

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

Группа АС-18-1

Руководитель

К.Н.

учёная степень, учёное звание

подпись, дата

подпись, дата

Сухоруков К.О.

фамилия, инициалы

Кургасов В.В.

фамилия, инициалы

Липецк 2020 г.

СОДЕРЖАНИЕ

Цель работы	2
1 Ход работы	3
1.1 Часть 1	3
1.2 Часть 2	11
Контрольные вопросы	14

Цель работы

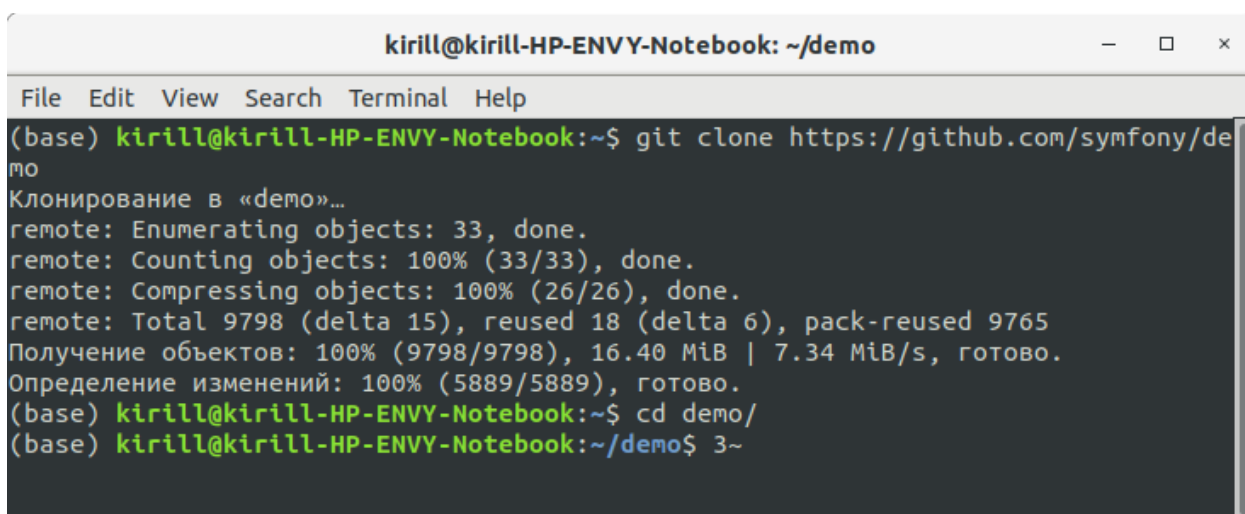
Изучить современные методы разработки ПО в динамических и распределенных средах на примере контейнеров Docker.

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.

