

**Липецкий государственный технический университет**

**Факультет автоматизации и информатики**

**Кафедра автоматизированных систем управления**

**ЛАБОРАТОРНАЯ РАБОТА №6**

**по дисциплине «OS Linux»**

**Контейнеризация**

Студент

Сухоруких А.О.

Группа АС-18

Руководитель

Кургасов В.В.

К.н.

Липецк 2020 г.

## Оглавление

Цель работы .....	3
1 Ход работы.....	4
1.1 Часть 1 .....	4
1.2 Часть 2 .....	9
Вывод.....	11

## Цель работы

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

## 1 Ход работы

### 1.1 Часть 1

Ознакомившись с официальной документацией установим docker, docker-compose. После этого устанавливаем менеджер пакетов для php – composer.

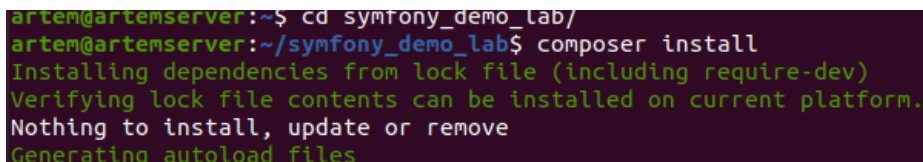
Клонируем проект Symfony Demo Application. Процесс клонирования проекта показан на рисунке 1.



```
artem@artemserver:~$ composer create-project symfony/symfony-demo symfony_demo_lab
Creating a "symfony/symfony-demo" project at "./symfony_demo_lab"
Installing symfony/symfony-demo (v1.6.4)
- Downloading symfony/symfony-demo (v1.6.4)
- Installing symfony/symfony-demo (v1.6.4): Extracting archive
Created project in /home/artem/symfony_demo_lab
Installing dependencies from lock file (including require-dev)
Verifying lock file contents can be installed on current platform.
Package operations: 185 installs, 0 updates, 0 removals
- Downloading symfony/routing (v5.2.1)
- Downloading symfony/http-foundation (v5.2.1)
- Downloading symfony/event-dispatcher (v5.2.1)
- Downloading symfony/var-dumper (v5.2.1)
- Downloading symfony/error-handler (v5.2.1)
- Downloading symfony/http-kernel (v5.2.1)
- Downloading symfony/finder (v5.2.1)
- Downloading symfony/filesystem (v5.2.1)
- Downloading symfony/dependency-injection (v5.2.1)
- Downloading symfony/config (v5.2.1)
- Downloading symfony/cache (v5.2.1)
- Downloading symfony/framework-bundle (v5.2.1)
```

Рисунок 1 – Клонирование проекта

С помощью команды `composer install` устанавливаем необходимые пакеты для проекта. Результат выполнения показан на рисунке 2.



```
artem@artemserver:~$ cd symfony_demo_lab/
artem@artemserver:~/symfony_demo_lab$ composer install
Installing dependencies from lock file (including require-dev)
Verifying lock file contents can be installed on current platform.
Nothing to install, update or remove
Generating autoload files
```

Рисунок 2 – Обновление и установка нужных пакетов

Следующий шаг, нам необходимо установить postgresql и создать там новую базу данных. Создадим базу данных с названием «dbsymfony». Результат создания базы данных показан на рисунке 3.

```
artem@artemserver:~/symfony_demo_lab$ psql
psql (12.5 (Ubuntu 12.5-0ubuntu0.20.04.1))
Type "help" for help.

artem=# \l

               List of databases
   Name      | Owner   | Encoding | Collate | Ctype   | Access privileges
-----+-----+-----+-----+-----+-----
 artem       | artem   | UTF8     | C.UTF-8 | C.UTF-8 | 
 dbsymfony   | artem   | UTF8     | C.UTF-8 | C.UTF-8 | 
 postgres    | postgres | UTF8     | C.UTF-8 | C.UTF-8 | 
 symfonydemo | artem   | UTF8     | C.UTF-8 | C.UTF-8 | 
 template0   | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres +
              |          |          |          |          | postgres=Ct/postgres
 template1   | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres +
              |          |          |          |          | postgres=Ct/postgres
 tmpLab8     | postgres | UTF8     | C.UTF-8 | C.UTF-8 | 
(7 rows)

artem=#
```

