

Липецкий государственный технический университет

Отчет по Лабораторной работе № 6 по дисциплине «Операционная система Linux» на тему «Контейнеризация»

Студент

Руководитель

доцент, к.п.н.

учёная степень, учёное звание

подпись, дата

подпись, дата

Елфимова Д.А.

фамилия, инициалы

Кургасов В.В.

фамилия, инициалы

Липецк 2019 г.

Задание

С помощью Docker Compose на своем компьютере поднять сборку nginx+php-fpm+postgres, продемонстрировать ее работоспособность, запустив внутри контейнера демо-проект на symfony (Исходники взять отсюда <https://github.com/symfony/demo> /ссылка на github/). По умолчанию проект работает с sqlite-базой. Нужно заменить ее на postgres. (Для этого: 1. Создать новую БД в postgres; 2. Заменить DATABASE_URL в /.env на строку подключения к postgres; 3. 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) контейнеры с fpm и nginx должны его подхватывать.

Для компонентов nginx, fpm есть готовые docker-образы, их можно и нужно использовать. Нужно расшарить папки с локального хоста, настроить подключение к БД. В .env переменных для постгреса 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

1. Ход работы

1. Клонировать тестовый проект, запустить его и открыть в браузере
Перед этим необходимо установить следующее

```
cd $HOME
```

```
sudo apt-get install php7.2-fpm
```

```
sudo apt-get install php7.2-sqlite
```

```
sudo apt-get install php7.2-xml
```

```
sudo apt-get install php7.2-mbstring
```

```
wget https://get.symfony.com/cli/installer -O - | bash
```

```
sudo mv /home/user/.symfony/bin/symfony  
/usr/local/bin/symfony
```

```
symfony check:requirements
```

```
git clone https://github.com/symfony/symfony-demo demo
```

```
sudo apt-get install composer
```

```
cd demo
```

```
composer install --no-interaction
```

```
wget https://getcomposer.org/installer
```

```
php bin/console server:run
```

```
elfida@elfida:~/demo$ php bin/console server:run

[OK] Server listening on http://127.0.0.1:8000

// Quit the server with CONTROL-C.

PHP 7.2.24-0ubuntu0.18.04.1 Development Server started at Fri Nov 29 00:50:53 2019
Listening on http://127.0.0.1:8000
Document root is /home/elfida/demo/public
Press Ctrl-C to quit.
```

