

Липецкий государственный технический университет

Кафедра прикладной математики

Отчет по лабораторной работе №6 «Контейнеризация.»

Студент

подпись, дата

Комолых Т.О.
фамилия, инициалы

Группа

ПМ-18

Руководитель

доц., к.п.н. кафедры АСУ
ученая степень, ученое звание

подпись, дата

Кургасов В. В.
фамилия, инициалы

Липецк 2020 г.

Содержание

Цель работы	3
Практическое задание	3
Выполнение практического задания	4
Вопросы для самопроверки	10
Вывод	13
Список литературы	14

Цель работы

Изучить современные методы разработки ПО в динамических и распределённых средах на примере контейнеров Docker.

Практическое задание

- 1) С помощью Docker Compose на своем компьютере поднять сборку nginx+php-fpm+postgres, продемонстрировать ее работоспособность, запустив внутри контейнера демо-проект на symfony. По умолчанию проект работает с sqlite-базой. Нужно заменить ее на postgres;
- 2) Создать новую БД в postgres.
- 3) Заменить DATABASE_URL в .env на строку подключения к postgres;
- 4) Создать схему БД и заполнить ее данными из фикстур, выполнив в консоли (
php bin/console doctrine:schema:create,
php bin/console doctrine:fixtures:load).

Проект должен открываться по адресу <http://demo-symfony.local/> (Код проекта должен располагаться в папке на локальном хосте). Для компонентов nginx, fpm есть готовые docker-образы, их можно и нужно использовать. Нужно распарить папки с локального хоста, настроить подключение к БД. В .env переменных для postgres нужно указать путь к папке, где будет лежать база, чтобы она не удалялась при остановке контейнера. На выходе должен получиться файл конфигурации docker-compose.yml и .env файл с настройками переменных окружения.

Дополнительные требования:

Postgres также должен работать внутри контейнера. В .env переменных нужно указать путь к папке на локальном хосте, где будут лежать файлы БД, чтобы она не удалялась при остановке контейнера.

Выполнение практического задания

Первоначально произведём клонирование текстового проекта с помощью команды «`get clone https://github.com/symfony/demo`», после чего совершаем переход в папку с проектом с помощью команды «`cd demo`».

Прежде чем запустить проект командой «`php -S localhost:8000 -t public/`», выполним установку php последней версии. После выполнения данного этапа в браузере стал доступен проект по адресу `http://localhost:8000`. Главное окно приложения имеет вид (рисунок 1).

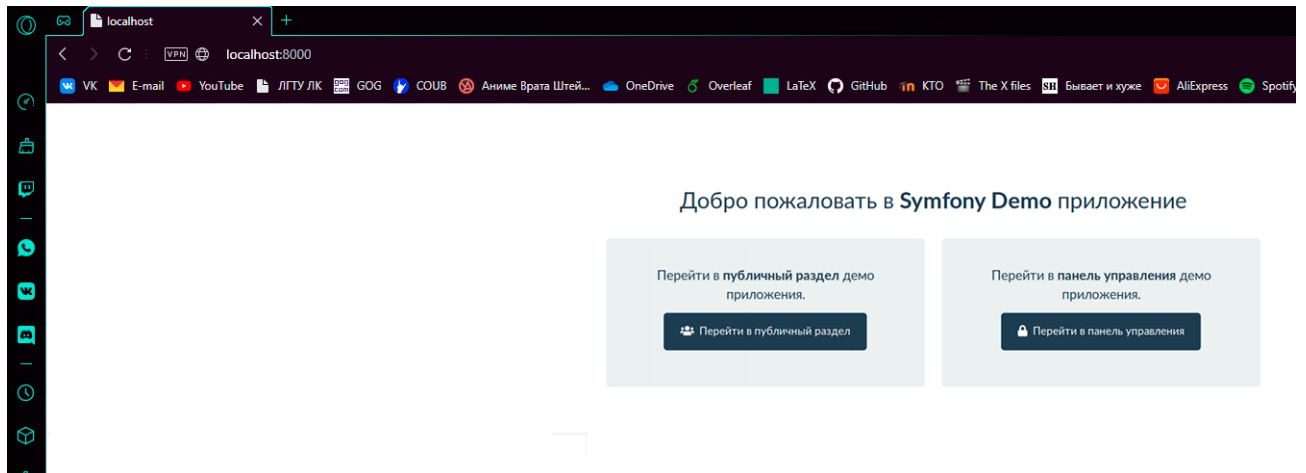


Рисунок 1.

