Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра Автоматизированных систем управления

Отчет по лабораторной работе № 6

«Контейнеризация»

по курсу «ОС Linux»

Студент Грунау Г.Ю.

Группа АИ-18

Руководитель Кургасов В.В.

Липецк 2020 г.

Цель работы

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

Задачи

Изучить теоретический материал и выполнить предложенные практические задания.

В результате необходимо:

- Знать назначение и возможности Docker;
- Знать особенности установки и настройки Docker;
- Владеть инструментом для определения и запуска многоконтейнерных приложений

Docker – Docker Compose.

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 (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

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.

Ход выполнения

1. Первоначально клонируем себе тестовый проект с помощью команды «git clone https://github.com/symfony/demo»

Рисунок 1 – Клонирование репозитория

2. Перейдем в папку с проектом с помощью команды «cd demo»

   Установим требующиеся расширения для запуска проекта:

   - sudo apt-get install php7.4-cli
   - sudo apt-get install php7.4-mbstring
   - sudo apt-get install php7.4-xml
   - sudo apt-get install php7.4-sqlite

   Далее устанавливаем composer – свободный пакетный менеджер для установки зависимостей и самих модулей PHP.

   - sudo apt install composer

Далее запустим команду composer install, чтобы импортировать пакеты и создать папку поставщика вместе со сценарием автоматической загрузки.

3. Запустим проект с помощью команды php bin/console server:start

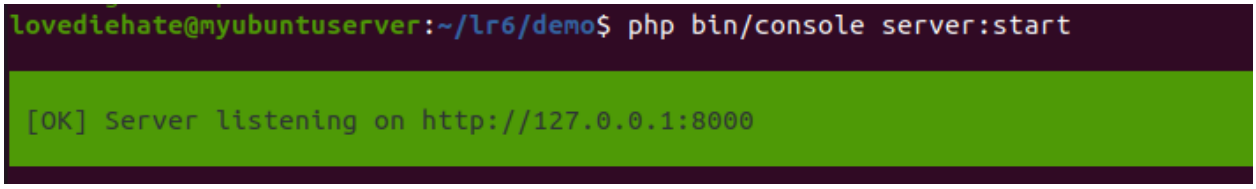

Рисунок 2 – Запуск проекта

127.0.0.1: - адрес локального хоста. По нему можно увидеть результат запуска проекта.

4. В браузере должен стать доступен проект по адресу http://localhost:8000. Главное окно приложения имеет вид:

