Липецкий государственный технический университет

Факультет автоматизации и информатики
Кафедра автоматизированных систем управления

Отчет по лабораторной работе № 6
по дисциплине «OS Linux»
на тему «Контейнеризация»

Студент _____ Сухоруков К.О.
подпись, дата фамилия, инициалы

Группа АС-18-1

Руководитель
к.н. _____ Кургасов В.В.
учёная степень, учёное звание подпись, дата фамилия, инициалы

Липецк 2020 г.

СОДЕРЖАНИЕ

Цель работы

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

1 Ход работы

1.1 Часть 1

Для начала перейдем на сайт https://docs.docker.com и установим docker, docker-compose с помощью команд указанных в документации. После этого устанавливаем composer – менеджер пакетов для php.

Клонируем проект Symfony Demo Application из репозитория расположенного по адресу: https://github.com/symfony/demo:

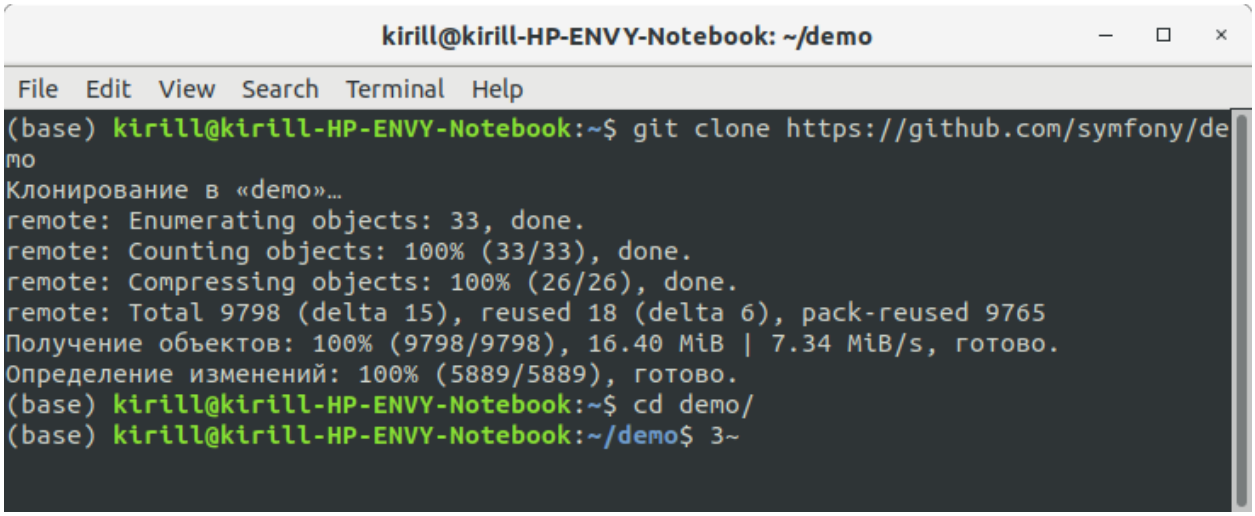

Рисунок 1.1 – Клонирование проекта

Переходим в папку с проектом и с помощью команды composer install устанавливаем все необходимые пакеты для данного проекта.