Выполним установку бесплатной версии Docker на Ubuntu 19, который в официальном репозитории поставщика называется `docker-ce`. Для успешной установки выполним следующие команды:

- 1) `sudo apt install apt-transport-https ca-certificates curl software-properties-common`
- 2) `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -`
- 3) `sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu disco test"`
- 4) `sudo apt update`
- 5) `sudo apt install docker-ce`

Скачиваем файл и сохраняем его под другим именем `/usr/local/bin/docker-compose` (рисунок 2).

```
tatyana@arianrod:~/demo$ sudo curl -L "https://github.com/docker/compose/releases/download/1.27.4/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
[sudo] password for tatiana:
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100  651  100  651    0     0   1966      0  --:--:-- --:--:-- --:--:--  1966
100 11.6M  100 11.6M    0     0 1023K      0  0:00:11  0:00:11 --:--:-- 1291k
tatyana@arianrod:~/demo$ sudo chmod +x /usr/local/bin/docker-compose
```

Рисунок 2.

Изменим права доступа к данному файлу, позволив всем пользователям выполнять его (рисунок 3).

```
tatyana@arianrod:~$ sudo chmod +x /usr/local/bin/docker-compose
tatyana@arianrod:~$ ls /usr/local/bin/docker-compose
/usr/local/bin/docker-compose
tatyana@arianrod:~$ ls -l /usr/local/bin/docker-compose
-rwxr-xr-x 1 root root 1438 Dec 10 17:35 /usr/local/bin/docker-compose
```

Рисунок 3.

Создадим жёсткую символическую ссылку `/usr/bin/docker-compose` (рисунок 4).

```
tatyana@arianrod:~$ sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

Рисунок 4.

Выполним установку «`sudo apt install composer`». Composer – менеджер зависимостей PHP. Он отслеживает необходимые проекту библиотеки, извлекает и устанавливает их.

Следующим шагом установим postgresql, зайдём в консоль управления и создадим базу данных «mydatabase». Проверим наличие созданной базы данных (рисунок 5).

```
postgres=# \l
```

List of databases					
Name	Owner	Encoding	Collate	Ctype	Access privileges
mydatabase	postgres	UTF8	C.UTF-8	C.UTF-8	=Tc/postgres + postgres=CTc/postgres+ other=CTc/postgres + supuser=CTc/postgres + root=CTc/postgres
postgres	postgres	UTF8	C.UTF-8	C.UTF-8	
template0	postgres	UTF8	C.UTF-8	C.UTF-8	=c/postgres + postgres=CTc/postgres
template1	postgres	UTF8	C.UTF-8	C.UTF-8	=c/postgres + postgres=CTc/postgres

(4 rows)

Рисунок 5.

Отредактируем драйвера БД в файле `doctrine.yaml` (рисунок 6).

```

parameters:
    env(DATABASE_URL): ''

doctrine:
    dbal:
        driver: 'pdo_pgsql'
        server-version: '10.5'
        charset: utf8

        url: '%env(resolve:DATABASE_URL)%'

        # IMPORTANT: You MUST configure your server version,
        # either here or in the DATABASE_URL env var (see .env file)
        #server_version: '5.7'

    orm:
        auto_generate_proxy_classes: '%kernel.debug%'
        naming_strategy: doctrine.orm.naming_strategy.underscore_number_aware
        auto_mapping: true
        mappings:
            App:
                is_bundle: false
                type: annotation
                dir: '%kernel.project_dir%/src/Entity'
                prefix: 'App\Entity'
                alias: App

```

Рисунок 6.

Отредактируем настройки окружения в `.env` (рисунок 7).