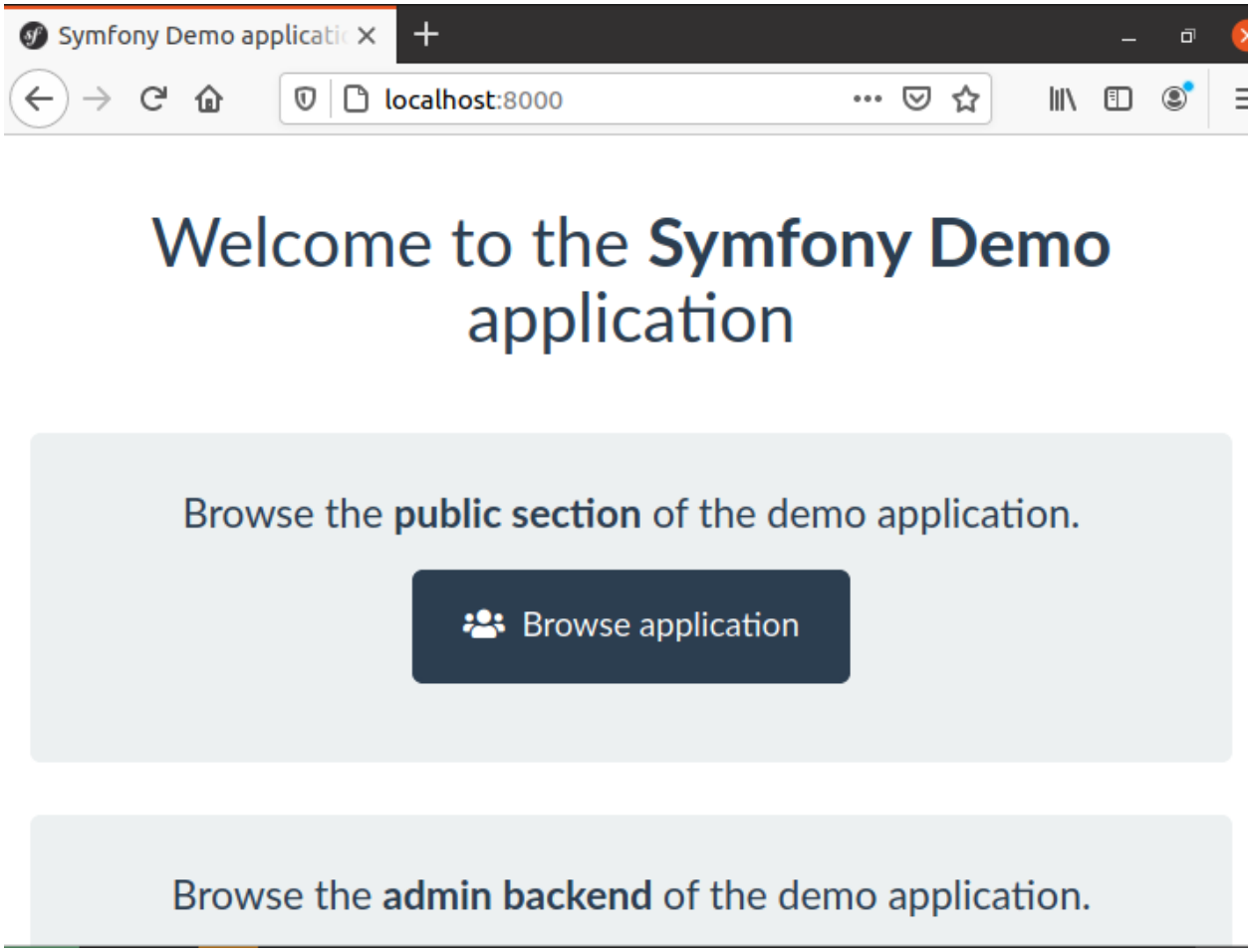

Рисунок 3 – Окно приложения

5. Установим Docker и Docker Compose посредством следующих команд:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
sudo apt update
sudo apt install docker-cesudo 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
```

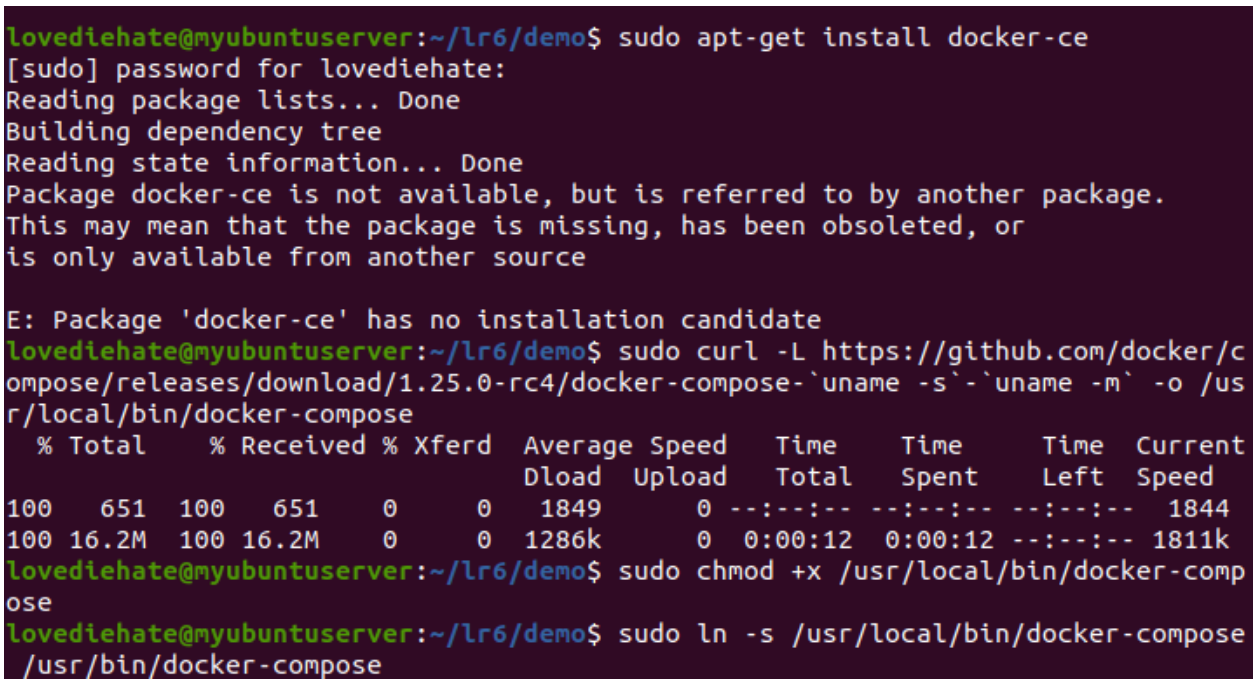

Рисунок 4 – Установка docker compose

В папке с проектом создадим директорию Docker для наших контейнеров и три каталога для каждого отдельного контейнера: nginx, php-fpm, postgres. В каждом каталоге создадим Dockerfile и заполним его следующим содержимым:

nginx/Dockerfile

FROM nginx:latest

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

Также добавим файл конфигурации **nginx/default.conf**:

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;
}
```

php-fpm/Dockerfile

```
FROM php:7.2-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/symphony
```

postgres/Dockerfile

```
FROM postgres:latest
RUN apt-get update
RUN apt-get install -y vim git
```