Рисунок 1. Запуск

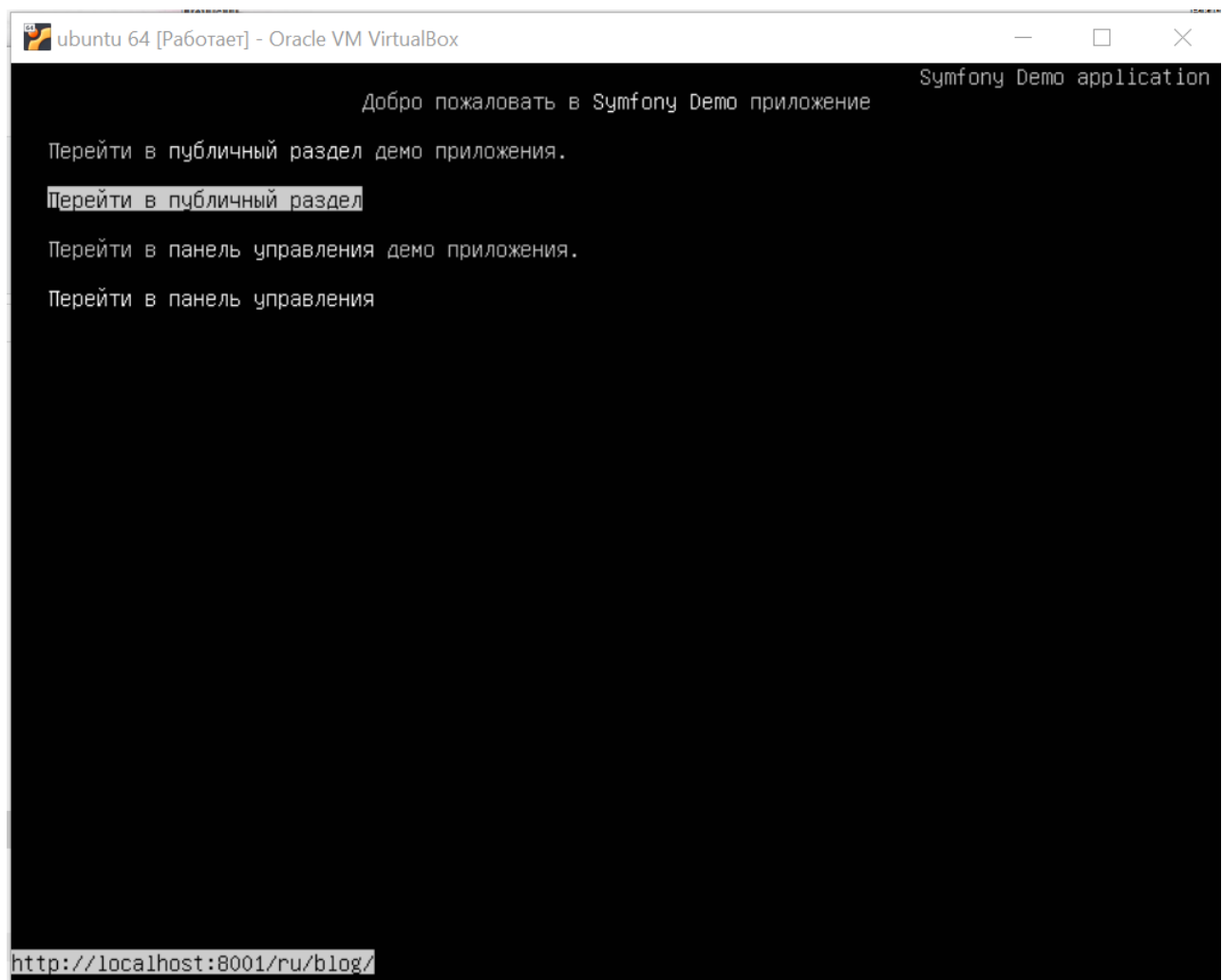


Рисунок 2. Запуск результата в текстовом браузере

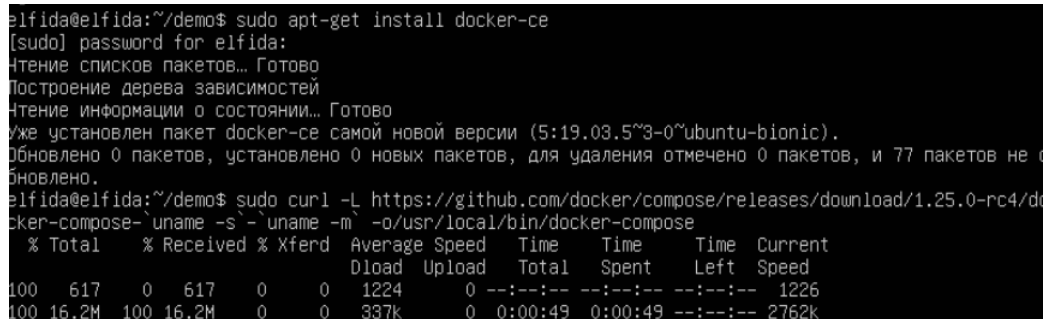
2. Установить Docker и Docker Compose

```

sudo apt-get install docker-ce
sudo curl -L https://github.com/docker/compose/releases/download/1.25.0-rc4/docker-compose -`uname -s` -`uname -m` -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

```

Рисунок 3. Установка



```

elfida@elfida:~/demo$ sudo apt-get install docker-ce
[sudo] password for elfida:
Чтение списков пакетов... Готово
Построение дерева зависимостей
Чтение информации о состоянии... Готово
Уже установлен пакет docker-ce самой новой версии (5:19.03.5~3-0~ubuntu-bionic).
Обновлено 0 пакетов, установлено 0 новых пакетов, для удаления отмечено 0 пакетов, и 77 пакетов не
обновлено.
elfida@elfida:~/demo$ sudo curl -L https://github.com/docker/compose/releases/download/1.25.0-rc4/d
docker-compose -`uname -s` -`uname -m` -o /usr/local/bin/docker-compose
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100 617      0 617    0     0  1224      0 --:--:-- --:--:-- --:--:--  1226
100 16.2M  100 16.2M    0     0  337k      0  0:00:49  0:00:49 --:--:-- 2762k