```

sion=11&charset=utf8"
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
DATABASE_URL="postgresql://other:other@127.0.0.1:5432/mydatabase?serverVersion=11&charset=utf8"
###< doctrine/doctrine-bundle ###

###> symfony/mailer ###
MAILER_URL=null://localhost

```

Рисунок 7.

Далее загрузим схему БД командой «`php bin/console doctrine:schema:create`» (рисунок 8) и сами данные командой «`php bin/console doctrine:fixtures:load`» (рисунок 9).

```

tatyana@arianrod:~/demo$ php bin/console doctrine:schema:create_

Creating database schema...

[OK] Database schema created successfully!

```

Рисунок 8.

```

tatyana@arianrod:~/demo$ php bin/console doctrine:fixtures:load

Careful, database "/home/tatyana/demo/data/database.sqlite" will be purged. Do you want to continue
? (yes/no) [no]:
> y

> purging database
> loading App\DataFixtures\AppFixtures

```

Рисунок 9.

В папке с проектом создадим файл `Dockerfile` и заполним его следующим содержимым (рисунок 10):

```
FROM richarvey/nginx-php-fpm
WORKDIR /var>>/www/html/demo
COPY composer.json ./
RUN COMPOSER_MEMORY_LIMIT=-1 composer install
COPY . .
EXPOSE 8000
CMD ["php", "-S", "0.0.0.0:8000", "-t", "public/"]
```

Рисунок 10.

Создадим файл `docker-compose.yml` и заполним его следующим содержимым (рисунок 11):

```
version: "3"
services:
  app:
    container_name: docker-node-mongo
    restart: always
    build: .
    ports:
      - "80"
    links:
      - postgres
  postgres:
    container_name: postgres
    image: postgres
    ports:
      - "5432"
    environment:
      POSTGRES_USER: other
      POSTGRES_PASSWORD: other
```

Рисунок 11.

Выполним команду «`docker-compose up -d`», которая выполнит скачивание образов и установку зависимостей composer (рисунок 12).

```
tatyana@arianrod:~/demo$ sudo docker-compose up -d
Creating network "demo_default" with the default driver
Pulling postgres (postgres:...)
latest: Pulling from library/postgres
6ec7b7d162b2: Pull complete
a7d6065bfd90: Pull complete
1b99004acb42: Pull complete
e11ab41f0489: Pull complete
eaa59f6a7cf0: Pull complete
e27743da9368: Pull complete
10ffdd8ade29: Pull complete
bdaa066489bc: Pull complete
18fcf6a7b457: Pull complete
7e48453e674b: Pull complete
d6a3ea95116d: Pull complete
0cd33e8b20bb: Pull complete
75f979730af2: Pull complete
17a2798f3b3f: Pull complete
Digest: sha256:c846f37f65fd0d3d8b43040df5ebdc5856a038415b019ba596864848fb717a8b
Status: Downloaded newer image for postgres:latest
Building app
```

Рисунок 12.

Осуществим настройку контейнеров. Внутри папки текстового проекта создадим папку `docker`

«mkdir docker» и перейдём в неё «cd docker» (рисунок 13).

```
tatyana@arianrod:~$ cd demo
tatyana@arianrod:~/demo$ ls
CONTRIBUTING.md  composer.json      index.html         src               var
Dockerfile       composer.lock      info.php           start_encode_php5.tar.gz  vendor
LICENSE          config             migrations         symfony.lock     webpack.config.js
README.md         data               package.json       templates        yarn.lock
assets            docker             phpunit.xml.dist  tests
bin               docker-compose.yml public              translations
```

Рисунок 13.

Внутри папки docker создадим файл docker-compose.yaml с конфигурацией всех контейнеров (рисунок 14). Также в папке docker создадим папки nginx «mkdir nginx» и php-fpm «mkdir php-fpm» (рисунок 15).

```
version: '2'
services:
  postgres:
    image: postgres
    ports:
      - '5432'
    volumes:
      - ../var/lib/postgresql/data:/var/postgresql/data
  php:
    build: php-fpm
    ports:
      - '9002'
    volumes:
      - ../:/var/www/symfony:cached
      - ../logs/symfony:/var/www/symfony/var/logs:cached
    links:
      - postgres
  nginx:
    build: nginx
    ports:
      - '80'
    links:
      - php
    volumes_from:
      - php
    volumes:
      - ../logs/nginx:/var/log/nginx:cached
```

Рисунок 14.