Создадим на одном уровне с каталогами контейнеров compose-файл **docker-compose.yaml** и заполним:

```
version: '2'
services:
 postgres:
  image: postgres
```

```
 ports:
 - '5435:5432'
 env_file:
 - database.env
 volumes:
 - ./var/lib/postgresql/data:/var/lib/postgresql/data
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:
 - '80:80'
 links:
 - php
 volumes_from:
 - php
 volumes:
 - ../logs/nginx/:/var/log/nginx:cached
```

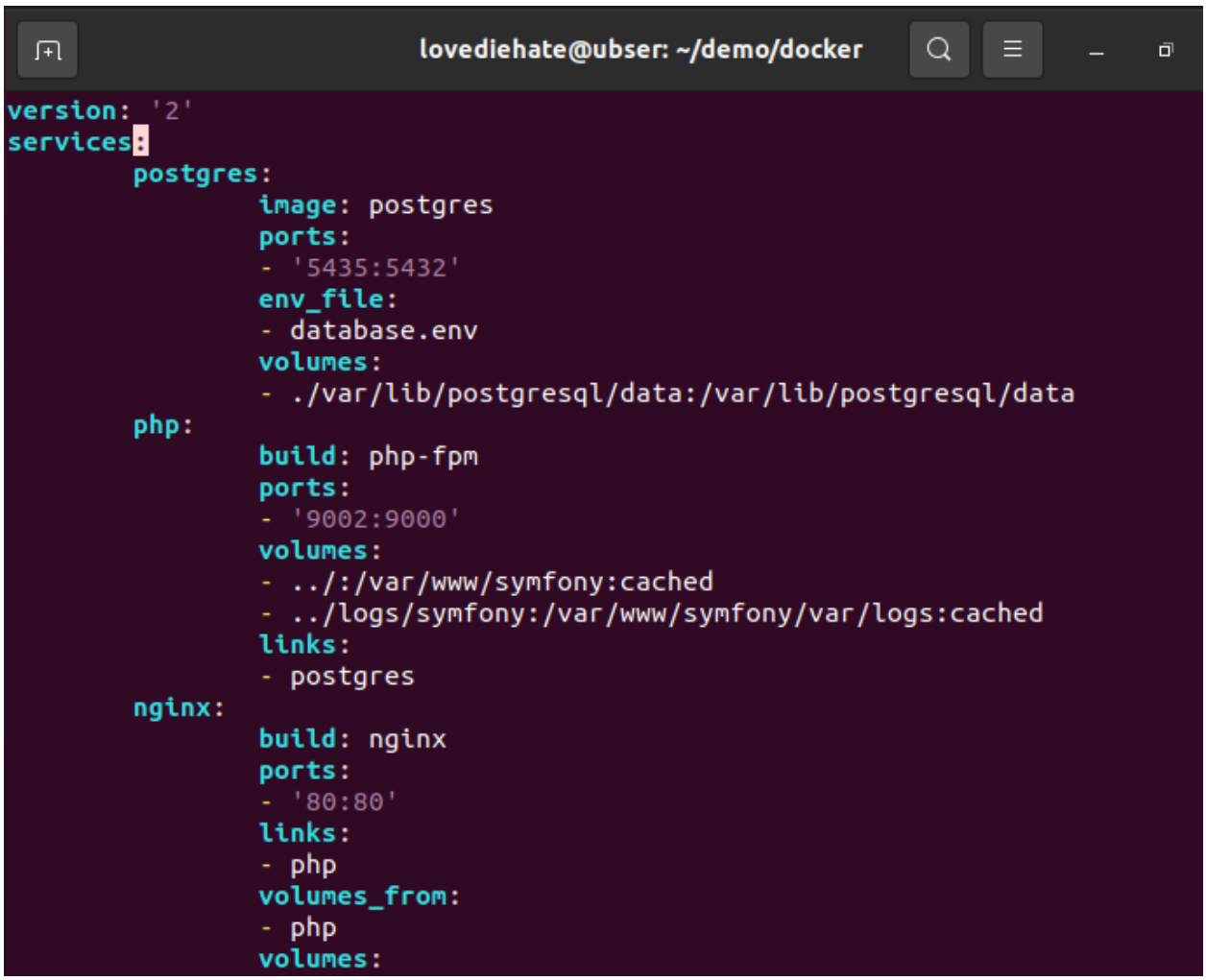

Рисунок 5 – Файл создан

Создадим и заполним БД данными:

Для начала нужно установить postgresql командой:

- sudo apt install postgresql

Чтобы открыть командную строку postgres нужно ввести psql.