```

Рисунок 4. Установка - 2

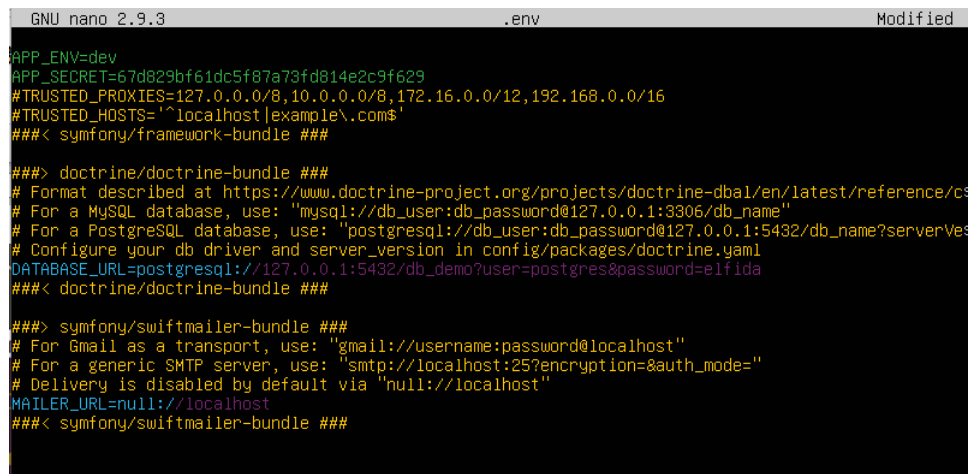
3. Создание бд

```

sudo su - postgres
psql
create database db_demo;
\password;

```

4. Изменение файла .env и doctrine.yaml(открытие локального проек-та без использования докер-контейнеров)



```

GNU nano 2.9.3 .env Modified
APP_ENV=dev
APP_SECRET=67d829bf61dc5f87a73fd814e2c9f629
#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/en/latest/reference/connections.html
# 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?serverVersion=12"
# Configure your db driver and server_version in config/packages/doctrine.yaml
DATABASE_URL=postgresql://127.0.0.1:5432/db_demo?user=postgres&password=elfida
###< doctrine/doctrine-bundle ###

###> symfony/swiftmailer-bundle ###
# For Gmail as a transport, use: "gmail://username:password@localhost"
# For a generic SMTP server, use: "smtp://localhost:25?encryption=&auth_mode="
# Delivery is disabled by default via "null://localhost"
MAILER_URL=null://localhost
###< symfony/swiftmailer-bundle ###

```

Рисунок 5. .env

```

GNU nano 2.9.3 doctrine.yaml Modified
parameters:
  # Adds a fallback DATABASE_URL if the env var is not set. This allows you
  # to run cache:warmup even if your environment variables are not available
  # yet. You should not need to change this value.
  env(DATABASE_URL): ''

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

    # IMPORTANT: when not use SQLite, you MUST configure your db driver and
    # server version, either here or in the DATABASE_URL env var (see .env file)
    # driver: 'pdo_pgsql'
    # server_version: '10.5'

```

Рисунок 6. .yaml

5. Создадим папку docker, в которой будут наши файлы для сборки трёх контейнеров: nginx+php-fpm+postgres. В папке будут содержаться следующие файлы: docker-compose.yaml(1); nginx/default.conf(2); nginx/Dockerfile(3); php-fpm/Dockerfile(4).

(1)

```
version: '2'
```

```
services:
```

```
  postgres:
```

```
    image: postgres
```

```
    ports:
```

```
      - '5435:5432'
```

```
  php:
```

```
    build: php-fpm
```

```
    ports:
```

```
      - '9002:9000'
```

```
    volumes:
```

```
      - ../:/var/www/symfony:cached
```

```
      - ./logs/symfony:/var/www/symfony/var/logs:cached
```

```
    links:
```

```
      - postgres
```

```
  nginx:
```

```
    build: nginx
```

```
    ports:
```

```
      - '8080:80'
```

```
    links:
```

```
      - php
```

```
    volumes_from:
```

```
      - php
```

```
    volumes:
```

```
      - ./logs/nginx:/var/log/nginx:cached
```

(2)

```
server {
    listen      80;
    server_name localhost;
    root /var/www/symfony/public;

    location / {
        try_files $uri @rewriteapp;
    }

    location @rewriteapp {
        rewrite ^(.*)$ /index.php/$1 last;
    }

    location ~ ^/index\.php(/|$) {
        fastcgi_pass php:9000;
        fastcgi_split_path_info ^(.+\.php)(/.*)$;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$
            fastcgi_script_name;
        fastcgi_param HTTPS off;
    }

    error_log /var/log/nginx/symfony_error.log;
    access_log /var/log/nginx/symfony_access.log;
}
```