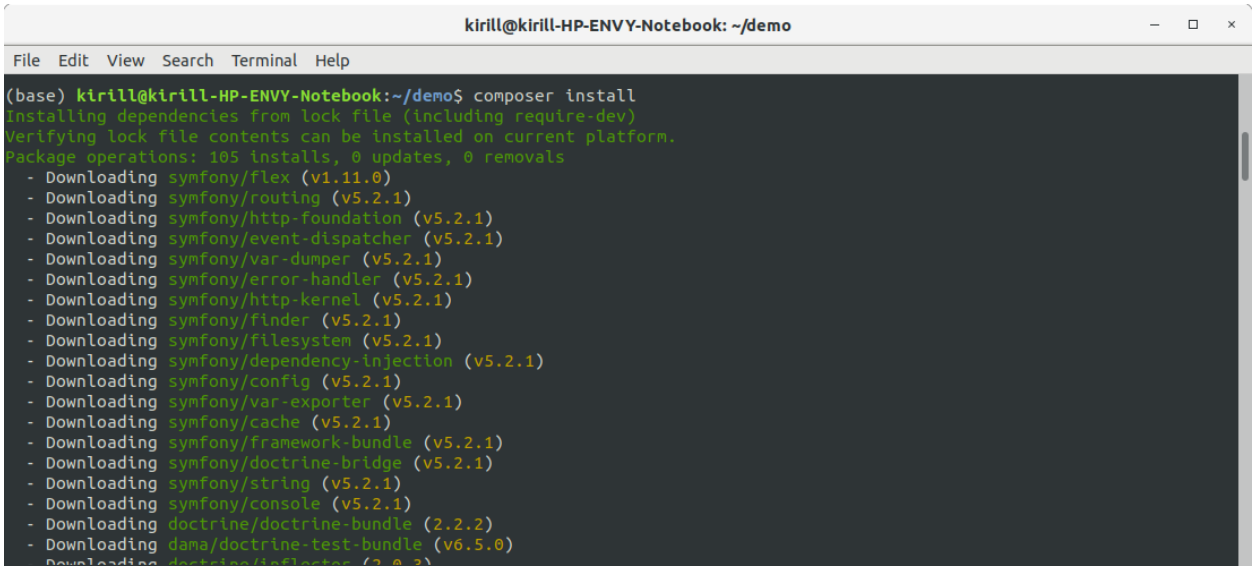

Рисунок 1.2 – Установка необходимых пакетов

Теперь можно попробовать запустить проект с помощью команды symfony serve.

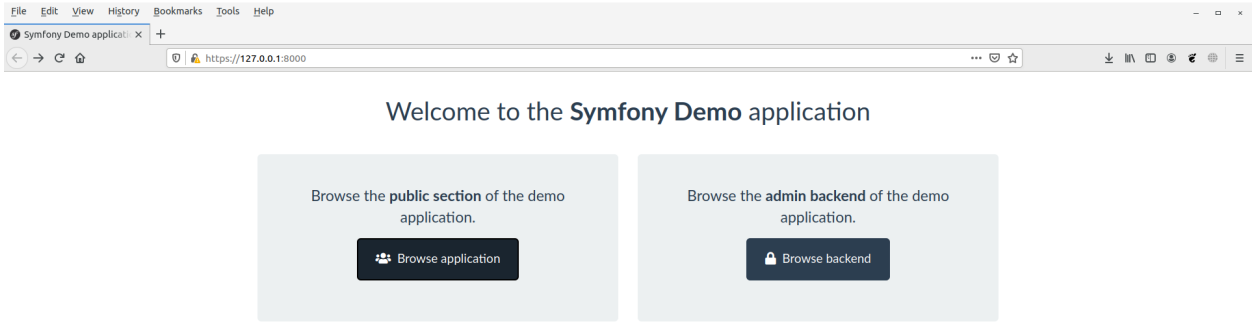

Рисунок 1.3 – Запуск демо проекта

Следующим шагом установим postgresql, создадим базу данных demo_db и подключим ее к нашему проекту.

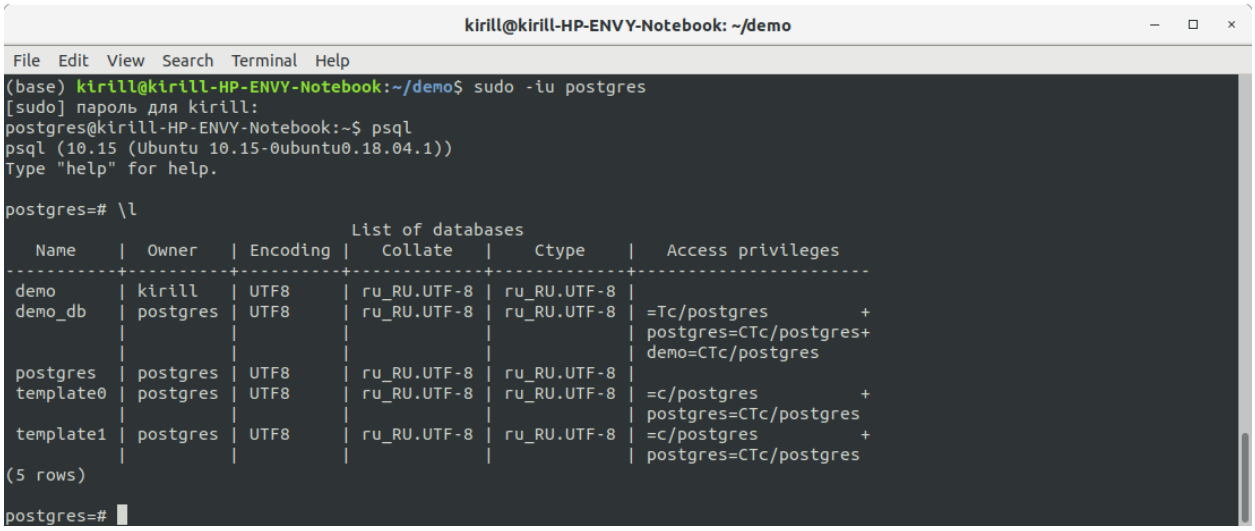

Рисунок 1.4 – Создание базы данных demo_db

Отредактируем файл окружения .env, чтобы подключить новую базу данных к нашему проекту.