Затем, создадим новую роль и её БД:

```
blessed_db=# CREATE USER lovediehate with password 'user12345';
CREATE ROLE
blessed_db=# CREATEDB lovediehatedb;
ERROR:  syntax error at or near "CREATEDB"
LINE 1: CREATEDB lovediehatedb;
        ^
blessed_db=# CREATE database lovediehatedb;
CREATE DATABASE
blessed_db=# grant all privileges on database lovediehatedb to lovediehate;
GRANT
blessed_db=# \q
```

Рисунок 6 – Создание роли и бд

Заполним базу данных данными из фикстур:

- php bin/console doctrine:schema:create
- php bin/console doctrine:fixtures:load

```
lovediehate@ubser:~/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!

lovediehate@ubser:~/demo$ php bin/console doctrine:fixtures:load

 Careful, database "lovediehatedb" will be purged. Do you want to continue? (ye
s/no) [no]:
 > y

   > purging database
   > loading App\DataFixtures\AppFixtures
```

Рисунок 7 – Создание схемы и загрузка фикстур

Затем нужно указать строку подключения к нашей базе данных в файле .env (обратить внимание на название сервера – postgres – имя контейнера, в котором лежит бд):

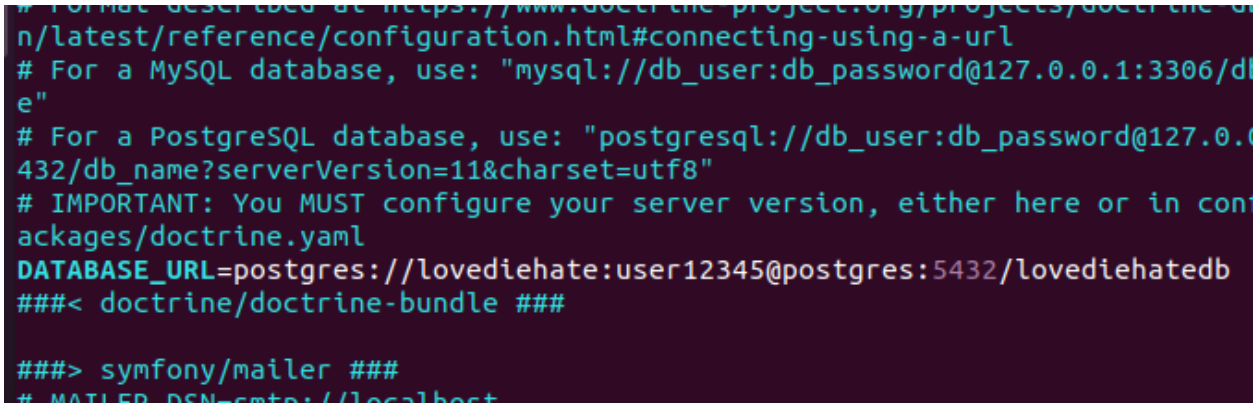

Рисунок 8 – Строка подключения

Далее перейдём в папку docker и запустим наши контейнеры:

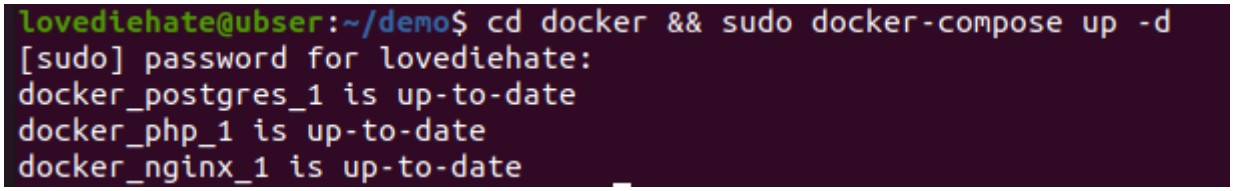

Рисунок 9 - Запуск docker-compose

Теперь контейнеры могут работать связанно, обращаясь друг к другу по имени.

Проверяя работу проекта, можно столкнуться со следующей проблемой:

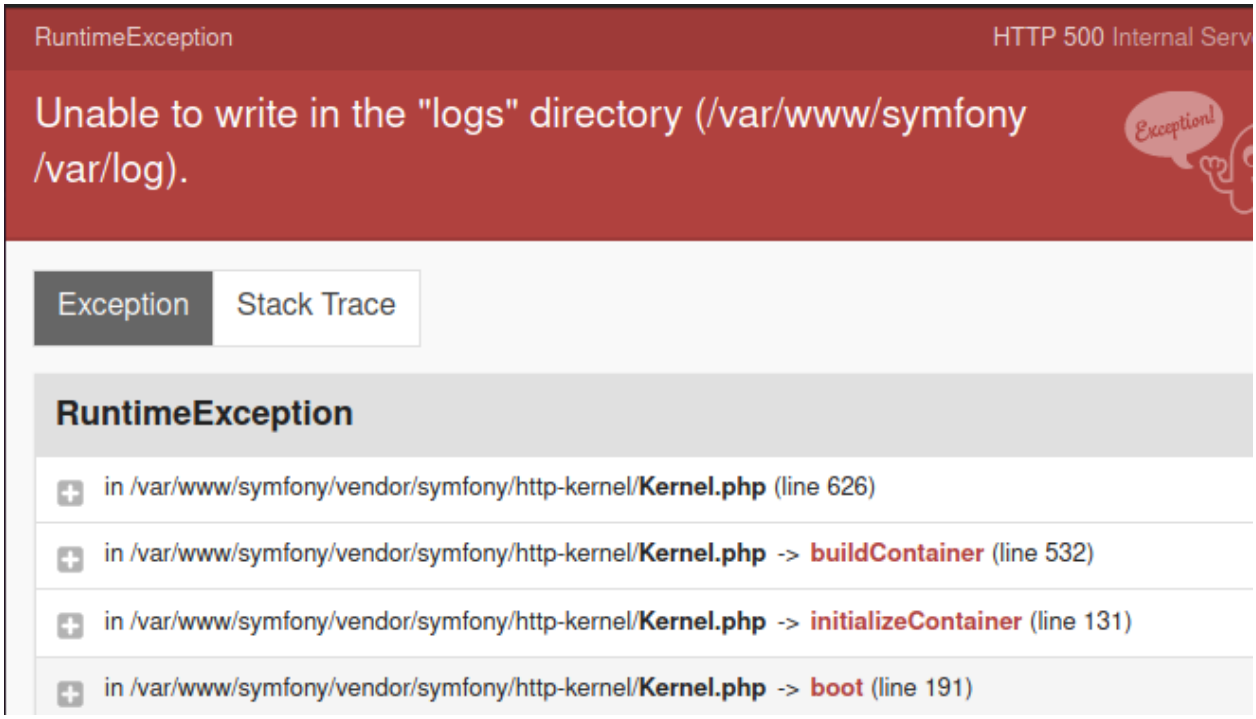

Рисунок 10 – Ошибка при запуске проекта

Чтобы решить её, нужно выполнить команду очистки кэша:

sudo docker-compose exec php php /var/www/symfony/bin/console cache:clear

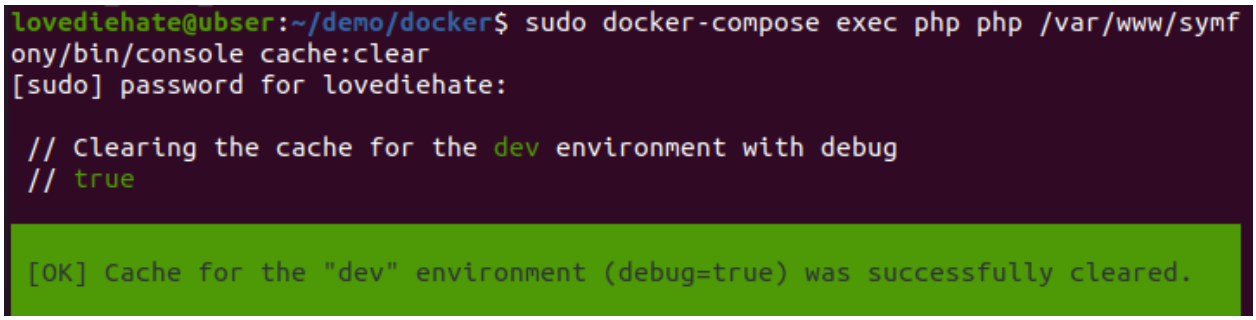

Рисунок 11 – Решение ошибки с логами

Далее выскочит ошибка, связанная с базой данных.