(3)

```
FROM nginx:latest
```

```
COPY default.conf /etc/nginx/conf.d/
```

(4)

```
FROM php:7.2-fpm
```

```
RUN apt-get update
```

```
RUN apt-get install -y zlib1g-dev libpq-dev git
    libcu-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
```

```

FROM nginx-php-fpm
WORKDIR /var/www/html/demo
COPY composer.json ./
RUN composer install
COPY ..
EXPOSE 8000
CMD ["php", "bin/console", "server:start"]

```

Рисунок 7. Dockerfile

6. Файл docker-compose.yml

```

GNU nano 2.9.3 docker-compose.yml
version: "3"
services:
  app:
    container_name: docker-node-mongo
    restart: always
    build: .
    ports:
      - "80:8000"
    links:
      - postgres
  postgres:
    container_name: postgres
    image: postgres
    ports:
      - "5432:5432"

```

Рисунок 8. docker-compose.yml

7. Заполнение БД данными

```

postgres=# \q
elfida@elfida:~/demo$ php bin/console doctrine:schema:create
! [CAUTION] This operation should not be executed in a production environment!
!
Creating database schema...

[OK] Database schema created successfully!

elfida@elfida:~/demo$ php bin/console doctrine:fixtures:load
Careful, database "demo_db" will be purged. Do you want to continue? (yes/no) [no]:
> y
> purging database
> loading App\DataFixtures\AppFixtures
elfida@elfida:~/demo$ _