```
tatyana@arianrod:~/demo/docker$ ls
docker-compose.yaml  nginx  php-fpm
```

Рисунок 15.

В папке nginx создадим файл Dockerfile с конфигурацией контейнера nginx (рисунок 16).

```
FROM nginx:latest
COPY default.conf/etc/nginx/conf.d/
```

Рисунок 16.

В папке `php-fpm` создадим файл `Dockerfile` с конфигурацией контейнера `php-fpm` (рисунок 17).

```
FROM php-fpm

RUN apt-get update
RUN apt-get install -y zlib1g-dev libpq-dev git libicu-dev libxml2-dev \
    && docker-php-ext-configure intl \
    && docker-php-ext-install intl \
    && docker-php-ext-configure pgsql -with-pgsql=/usr/local/pgsql \
    && docker-php-ext-install pdo pdo_pgsql pgsql \
    && docker-php-ext-install zip xml

WORKDIR /var/www/symfony_
```

Рисунок 17.

Теперь перезапустим все контейнеры командой «`sudo docker-compose up -d`». Перейдём в контейнер с идентификатором `18e3818f4aa6`, используя команду «`sudo docker exec -it 18e3818f4aa6 bash`», и внутри контейнера зайдём в консоль `psql` «`psql -U other`». Теперь можно создать базу данных внутри контейнера командой «`create database db;`».

```
tatyana@arianrod:~/demo$ sudo docker exec -it 18e3818f4aa6 bash
root@18e3818f4aa6:/# psql -U other
psql (13.1 (Debian 13.1-1.pgdg100+1))
Type "help" for help.

