovnye-vozmozhnosti/