Рисунок 3 – Создание базы данных

Дальше нам необходимо отредактировать файл .env, для подключения новой базы данных.

```
GNU nano 4.8 .env
#
# DO NOT DEFINE PRODUCTION SECRETS IN THIS FILE NOR IN ANY OTHER COMMITTED FILE
#
# Run "composer dump-env prod" to compile .env files for production use (required)
# https://symfony.com/doc/current/best_practices.html#use-environment-variables

###> symfony/framework-bundle ###
APP_ENV=dev
APP_SECRET=2ca64f8d83b9e89f5f19d672841d6bbb
#TRUSTED_PROXIES=127.0.0.0/8,10.0.0.0/8,172.16.0.0/12,192.168.0.0/16
#TRUSTED_HOSTS='^(localhost|example\..com)$'
###< symfony/framework-bundle ###

###> doctrine/doctrine-bundle ###
# Format described at https://www.doctrine-project.org/projects/doctrine-dbal/
# For a MySQL database, use: "mysql://db_user:db_password@127.0.0.1:3306/db_name"
# For a PostgreSQL database, use: "postgresql://db_user:db_password@127.0.0.1:5432/db_name"
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
DATABASE_URL=postgresql://artem:qwerty@postgres:5432/dbsymfony?serverVersion=12
###< doctrine/doctrine-bundle ###

###> symfony/mailer ###
# MAILER_DSN=smtplib://localhost
###< symfony/mailer ###
```

Рисунок 4 – Файл .env

Загружаем схему БД командой `php bin/console doctrine:schema:create` и заполняем данными с помощью команды `php bin/console doctrine:fixtures:load`. Проверяем работоспособность проекта.

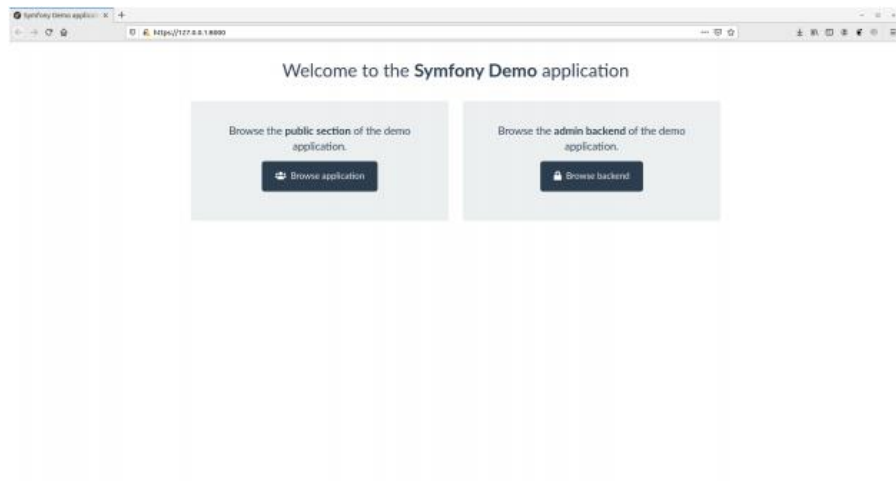


Рисунок 5 – Страница проекта

Далее нам необходимо настроить контейнеры. Создадим файл `docker-compose.yml`. Содержание файла:

```
version: "3"
services:
  php-fpm:
    container_name: php-fpm
    build:
      context: .
      dockerfile: docker/php.Dockerfile
    volumes:
      - ./:/var/www/symfony:rw
      - ./logs/symfony:/var/www/symfony/log:rw
    links:
      - postgres

  nginx:
    container_name: nginx
    build:
      context: .
      dockerfile: docker/nginx.Dockerfile
    ports:
```

- "8084:80"

volumes:

- ./:/var/www/symfony:rw
- ./logs/nginx:/var/log/nginx:rw

links:

- php-fpm

postgres:

container\_name: postgres

image: postgres

environment:

POSTGRES\_DB: dbsymfony

POSTGRES\_USER: artem

POSTGRES\_PASSWORD: qwerty

volumes:

- ./data/postgresql:/var/lib/postgresql/data:rw

ports:

- 5432:5432

Теперь создадим папку docker и внутри нее создади 2 файла: nginx.Dockerfile и php.Dockerfile, а также каталог conf, содержащий файл конфигурации nginx vhost.conf.

Файл nginx.Dockerfile

FROM nginx:latest

COPY ./docker/conf/vhost.conf /etc/nginx/conf.d/default.conf

WORKDIR /var/www/symfony

Файл vhost.conf

server {

listen 80;

```
root /var/www/symfony/public;
```

```
server_name _;
```

```
error_log /var/log/nginx/symfony_error.log;
```

```
access_log /var/log/nginx/symfony_access.log;
```

```
location / {
```

```
    try_files $uri /$uri /index.php?$query_string;
```

```
}
```

```
location ~ ^/index\.php(/|$) {
```

```
    fastcgi_pass php-fpm:9000;
```

```
    fastcgi_split_path_info ^(.+\.php)(/.*)$;
```

```
    include fastcgi_params;
```

```
    fastcgi_param
```