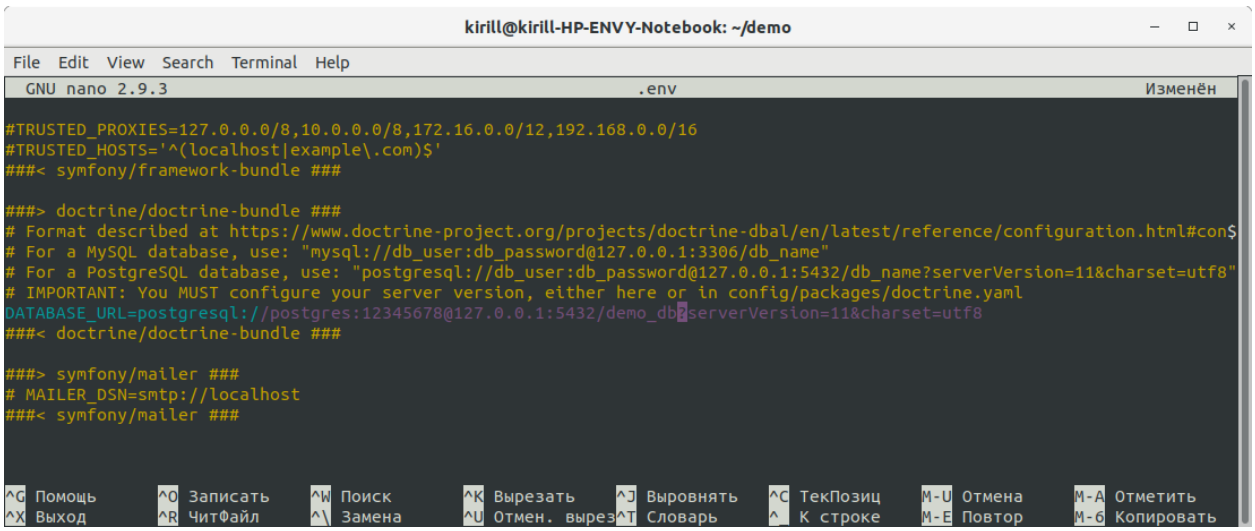

Рисунок 1.5 – Поключение базы данных в файле .env

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, запускаем его с помощью команды symfony serve.