other=# create database db;
CREATE DATABASE
```

Рисунок 18.

Далее необходимо перейти в контейнер и выполнить загрузку схемы «`php bin/console doctrine:schema:create`» и данных «`php bin/console doctrine:fixtures:load`»

Теперь произведём тестовый запуск проекта в контейнерах (рисунок 19).

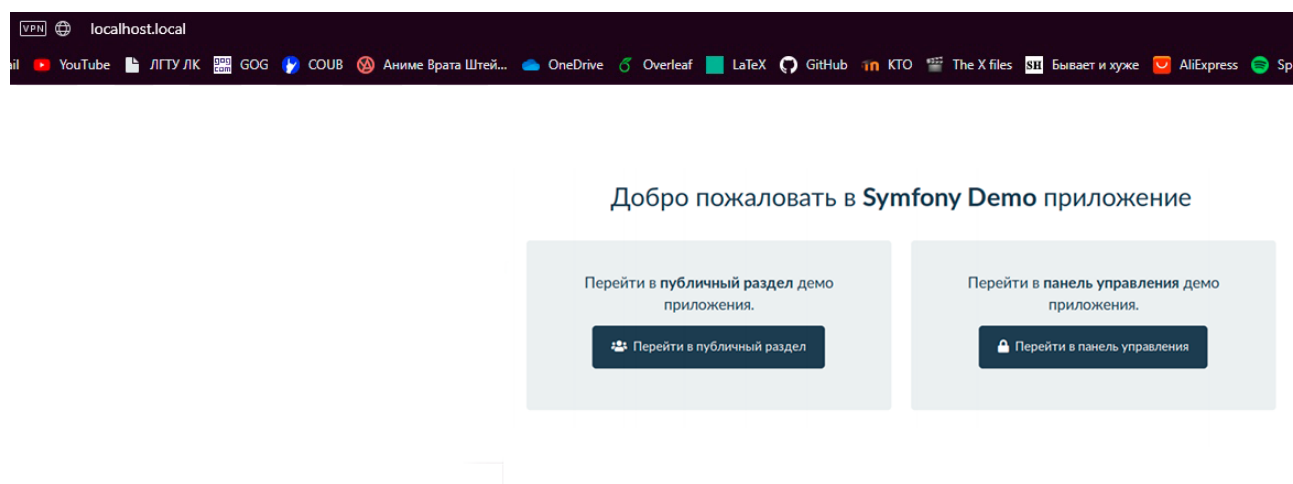


Рисунок 19.

Вопросы для самопроверки

Назовите отличия использования контейнеров по сравнению с виртуализацией.

А. Меньшие накладные расходы на инфраструктуру

Назовите основные компоненты Docker.

В. Контейнеры

Какие технологии используются для работы с контейнерами?

С. Контрольные группы (cgroups)

Найдите соответствие между компонентом и его описание:

образы – доступные только для чтения шаблоны приложений;

контейнеры – изолированные при помощи технологий операционной системы пользовательские окружения, в которых выполняются приложения;

реестры (репозитории) – сетевые хранилища образов.

В чём отличие контейнеров от виртуализации?

Виртуализация обеспечивает одновременную работу нескольких операционных систем на одном компьютере.

Контейнеры используют одно и то же ядро операционной системы и изолируют процессы приложения от остальной системы.

Для одновременной работы нескольких ОС в одной программе, которая реализует виртуализацию, требуется больше системных ресурсов, чем для аналогичной конфигурации на основе контейнеров. Ресурсы, как правило, не безграничны, поэтому, чем меньше «весят» приложения, тем плотнее их можно разместить на серверах. Все Linux-контейнеры, работающие на компьютере, используют одну и ту же ОС, поэтому приложения и сервисы остаются легковесными и работают в параллельном режиме, не тормозя друг друга.

Перечислите основные команды утилиты Docker с их кратким описанием.

`docker run` – Создание нового контейнера образа и его выполнение.

`docker start` – Запуск существующего контейнера. Данная команда используется в случае необходимости повторного запуска давшего сбой или совершившего выход контейнера.

`docker stop` – Остановка существующего контейнера Docker.

`docker build` – Сборка образа Docker из файла докер (`dockerfile`) и контекста сборки. Контекст сборки — это набор файлов, расположенных по определенному пути.

`docker ps` – Вывод списка запущенных контейнеров. Можно передать несколько параметров: `q` — «тихий» режим, в котором команда выводит только `id` контейнеров; `-a` — показывает все контейнеры, а не только запущенные.

`docker exec -ti` – Выполнение команды оболочки внутри конкретного контейнера.

Каким образом осуществляется поиск образов контейнеров?

Изначально Docker проверяет локальный репозиторий на наличие нужного образа. Если образ не найден, Docker проверяет удаленный репозиторий Docker Hub.

Каким образом осуществляется запуск контейнера?

Docker выполняет инициализацию и запуск ранее созданного по образу контейнера по имени. Для запуска Docker необходимо иметь права root-пользователя или пользователя из группы Docker, создаваемой автоматически во время установки сервиса. Если попробовать запустить в терминале Docker без этих прав или, не являясь пользователем группы docker, появится ситуация, когда не запускается контейнер.

Что значит управлять состоянием контейнеров?

Контейнеры Docker могут находиться в нескольких состояниях:

Исполняется (running) – контейнер работает. В выводе `docker ps` можно увидеть статус «Up» и время, в течение которого он исполняется.

Создан (created) – контейнер создан, но в настоящий момент не выполняется. Такое состояние будет у контейнера после команды `docker create`.

Завершил исполнение (exited) – контейнер завершил исполнение.

Поставлен на паузу (paused) – процесс контейнера остановлен, но существует. Поставить контейнер на паузу можно при помощи команды `docker pause`.

Запущен заново (restarting) – контейнер рестартует. Остановленный контейнер можно рестартовать. Рестарт означает, что контейнер заново запустится с теми же идентификатором и настройками сети, что были до остановки. Однако это будет уже другой процесс с другим PID.

«Умер» (dead) – контейнер перестал функционировать вследствие сбоя.

Управление состоянием контейнеров - это контроль хода выполнения контейнера и перевод его в конкретный режим в любой момент времени.

Как изолировать контейнер?