SCRIPT\_FILENAME

```
$document_root$fastcgi_script_name;
```

```
    fastcgi_param HTTPS off;
```

```
}
```

```
}
```

Файл php.Dockerfile

FROM php:7.4-fpm

WORKDIR /var/www/symfony

RUN apt-get update && apt-get install -y \

libpq-dev \

wget \

zlib1g-dev \

libmcrypt-dev \

libzip-dev



RUN `docker-php-ext-install pdo pdo_pgsql pgsql`

CMD `php-fpm`

Следующим шагом запускаем все контейнеры в фоне, с помощью команды `docker-compose up -d`, переходим в контейнер с `postgresql` (команда `docker exec -it postgres bash`), внутри контейнера переходим в консоль `psql` и создаем БД `dbsymfony`. Переходим в контейнер с `php` и загрузить схему БД (команда `php bin/console doctrine:schema:create`) и данные для БД (команда `php bin/console doctrine:fixtures:load`). После этого редактируем файл `/etc/hosts` и добавляем псевдоним адресу `127.0.0.1 demo-symfony.local`. Результат выполнения команд показан на рисунках 6, 7.

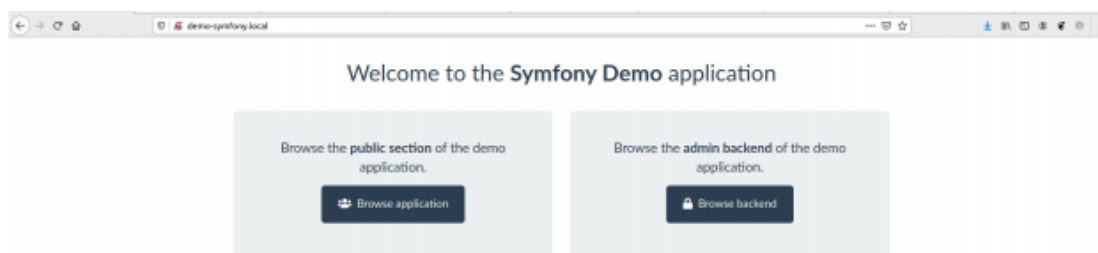


Рисунок 6 — Запуск проекта в контейнерах

## 1.2 Часть 2

Нам необходимо удалить конфигурацию контейнера с `postgresql` и произвести подключение к базе данных локально. Для этого нам необходимо знать `ip` локальной машины (команда `hostname -I | cut -d ' ' -f1`) и добавить этот `ip` под псевдонимом `bd` в файл `docker-compose.yml`.

```

1 version: "3"
2 services:
3   php-fpm:
4     container_name: php-fpm
5     build:
6       context: .
7       dockerfile: docker/php.Dockerfile
8     volumes:
9       - ./var/www/symfony:rw
10      - ./logs/symfony/:/var/www/symfony/log:rw
11     extra_hosts:
12       - "db:172.28.0.1"
13
14   nginx:
15     container_name: nginx
16     build:
17       context: .
18       dockerfile: docker/nginx.Dockerfile
19     ports:
20       - "8084:80"
21     volumes:
22       - ./var/www/symfony:rw
23       - ./logs/nginx/:/var/log/nginx:rw

```

Рисунок 7 – Файл docker-compose.yml

Чтобы она допустить подключение из контейнера, изменим файлы конфигурации. Для этого изменим файлы конфигурации postgresql.conf и pg\_hba.conf. Изменение файлов показано на рисунках 8, 9.

```

#-----
# CONNECTIONS AND AUTHENTICATION
#-----

# - Connection Settings -

#listen_addresses = '*'          # what IP address(es) to listen on;
                                # comma-separated list of addresses;
                                # defaults to 'localhost'; use '*' for
                                # (change requires restart)
port = 5432                     # (change requires restart)
max_connections = 100           # (change requires restart)
#superuser_reserved_connections = 3 # (change requires restart)
unix_socket_directories = '/var/run/postgresql' # comma-separated list of dire
                                # (change requires restart)
#unix_socket_group = ''         # (change requires restart)
#unix_socket_permissions = 0777 # begin with 0 to use octal notation
                                # (change requires restart)
#bonjour = off                  # advertise server via Bonjour
                                # (change requires restart)
#bonjour_name = ''              # defaults to the computer name
                                # (change requires restart)

# - TCP settings -

```

Рисунок 8 – Изменение файла postgresql.conf

```

# "local" is for Unix domain socket connections only
local    all             all                                peer
# IPv4 local connections:
host     all             all 127.0.0.1/32                  md5
host     all             all 0.0.0.0/0                  md5
# IPv6 local connections:
host     all             all ::1/128                  md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
local    replication     all                                peer
host     replication     all 127.0.0.1/32                  md5

```

Рисунок 9 – Изменение файла pg\_hba.conf

## Вывод

В ходе выполнения лабораторной работы мы изучили современные методы разработки ПО в динамических и распределенных средах на примере контейнеров Docker.