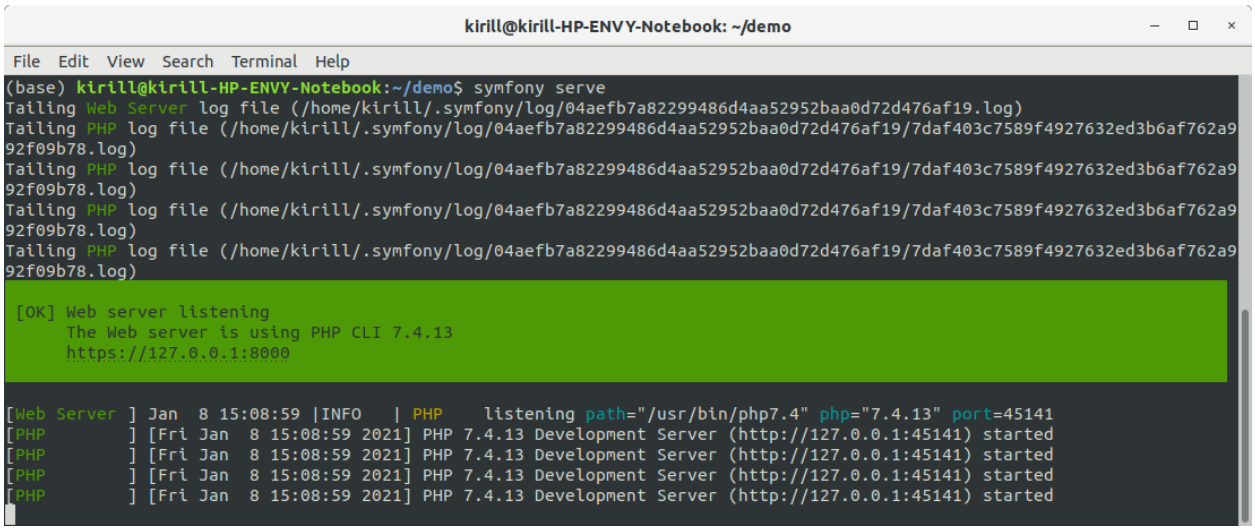

Рисунок 1.6 – Запуск проекта

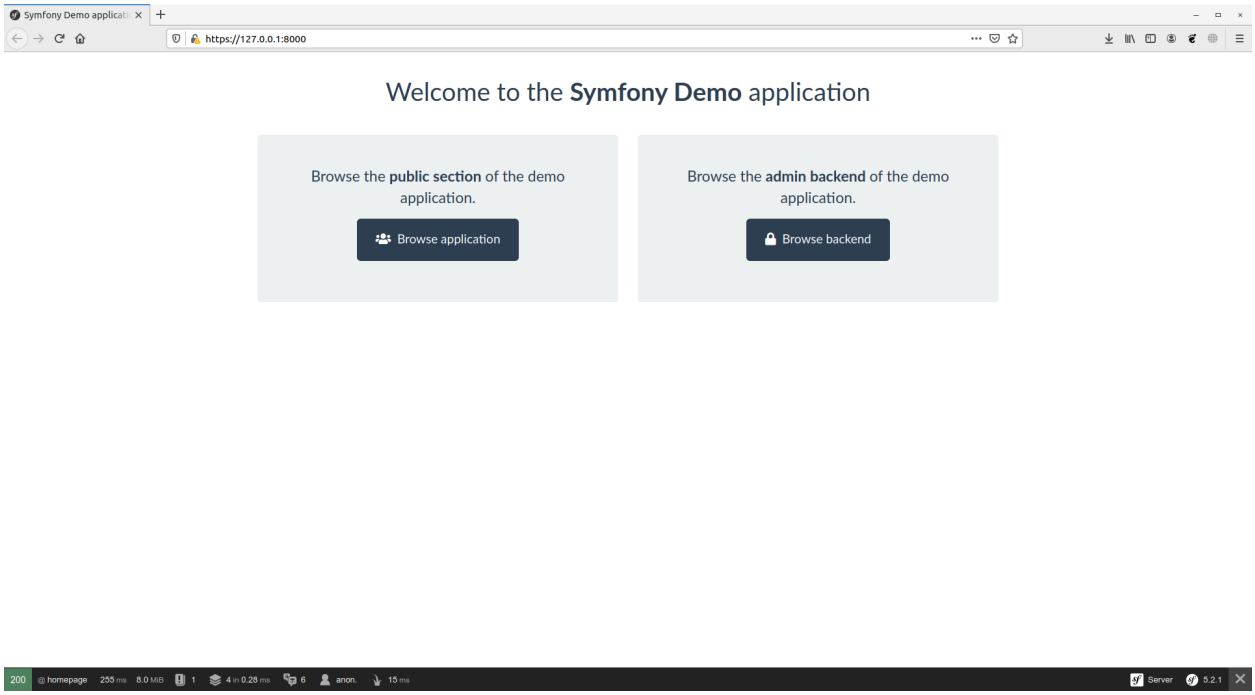

Рисунок 1.7 – Стартовая страница проекта

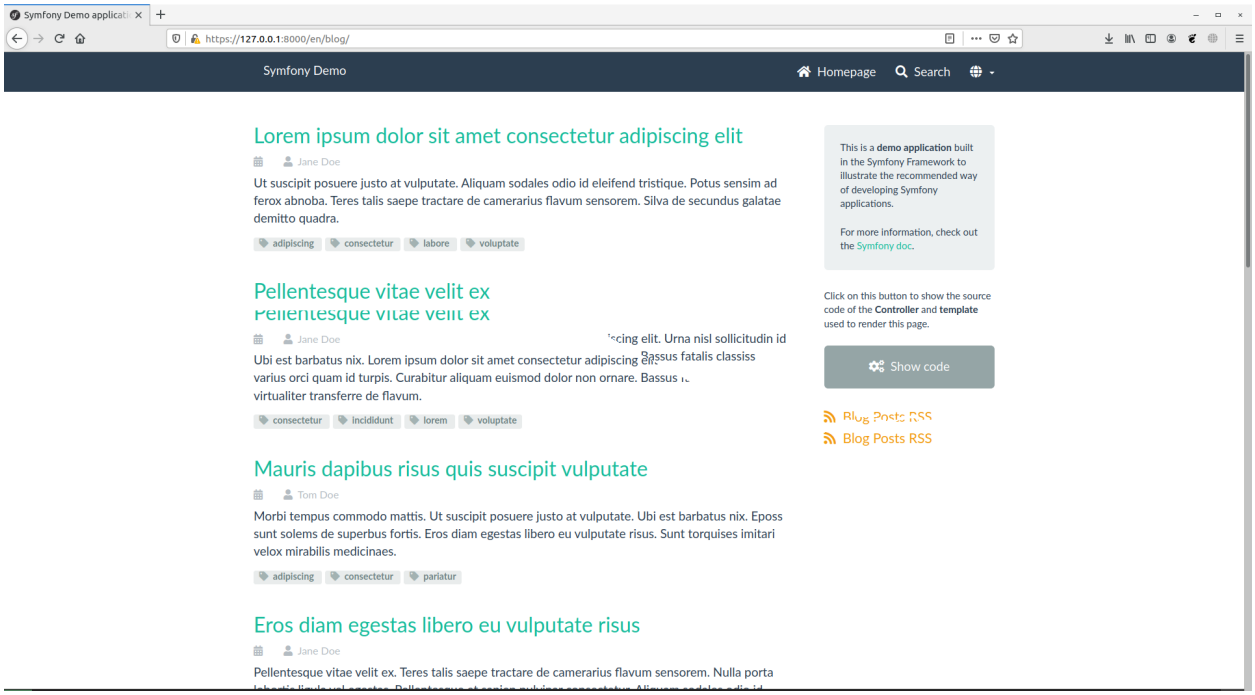

Рисунок 1.8 – Проверка работоспособности базы данных

Перейдем к настройке контейнеров. Первым делом создадим файл docker-compose.yml и заполним его следующим содержимым:

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

    nginx:
        container_name: nginx
        build:
            context: .
            dockerfile: docker/nginx.Dockerfile
        ports:
            - "8084:80"
        volumes:
            - ./:/var/www/symfony
            - ./logs/nginx/:/var/log/nginx
        links:
            - php-fmp

    postgres:
        container_name: postgres
        image: postgres
        environment:
            POSTGRES_DB: demo_db
            POSTGRES_USER: demo_usr
            POSTGRES_PASSWORD: demo_pwd
        volumes:
            - ./data/postgresql/demo_db:/var/lib/postgresql/data
        ports:
            - 5432:5432
```

В данном файле мы определяем структуру и связи наших контейнеров, перенаправляем некоторые порты, а так же связываем файлы и каталоги на локальном компьютере с контейнерами. В частности с помощью данного механизма исключается возможность потери базы данных при остановке контейнеров.

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
```

После этого также редактируем файл .env, заменяем сроку подключения к БД следующей строкой:

DATABASE_URL=postgresql://demo_usr:demo_pwd@postgres:5432/demo_db?serverVersion=11&charset=utf8

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 demo_bd (команда create database demo_db;). Дальше 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.

Запускаем контейнеры командой docker-compose up -d и проверяем работу проекта.