```

Рисунок 9. docker-compose.yml

8. Работа в браузере с бд postgresql

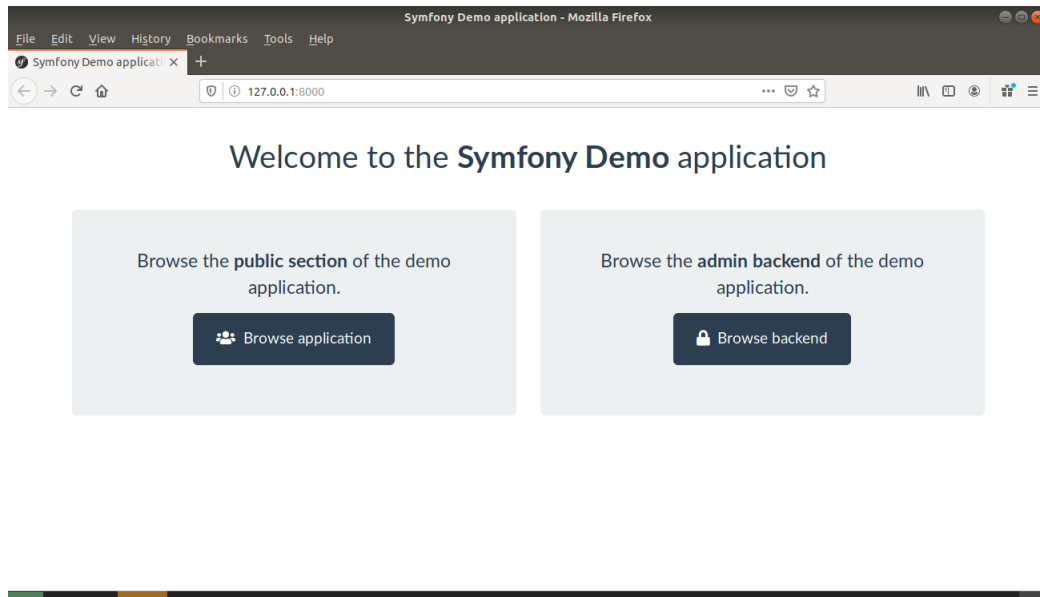


Рисунок 10. Работа в браузере

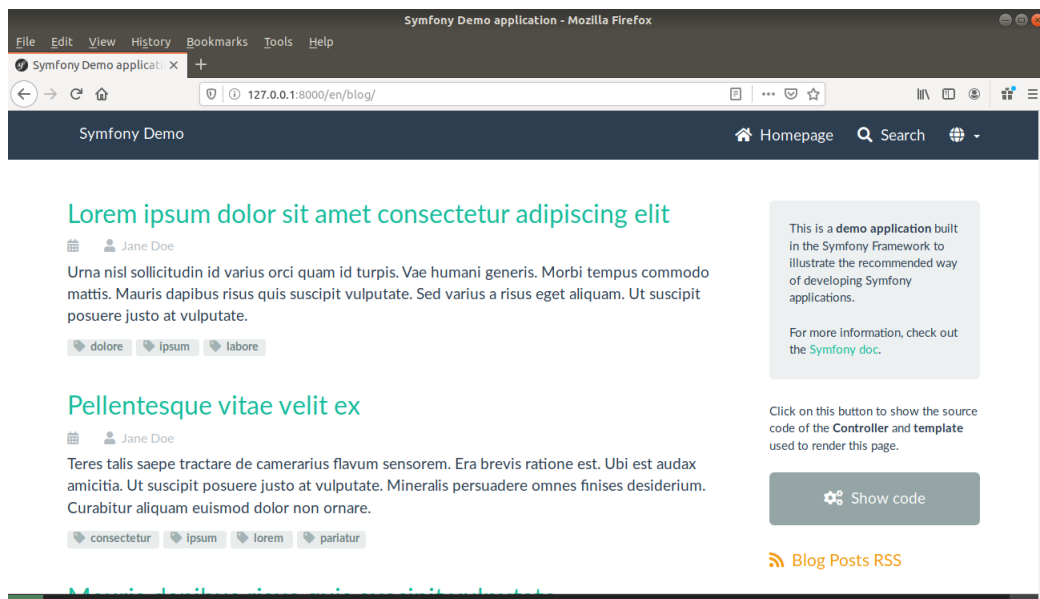


Рисунок 11. Работа в браузере - 2

9. Изменим файл .env, для возможности последующей работы с БД

```
###> symfony/framework-bundle ###
APP_ENV=dev
APP_SECRET=67d829bf61dc5f87a73fd814e2c9f629
#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/en/latest/reference/configuration.html
# 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?serverVersion=11"
# Configure your db driver and server_version in config/packages/doctrine.yaml
DATABASE_URL=postgresql://postgres:5432/db_demo?user=postgres&password=elfida
###< doctrine/doctrine-bundle ###
```

Рисунок 12. .env

10. Соберем контейнеры с docker-compose, затем зайдём в контейнер `docker_postgres_1` и создадим БД:

```
docker exec -it docker_postgres_1 bash
psql -U postgres
create database db_demo;
\password;
```

11. Во избежания некоторых ошибок необходимо изменить файл, добавив доступ к бд postgres

```
demo/docker/var/lib/postgresql/data/pg_hba.conf
```

(изменять файл следует под root).

host	all	postgres	172.18.0.3/32	md5
------	-----	----------	---------------	-----

Рисунок 13. hba.conf

12. И в этой же директории добавляем в конец следующую строку:

```
listen_addresses='*'
```

13. Зайдём в контейнер `docker_php_1` и заполним нашу БД данными:

```
docker exec -it docker_php_1 bash
```