## Ответы на контрольные вопросы

1. Назовите отличия использования контейнеров по сравнению с виртуализацией.

А. Меньшие накладные расходы на инфраструктуру

2. Назовите основные компоненты Docker.

В. Контейнеры

3. Какие технологии используются для работы с контейнерами?

С. Контрольные группы (cgroups)

4. Найдите соответствие между компонентом и его описанием:

— 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.

5. В чем отличие контейнеров от виртуализации?

В отличие от виртуальной машины, обеспечивающей аппаратную виртуализацию, контейнер обеспечивает виртуализацию на уровне операционной системы с помощью абстрагирования пользовательского пространства. В целом контейнеры выглядят как виртуальные машины. Например, у них есть изолированное пространство для запуска приложений, они позволяют выполнять команды с правами суперпользователя, имеют частный сетевой интерфейс и IP-адрес, пользовательские маршруты и правила межсетевого экрана и т. д. Одна большая разница между контейнерами и виртуальными машинами в том, что контейнеры разделяют ядро хоста с другими контейнерами.

6. Перечислите основные команды утилиты Docker с их кратким описанием.

— `docker ps` — показывает список запущенных контейнеров;

— `docker pull` — скачать определённый образ или набор образов (репозиторий);

— `docker build` — эта команда собирает образ Docker из Dockerfile и «контекста»;

— `docker run` — запускает контейнер, на основе указанного образа;

— `docker logs` — эта команда используется для просмотра логов указанного контейнера;

— `docker volume ls` — показывает список томов, которые являются предпочитаемым механизмом для сохранения данных, генерируемых и используемых контейнерами Docker;

— `docker rm` — удаляет один и более контейнеров;

— `docker rmi` — удаляет один и более образов;

— `docker stop` — останавливает один и более контейнеров;

— `docker exec -it ...` - выполняет команду в определенном контейнере

7. Каким образом осуществляется поиск образов контейнеров?

Сначала проверяется локальный репозиторий на наличия нужного контейнера, если он не найден локально, то поиск производится в репозитории Docker Hub.

8. Каким образом осуществляется запуск контейнера?

Для запуска контейнера его необходимо изначально создать из образа, поэтому изначально контейнер собирается с помощью команды `docker build`, а уже затем запускается с помощью команды `docker run`.

9. Что значит управлять состоянием контейнеров?

Это означает, что в любой момент времени есть возможность запустить, остановить или выполнить команды внутри контейнера.

10. Как изолировать контейнер?

Контейнеры уже по сути своей являются изолированными единицами, поэтому достаточно без ошибок сконфигурировать файлы Dockerfile и/или `docker-compose.yml`.

11. Опишите последовательность создания новых образов, назначение Dockerfile?

Производится выбор основы для нового образа на Docker Hub, далее производится конфигурация Dockerfile, где описываются все необходимые пакеты, файлы, команды и т.п. Dockerfile — это текстовый файл с инструкциями, необходимыми для создания образа контейнера. Эти инструкции включают идентификацию существующего образа, используемого в качестве основы, команды, выполняемые в процессе создания образа, и команду, которая будет выполняться при развертывании новых экземпляров этого образа контейнера.

12. Возможно ли работать с контейнерами Docker без одноименного движка?

Да, с использованием Kubernetes

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

Kubernetes — открытое программное обеспечение для автоматизации развёртывания, масштабирования контейнеризированных приложений и управления ими.

Поддерживает основные технологии контейнеризации, включая Docker, rkt, также возможна поддержка технологий аппаратной виртуализации.

— Nodes: Нода это машина в кластере Kubernetes.

— Pods: Pod это группа контейнеров с общими разделами, запускаемых как единое целое.

— Replication Controllers: replication controller гарантирует, что определенное количество «реплик» pod'ы будут запущены в любой момент времени.

— Services: Сервис в Kubernetes это абстракция которая определяет логический объединённый набор pod и политику доступа к ним.

— Volumes: Volume(раздел) это директория, возможно, с данными в ней, которая доступна в контейнере.

— Labels: Label'ы это пары ключ/значение которые прикрепляются к объектам, например pod'ам. Label'ы могут быть использованы для создания и выбора наборов объектов.

— Kubectl Command Line Interface: kubectl интерфейс командной строки для управления Kubernetes.