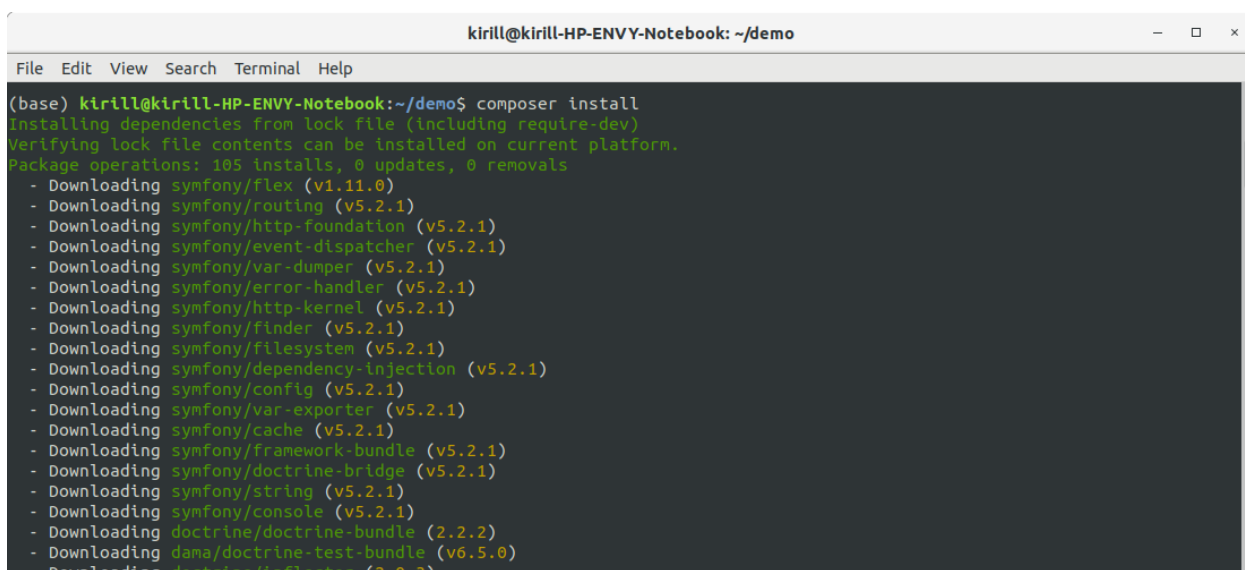
Клонируем проект Symfony Demo Application из репозитория расположенного по адресу: <https://github.com/symfony/demo>:



```
kirill@kirill-HP-ENVY-Notebook: ~/demo
File Edit View Search Terminal Help
(base) kirill@kirill-HP-ENVY-Notebook:~$ git clone https://github.com/symfony/demo
Клонирование в «demo»...
remote: Enumerating objects: 33, done.
remote: Counting objects: 100% (33/33), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 9798 (delta 15), reused 18 (delta 6), pack-reused 9765
Получение объектов: 100% (9798/9798), 16.40 MiB | 7.34 MiB/s, готово.
Определение изменений: 100% (5889/5889), готово.
(base) kirill@kirill-HP-ENVY-Notebook:~$ cd demo/
(base) kirill@kirill-HP-ENVY-Notebook:~/demo$ 3~
```

Рисунок 1.1 – Клонирование проекта

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



```
kirill@kirill-HP-ENVY-Notebook: ~/demo
File Edit View Search Terminal Help
(base) kirill@kirill-HP-ENVY-Notebook:~/demo$ composer install
Installing dependencies from lock file (including require-dev)
Verifying lock file contents can be installed on current platform.
Package operations: 105 installs, 0 updates, 0 removals
- Downloading symfony/flex (v1.11.0)
- 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/var-exporter (v5.2.1)
- Downloading symfony/cache (v5.2.1)
- Downloading symfony/framework-bundle (v5.2.1)
- Downloading symfony/doctrine-bridge (v5.2.1)
- Downloading symfony/string (v5.2.1)
- Downloading symfony/console (v5.2.1)
- Downloading doctrine/doctrine-bundle (2.2.2)
- Downloading dama/doctrine-test-bundle (v6.5.0)
- Downloading doctrine/doctrine-bundle (2.2.2)
```

Рисунок 1.2 – Установка необходимых пакетов

Теперь можно попробовать запустить проект с помощью команды `symfony serve`.