Для изоляции контейнера достаточно правильно сконфигурировать файлы Dockerfile и `docker-compose.yml`. По умолчанию контейнеры запускаются от прав root пользователя.

Опишите последовательность создания новых образов, назначение Dockerfile?

Dockerfile - это специальный скрипт, в котором содержатся команды и инструкции, которые будут в заданной последовательности выполнены для сборки нового образа Docker.

Для создания нового образа необходимо:

- 1) Указать базовый образ для создания нового. Эта команда должна быть первой в Dockerfile.
- 2) Добавить необходимые слои.
- 3) Выполнить нужные операции.
- 4) Развернуть рабочее окружение внутри контейнера с необходимыми зависимостями.

Движок Docker-а при запуске распарсит Dockerfile и создаст из него соответствующий образ (Image), который был описан.

Возможно ли работать с контейнерами Docker без одноимённого движка?

С контейнерами Docker можно работать без одноимённого движка в среде другой виртуализации Kubernetes.

Опишите назначение системы оркестрации контейнеров Kubernetes. Перечислите основные объекты Kubernetes.

Kubernetes — это система с открытым исходным кодом, предназначенная для оркестрации контейнеров. Она позволяет создавать, обновлять и масштабировать контейнеры, не беспокоясь о вынужденном простое. Использование Kubernetes позволяет организовать многократное использование контейнеров и переключение между ними.

Основные объекты:

- 1) Поды — неделимая единица в платформе Kubernetes. При создании развёртывания в Kubernetes, создаются поды с контейнерами внутри (в отличие от непосредственного создания контейнеров). Каждый Pod-объект связан с узлом, на котором он размещён, и остаётся там до окончания работы (согласно стратегии перезапуска) либо удаления. В случае неисправности узла такой же под будет распределён на другие доступные узлы в кластере.
- 2) Узел — это рабочая машина в Kubernetes, которая в зависимости от кластера может быть либо виртуальной, либо физической. Каждый узел управляется мастером (ведущим узлом). Узел может содержать несколько подов, которые мастер Kubernetes автоматически размещает на разные узлы кластера. Ведущий узел при автоматическом планировании (распределении подов по узлам) учитывает доступные ресурсы на каждом узле.
- 3) Том - общий ресурс хранения для совместного использования из контейнеров, развёрнутых в пределах одного пода.
- 4) Все объекты управления (узлы, поды, контейнеры) в Kubernetes помечаются метками, селекторы меток — это запросы, которые позволяют получить ссылку на объекты, соответствующие какой-то из меток; метки и селекторы — это главный механизм Kubernetes, который позволяет выбрать, какой из объектов следует использовать для запрашиваемой операции.
- 5) Сервисом в Kubernetes называют совокупность логически связанных наборов подов и политик доступа к ним.
- 6) Контроллер — это процесс, который управляет состоянием кластера, пытаясь привести его от фактического к желаемому; он делает это, оперируя набором подов, который определяется с помощью селекторов меток, являющихся частью определения контроллера.
- 7) Операторы — специализированный вид программного обеспечения Kubernetes, предназначенный для включения в кластер сервисов, сохраняющих своё состояние между выполнениями, таких как СУБД, системы мониторинга или кэширования. Назначение операторов — предоставить возможность управления stateful-приложениями в кластере Kubernetes прозрачным способом и скрыть подробности их настроек от основного процесса управления кластером Kubernetes.

Вывод

В ходе лабораторной работы были изучены современные методы разработки ПО в динамических и распределённых средах на примере контейнеров Docker.

Список литературы

- [1] Львовский, С.М. Набор и верстка в системе L^AT_EX [Текст] / С.М. Львовский. М.: МЦНМО, 2006. — 448 с.
- [2] Badcode. Docker самый простой и понятный tutorial. Изучаем докер, так, если бы он был игровой приставкой: <https://badcode.ru/docker-tutorial-dlia-novichkov-rassmatrivaem-docker-tak-iesli-by-on-byi-ighrovoi-pristavkoi/> (дата обращения: 02.02.2021). - Текст: электронный.
- [3] jeka.by. Работаем с PostgreSQL через командную строку в Linux: <https://jeka.by/post/1104/postgresql-connect-via-terminal/> (дата обращения: 02.02.2021). - Текст: электронный.