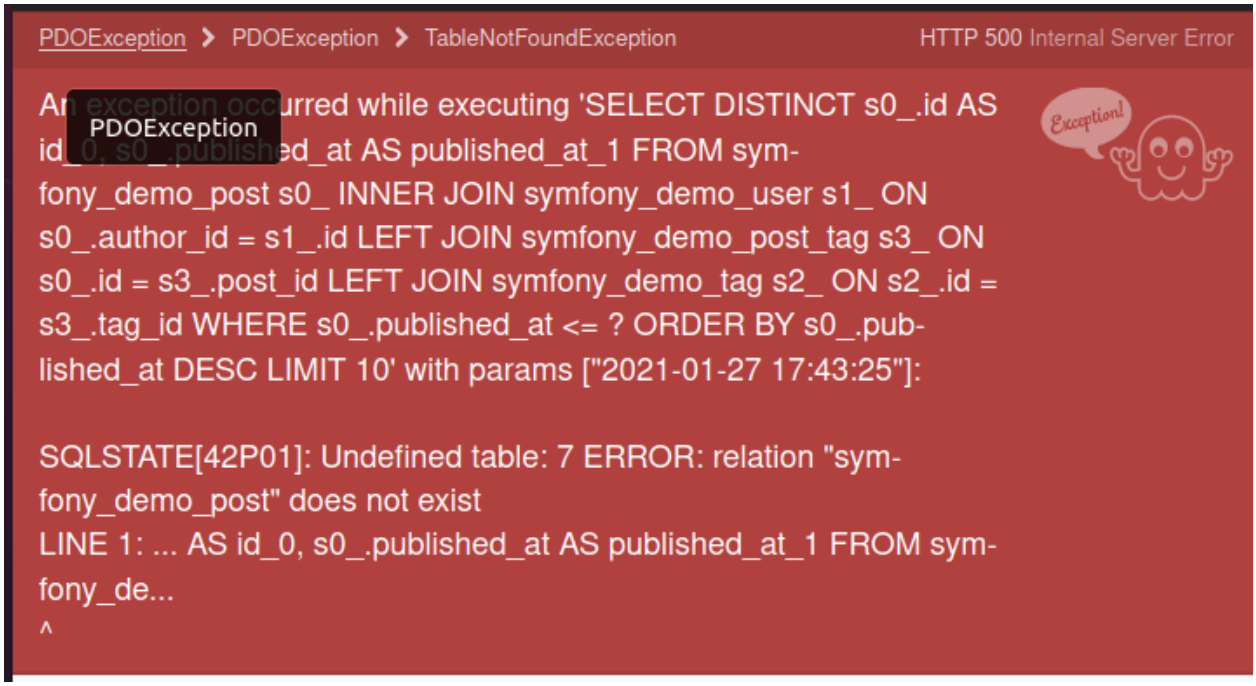

Рисунок 12 – Ошибка SQL

Нужно запустить контейнеры и перейти к контейнеру проекта sudo docker exec -it docker_php_1 bash и выполнить инициализацию таблиц (php bin/console doctrine:schema:create && php bin/console doctrine:fixtures:load)

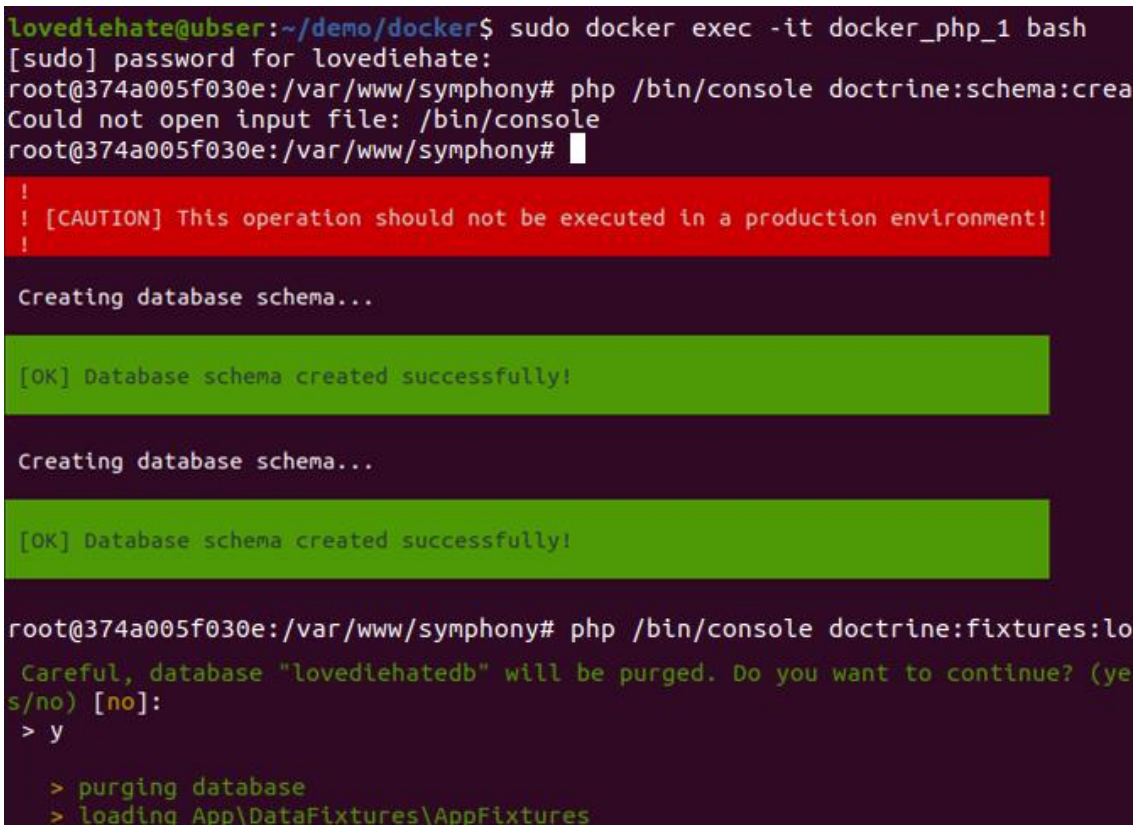

Рисунок 13 – Заполнение таблиц данными

Проделав вышеперечисленные действия, сайт отобразит данные.