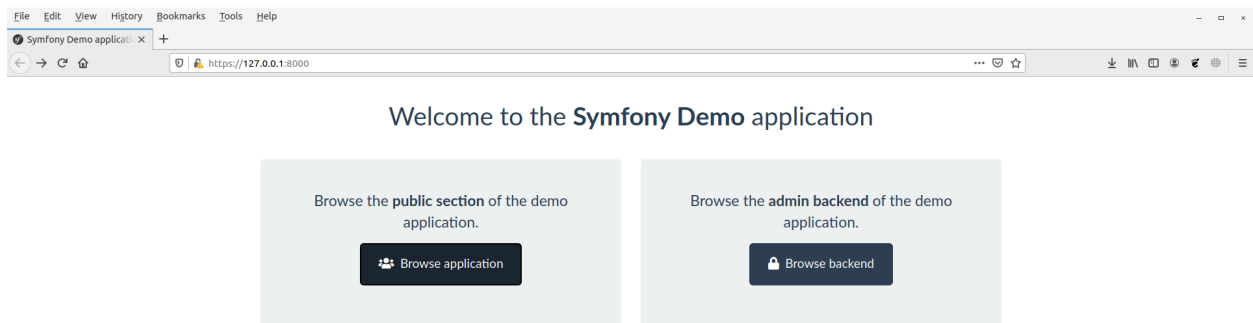


Рисунок 1.3 – Запуск демо проекта

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

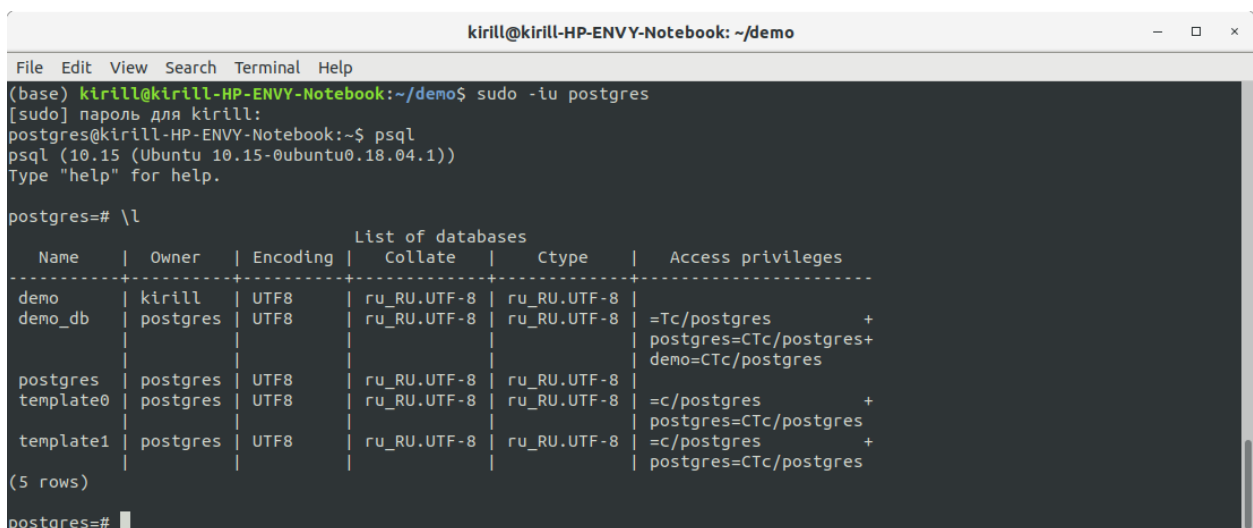
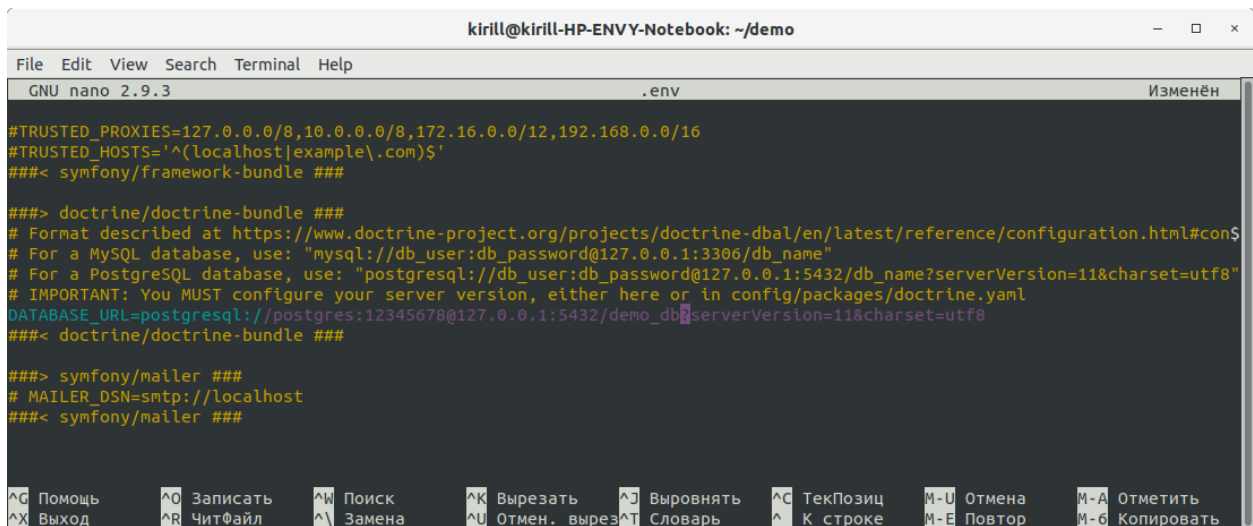