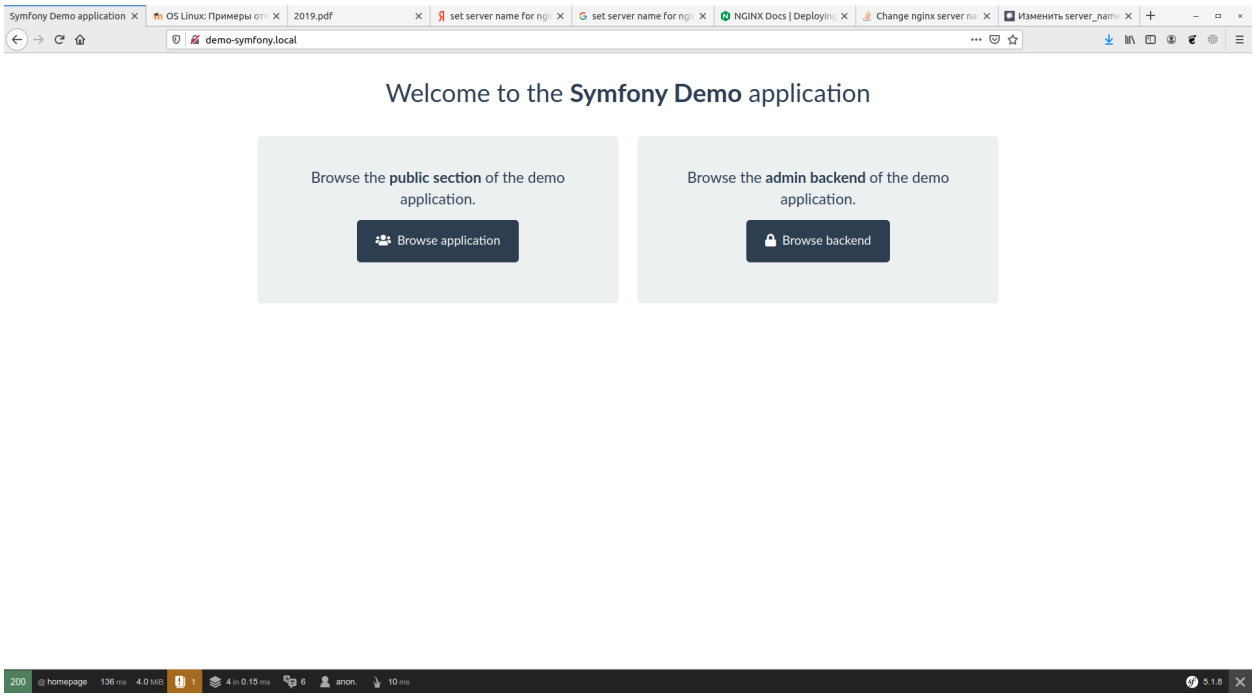

Рисунок 1.9 – Запуск проекта в конейнерах

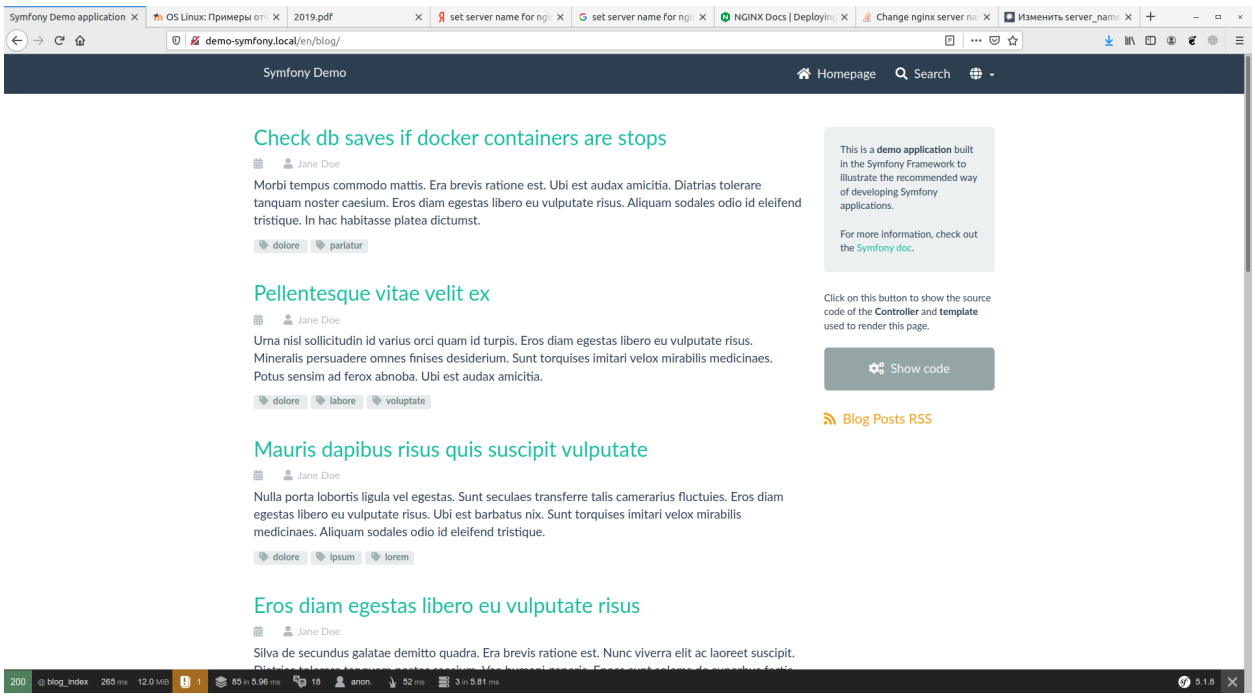

Рисунок 1.10 – Проверка, что БД не удаляется полсле перезагрузки проекта

1.2 Часть 2

Теперь удаляем конфигурацию контейнера с postgresql и подключим проект к локальной базе данных. Для этого узнаем ip локальной машины с помощью команды hostname -I | cut -d ' ' -f1 и добавим этот ip под псевдонимом bd в наш файл docker-compose.yml, получим следующее содержимое:

```
version: "3"
services:
    php-fmp:
        container_name: php-fpm
        build:
            context: .
            dockerfile: docker/php.Dockerfile
        volumes:
            - ./:/var/www/symfony
            - ./logs/symfony:/var/www/symfony/log
        extra_hosts:
            - "db:192.168.0.103"

    nginx:
        container_name: nginx
        build:
            context: .
            dockerfile: docker/nginx.Dockerfile
        ports:
            - "80:80"
        volumes:
            - ./:/var/www/symfony
            - ./logs/nginx/:/var/log/nginx
        links:
            - php-fmp
```

Также изменим этот адрес в файле .env.