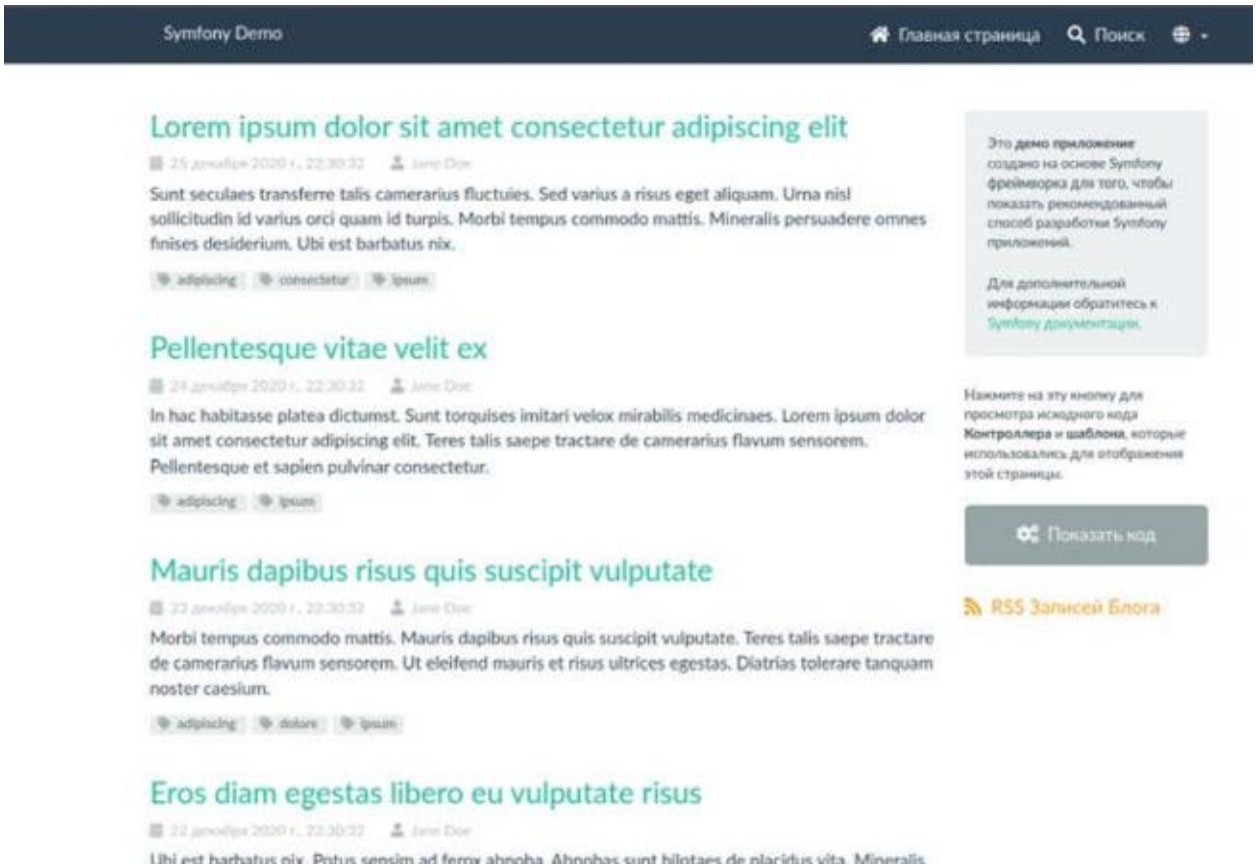

Рисунок 14 - Демонстрация работы проекта

Теперь добавим адрес demo-symfony.local к нашему локальному адресу 127.0.0.1, изменив в папке /etc файл hosts.

```
127.0.0.1 localhost
127.0.1.1 ubser
127.0.0.1 demo-symfony.local
# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Рисунок 15 – Изменение /etc/hosts

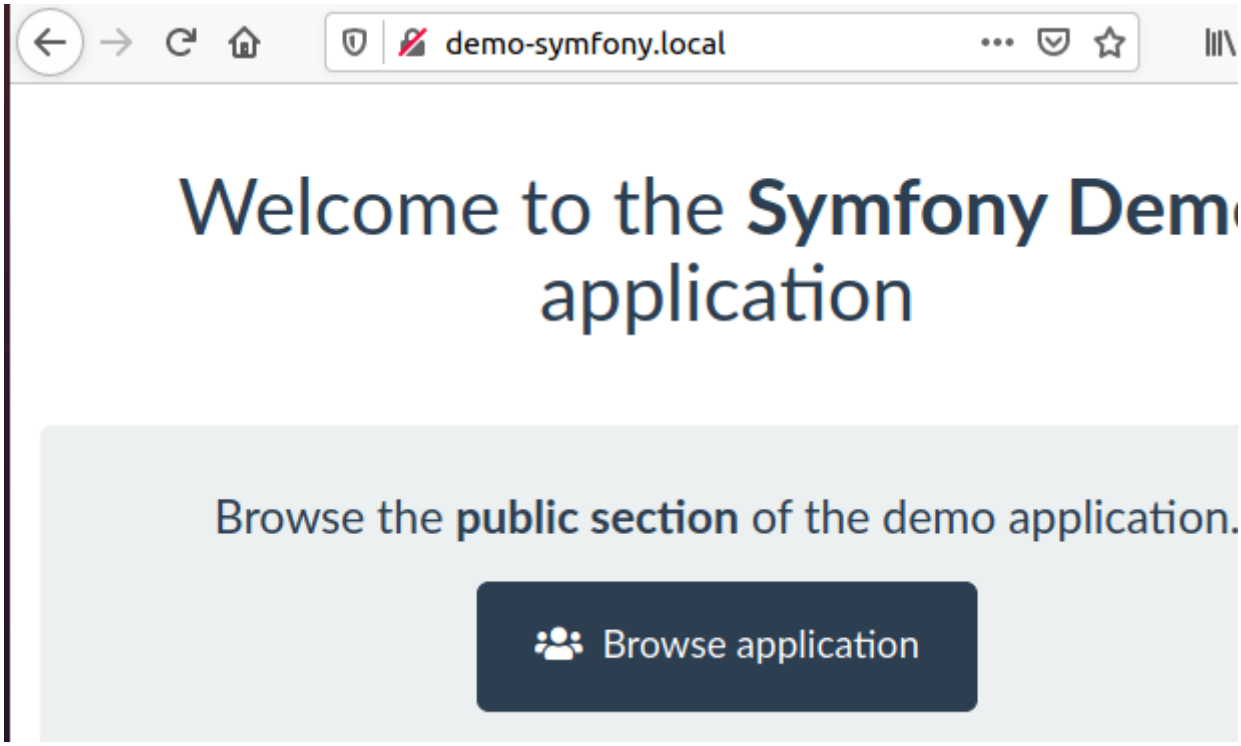

Рисунок 16 – Доступ по demo-symfony.local

Теперь сайт доступен по адресу demo-symfony.local точно так же, как по адресу 127.0.0.1

Вопросы для самопроверки

1.Назовите отличия использования контейнеров по сравнению с виртуализацией.

A. Меньшие накладные расходы на инфраструктуру

2.Назовите основные компоненты Docker.

B. Контейнеры

3.Какие технологии используются для работы с контейнерами?

C. 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?

Виртуальную машину, запускаемую на хосте, также часто называют «гостевой машиной». Гостевая машина содержит как приложение, так и все, что нужно для его запуска (например, системные исполняемые файлы и библиотеки). Она также несет в себе весь аппаратный стек, включая виртуальные сетевые адаптеры, файловое хранилище и центральный процессор, и свою собственную полноценную гостевую операционную систему.

В отличие от виртуальной машины, обеспечивающей аппаратную виртуализацию, контейнер обеспечивает виртуализацию на уровне операционной системы с помощью абстрагирования «пользовательского пространства».

15

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

docker build – сборка образа по настройкам в Dockerfile’е

docker run … – 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

docker exec –it… - выполнить команду в определенном контейнере