Рисунок 1.4 – Создание базы данных `demo_db`

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



```
kirill@kirill-HP-ENVY-Notebook: ~/demo
File Edit View Search Terminal Help
GNU nano 2.9.3 .env Изменён

#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#conS
# 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&charset=utf8"
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
DATABASE_URL=postgresql://postgres:12345678@127.0.0.1:5432/demo_db?serverVersion=11&charset=utf8
###< doctrine/doctrine-bundle ###

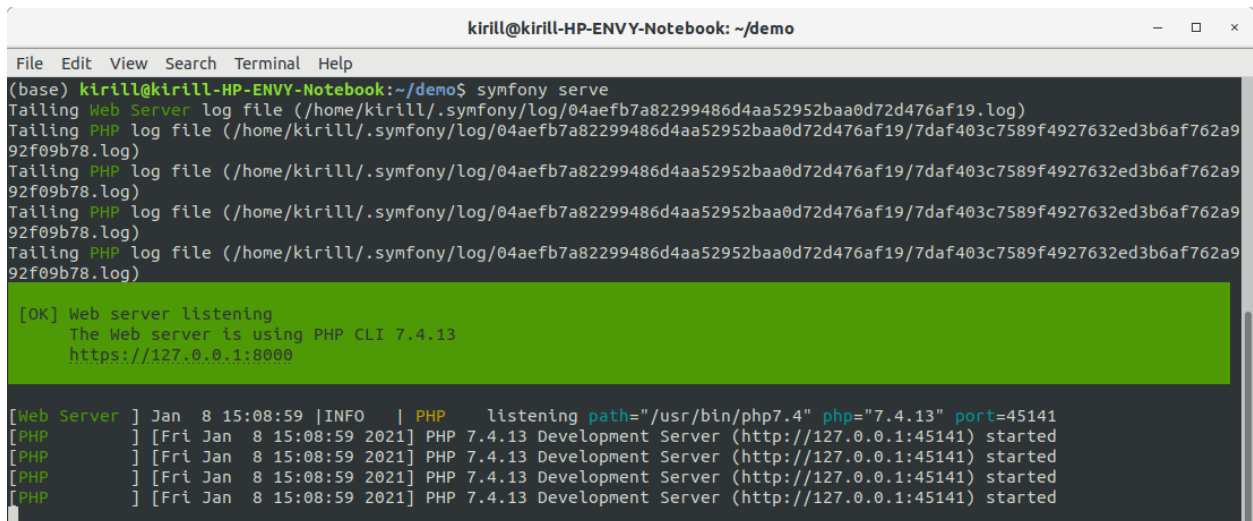
###> symfony/mailer ###
# MAILER_DSN=smtp://localhost
###< symfony/mailer ###

^C Помощь ^O Записать ^W Поиск ^K Вырезать ^J Выводить ^C ТекПозиц M-U Отмена M-A Отметить
^X Выход ^R ЧитФайл ^_ Замена ^U Отмен. вырезAT Словарь ^_ К строке M-E Повтор M-G Копировать
```

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`.



```
kirill@kirill-HP-ENVY-Notebook: ~/demo
File Edit View Search Terminal Help
(base) kirill@kirill-HP-ENVY-Notebook:~/demo$ symfony serve
Tailing Web Server log file (/home/kirill/.symfony/log/04aefb7a82299486d4aa52952baa0d72d476af19.log)
Tailing PHP log file (/home/kirill/.symfony/log/04aefb7a82299486d4aa52952baa0d72d476af19/7daf403c7589f4927632ed3b6af762a992f09b78.log)
Tailing PHP log file (/home/kirill/.symfony/log/04aefb7a82299486d4aa52952baa0d72d476af19/7daf403c7589f4927632ed3b6af762a992f09b78.log)
Tailing PHP log file (/home/kirill/.symfony/log/04aefb7a82299486d4aa52952baa0d72d476af19/7daf403c7589f4927632ed3b6af762a992f09b78.log)
Tailing PHP log file (/home/kirill/.symfony/log/04aefb7a82299486d4aa52952baa0d72d476af19/7daf403c7589f4927632ed3b6af762a992f09b78.log)

[OK] Web server listening
The Web server is using PHP CLI 7.4.13
https://127.0.0.1:8000

[Web Server] Jan  8 15:08:59 [INFO] | PHP listening path="/usr/bin/php7.4" php="7.4.13" port=45141
[PHP] [Fri Jan  8 15:08:59 2021] PHP 7.4.13 Development Server (http://127.0.0.1:45141) started
[PHP] [Fri Jan  8 15:08:59 2021] PHP 7.4.13 Development Server (http://127.0.0.1:45141) started
[PHP] [Fri Jan  8 15:08:59 2021] PHP 7.4.13 Development Server (http://127.0.0.1:45141) started
[PHP] [Fri Jan  8 15:08:59 2021] PHP 7.4.13 Development Server (http://127.0.0.1:45141) started
```