14. Зайдём на сервер по адресу `demo-symfony.local:8080` и поработаем с бд

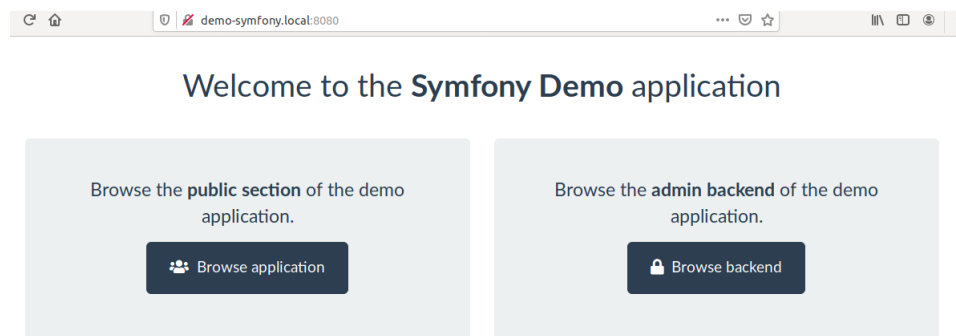


Рисунок 14. Работа с бд

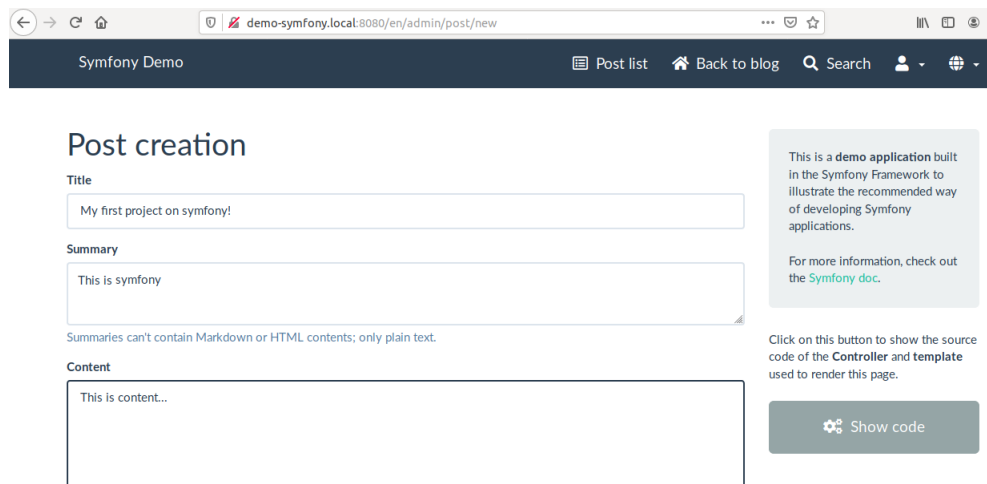


Рисунок 15. Работа с бд - 2

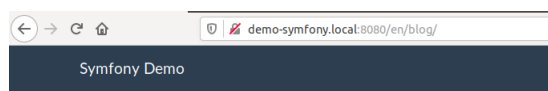


Рисунок 16. Работа с бд - 3

2. 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?
Главное отличие — это среда, в которой работает система. Если запуск осуществляется в виде контейнера, то приложение попадает

в ту же операционную среду, в которой работают остальные приложения заказчика и с которыми происходит взаимодействие. Если же запуск осуществляется в виде виртуальной машины на сервере, то любые обращения в ее сторону запросы со стороны контактирующих приложений проходят длинный путь из одной операционной среды в другую и обратно.

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

`docker build` – сборка образа по настройкам в `Dockerfile`'е

`docker run . . .` – запуск контейнера

`docker start . . .` – запуск контейнера

`docker stop . . .` – остановка контейнера

`docker images` – отобразить образы в локальном репозитории

`docker ps` – отобразить все запущенные контейнеры

`docker ps -a` – отобразить остановленные контейнеры

7. Каким образом осуществляется поиск образов контейнеров?

Изначально `docker` проверяет локальный репозиторий на наличие нужного образа. Если образ не найден, `docker` проверяет удаленный репозиторий.

8. Каким образом осуществляется запуск контейнера?

Docker выполняет инициализацию и запуск ранее созданного по образу контейнера по его имени.

9. Что значит управлять состоянием контейнеров?

Это значит иметь возможность взаимодействовать с контролирующим его процессом — например для корректного завершения своей работы по команде извне. Это позволит аккуратно закрывать транзакции, препятствуя потере пользовательских данных в результате остановки или уничтожения контейнера.

10. Как изолировать контейнер?

Для изоляции контейнера достаточно некоторым образом сконфигурировать `Dockerfile` и `docker-compose.yml`

11. Опишите последовательность создания новых образов, назначение `Dockerfile`?

Для создания нового образа выбирается основа образа (любой подходящий пакет из репозитория Docker Hub), добавляются необходимые слои, выполняются нужные операции и разворачивается рабочее окружение внутри контейнера с необходимыми зависимостями. После чего происходит сборка образа

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

Да, в среде другой виртуализации

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

Назначение Kubernetes состоит в выстраивании эффективной системы распределения контейнеров по узлам кластера в зависимости от текущей нагрузки и имеющихся потребностей при работе сервисов. Kubernetes способен обслуживать сразу большое количество хостов, запускать на них многочисленные контейнеры Docker или Rocket, отслеживать их состояние, контролировать совместную работу и репликацию, проводить масштабирование и балансировку нагрузки.

Основные объекты:

Nodes (node.md): Нода это машина в кластере Kubernetes.

Pods (pods.md): Pod это группа контейнеров с общими разделами, запускаемых как единое целое.

Replication Controllers (replication-controller.md): replication controller гарантирует, что определенное количество «реplik» pod'ы будут запущены в любой момент времени.

Services (services.md): Сервис в Kubernetes это абстракция которая определяет логический объединённый набор pod и политику доступа к ним.

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

Labels (labels.md): 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.md): kubectl интерфейс командной строки для управления Kubernetes.

Заключение

В ходе данной лабораторной работы были изучены или повторно рассмотрены некоторые команды ОС Linux, было проведено ознакомление и анализ рекомендованной литературы, а также информации о Docker в ОС Ubuntu.