7.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.

8.Каким образом осуществляется запуск контейнера?

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

9.Что значит управлять состоянием контейнеров?

Это означает контролировать ход выполнения контейнера и в любой момент времени переводить его в остановленный/запущенный режим и выполнять команды внутри контейнера.

10.Как изолировать контейнер?

Для изоляции контейнера достаточно правильно сконфигурировать файлы Dockerfile и docker-compose.yaml (если есть). По умолчанию

16

контейнеры запускаются от root прав, поэтому стоит быть осторожным с монтированием томов на хост машину.

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 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.

Основные объекты:

Узел — это отдельная физическая или виртуальная машина, на которой развёрнуты и выполняются контейнеры приложений. Каждый узел в кластере содержит сервисы для запуска приложений в контейнерах (например, Docker),

а также компоненты, предназначенные для централизованного управления узлом.

Под — базовая единица для управления и запуска приложений, один или несколько контейнеров, которым гарантирован запуск на одном узле,

17

обеспечивается разделение ресурсов, межпроцессное взаимодействие и предоставляется уникальный в пределах кластера IP-адрес.

Том — общий ресурс хранения для совместного использования из контейнеров, развёрнутых в пределах одного пода.

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.

Сервисом в Kubernetes называют совокупность логически связанных наборов подов и политик доступа к ним.

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.

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.