Рисунок 1.6 – Запуск проекта

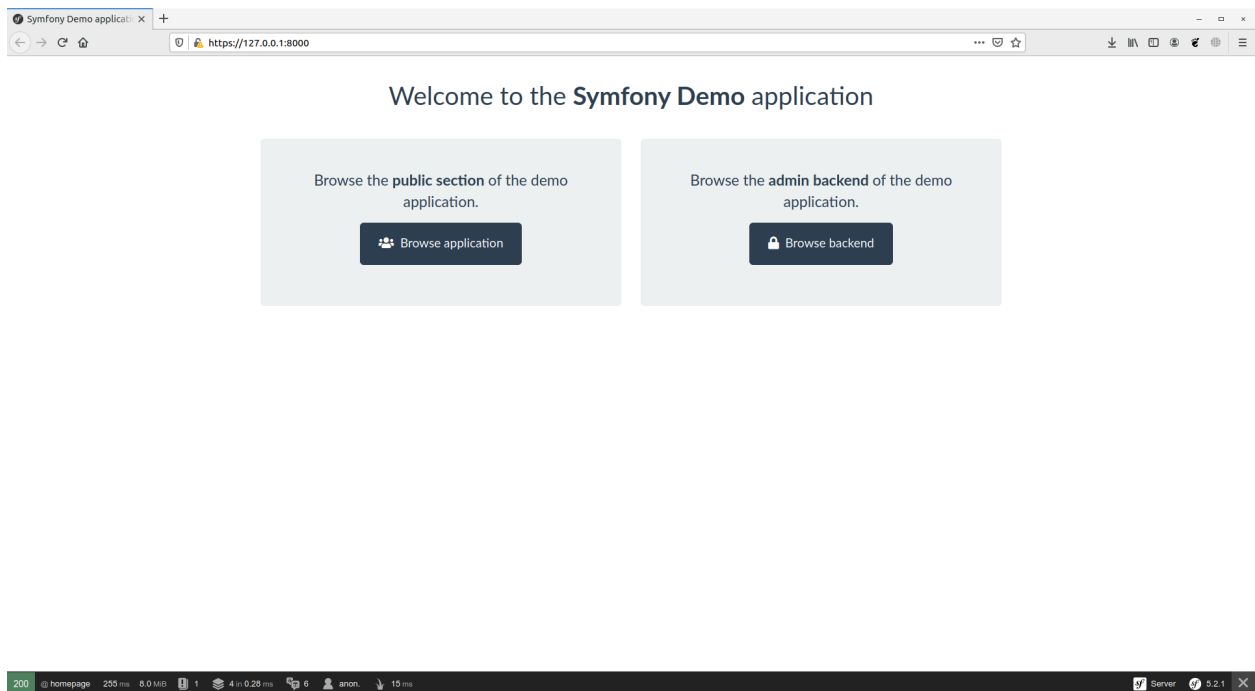


Рисунок 1.7 – Стартовая страница проекта

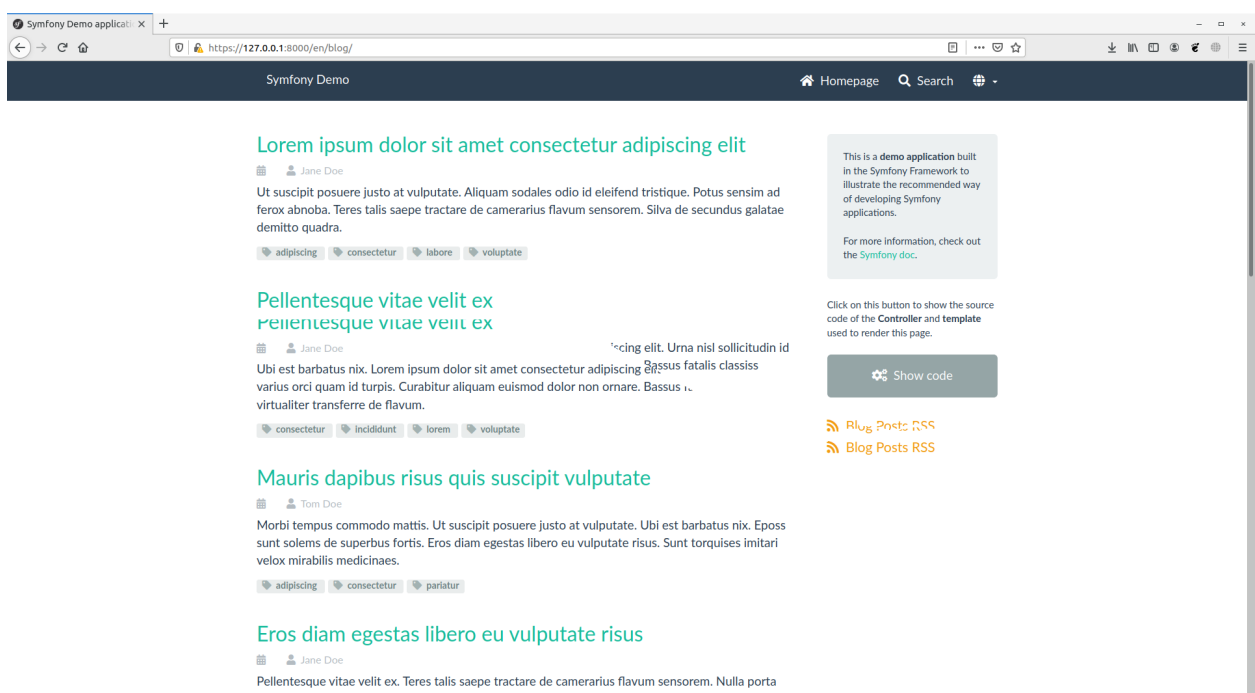


Рисунок 1.8 – Проверка работоспособности базы данных

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

```
version: "3"
services:
  php-fpm:
    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-fpm