Теперь изменим конфигурацию локальной базы данных, так, чтобы она допускала подключение из контейнера. Для этого изменим файлы конфигурации /etc/postgresql/10/main/postgresql.conf и pg_hba.conf:

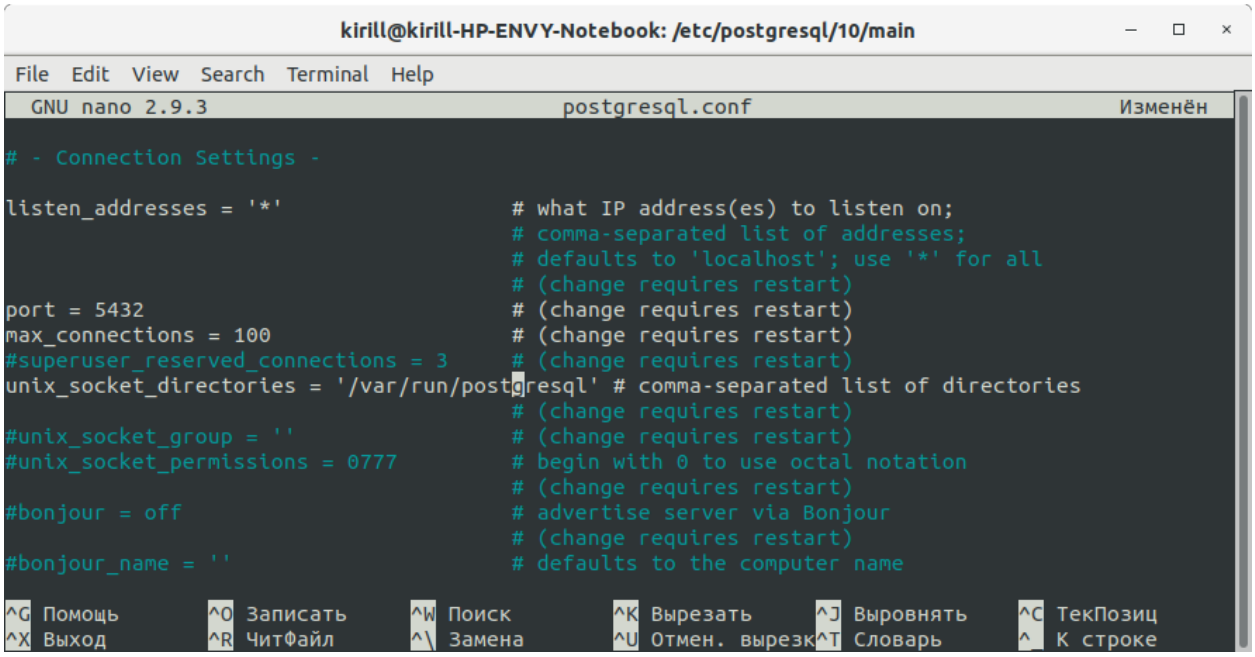

Рисунок 1.11 – Изменение файла postgresql.conf

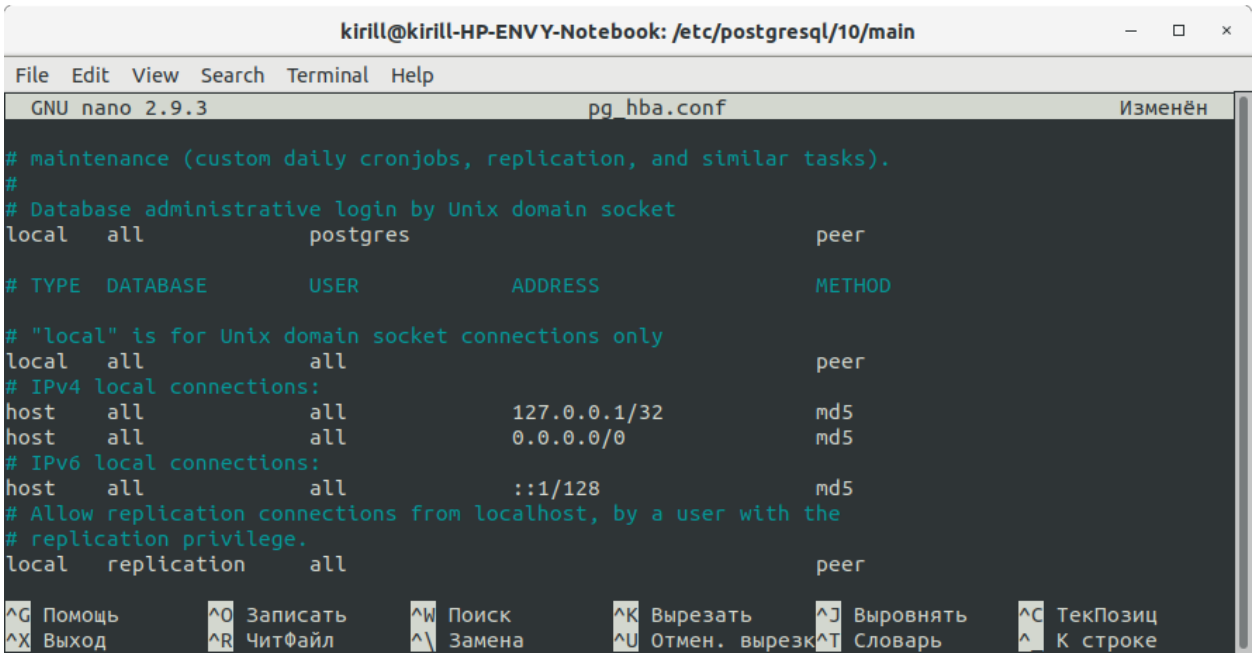

Рисунок 1.12 – Изменение файла pg_hba.cong

Перезапустим postgresql с помощью команды service postgres restart и попробуем запустить наш проект.

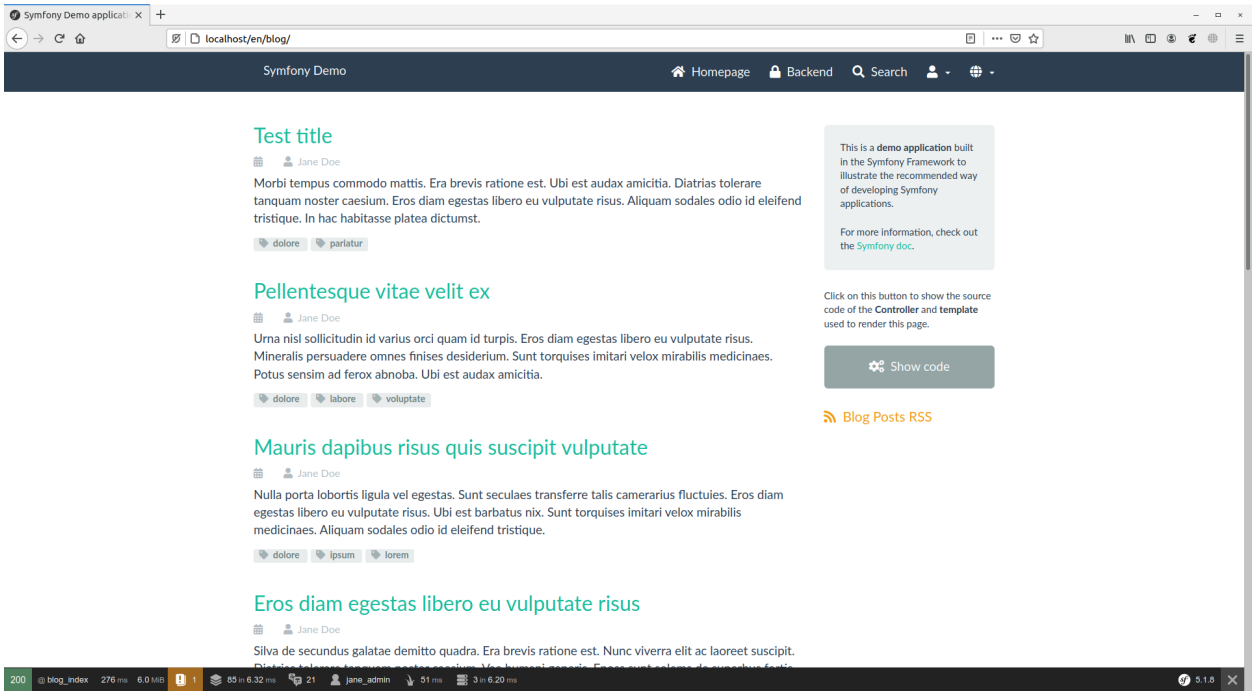

Рисунок 1.13 – Подключение к локальной БД

Контрольные вопросы

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?

Виртуальная машина – программная и/или аппаратная система, эмулирующая аппаратное обеспечение некоторой целевой и исполняющая программы для гостевой платформы на платформе-хозяине (хосте) или виртуализирующая некоторую платформу и создающая на ней среды, изолирующие друг от друга программы и даже операционные системы. Виртуальные машины запускают на физических машинах, используя гипервизор.

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, если использовать 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.