  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 создадим 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 строку 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;`). Далее 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.

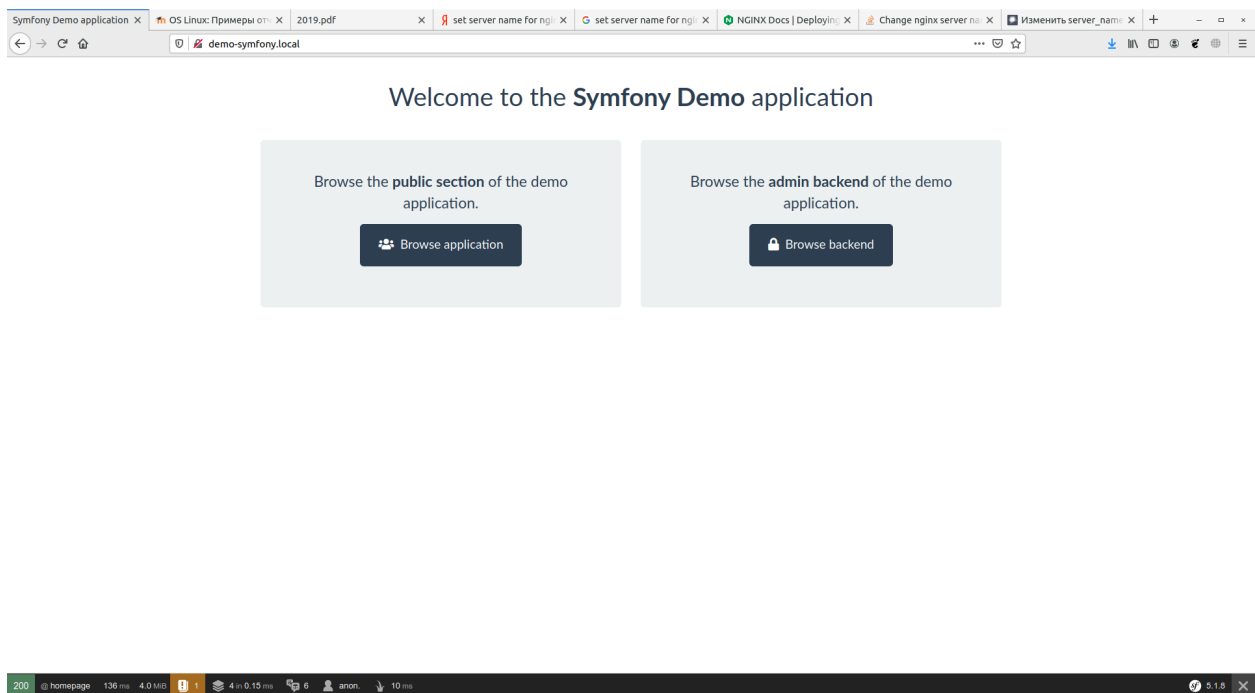


Рисунок 1.9 – Запуск проекта в контейнерах

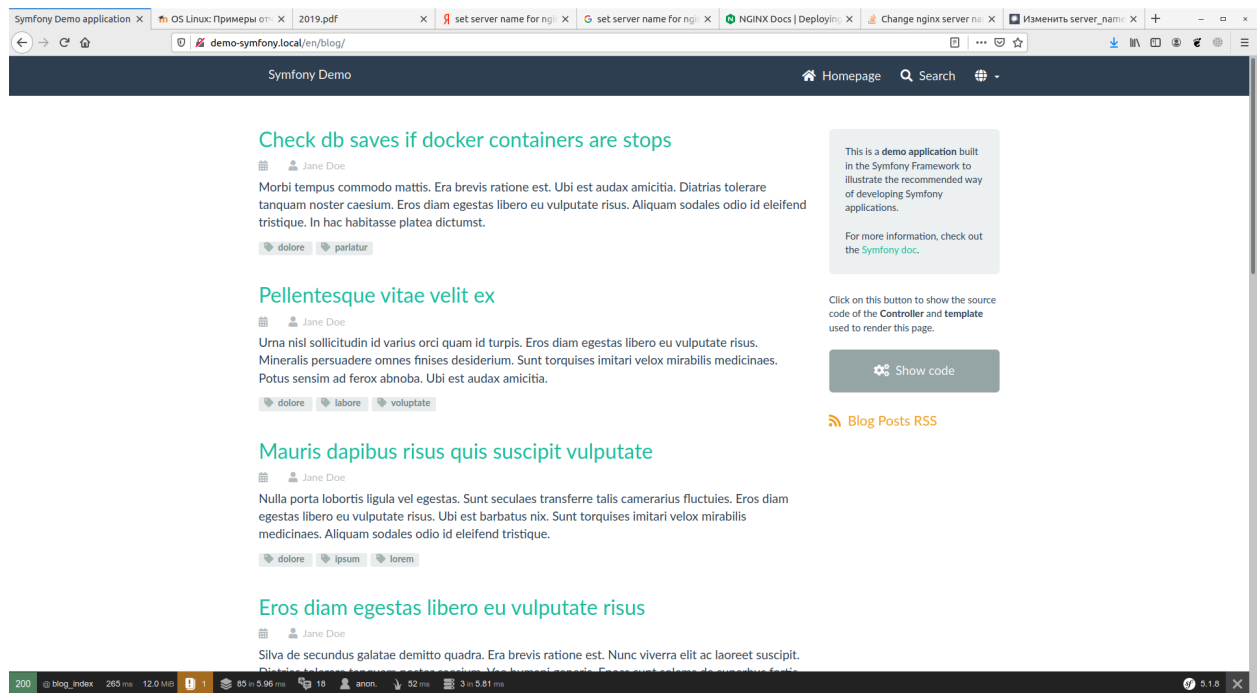


Рисунок 1.10 – Проверка, что БД не удаляется после перезагрузки проекта

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 `db` в наш файл `docker-compose.yml`, получим следующее содержимое:

```
version: "3"
services:
  php-fpm:
    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-fpm
```

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`:

```
kirill@kirill-HP-ENVY-Notebook: /etc/postgresql/10/main
File Edit View Search Terminal Help
GNU nano 2.9.3 postgresql.conf Изменён

# - Connection Settings -

listen_addresses = '*'          # what IP address(es) to listen on;
                                # comma-separated list of addresses;
                                # defaults to 'localhost'; use '*' for all
                                # (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 directories
                                # (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

^G Помощь      ^O Записать    ^W Поиск      ^K Вырезать   ^J Выводить   ^C ТекПозиц
^X Выход      ^R ЧитФайл    ^\ Замена    ^U Отмен. выр ^T Словарь   ^_ К строке
```

Рисунок 1.11 – Изменение файла postgresql.conf

```
kirill@kirill-HP-ENVY-Notebook: /etc/postgresql/10/main
File Edit View Search Terminal Help
GNU nano 2.9.3 pg_hba.conf Изменён

# maintenance (custom daily cronjobs, replication, and similar tasks).
#
# Database administrative login by Unix domain socket
local all postgres peer

# TYPE DATABASE USER ADDRESS METHOD

# "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

^G Помощь      ^O Записать    ^W Поиск      ^K Вырезать   ^J Выводить   ^C ТекПозиц
^X Выход      ^R ЧитФайл    ^\ Замена    ^U Отмен. выр ^T Словарь   ^_ К строке
```

Рисунок 1.12 – Изменение файла pg_hba.conf

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

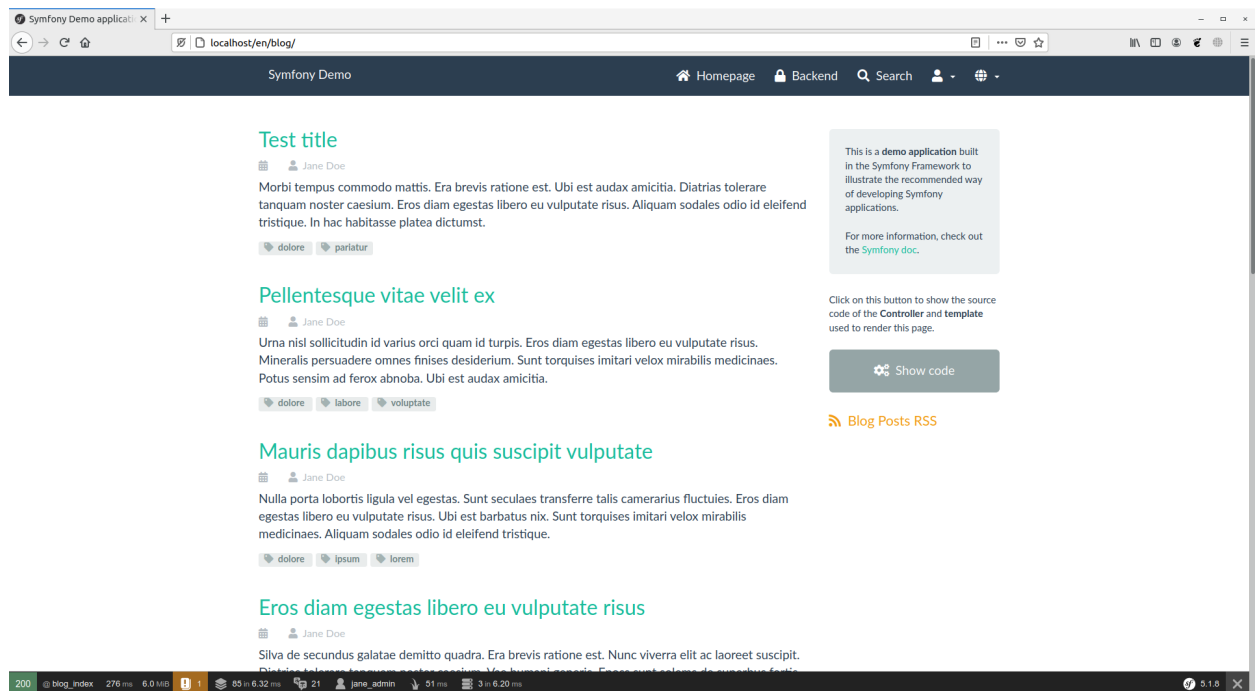


Рисунок 1.13 – Подключение к локальной БД

Контрольные вопросы

1. Назовите отличия использования контейнеров по сравнению с виртуализацией.

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?

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 меж-сетевого 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 разворачивании 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.