

**Липецкий государственный технический университет**

**Факультет автоматизации и информатики**

**Кафедра автоматизированных систем управления**

**ЛАБОРАТОРНАЯ РАБОТА №6**

**по Операционной системе Linux**

**Контейнеризация**

Студент

Лобов М.Ю.

Группа АИ-18

Руководитель

Кургасов В.В.

Доцент, к.п.н.

Липецк 2021 г.

## Оглавление

Цель работы .....	3
Ход работы.....	4
Вывод.....	12
Контрольные вопросы .....	13

## Цель работы

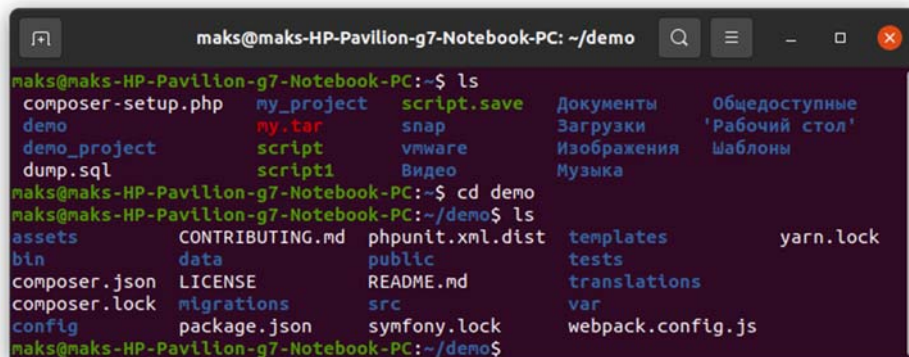
Изучить современные методы разработки ПО в динамических и распределенных средах на примере контейнеров Docker.

## Ход работы

Перед тем, как переходить к выполнению лабораторной работы, нужно установить необходимое ПО, а именно:

- PHP и необходимые расширения (PDO-SQLite PHP, PHP-XML и PHP-pgsql);
- Composer;
- Symfony.

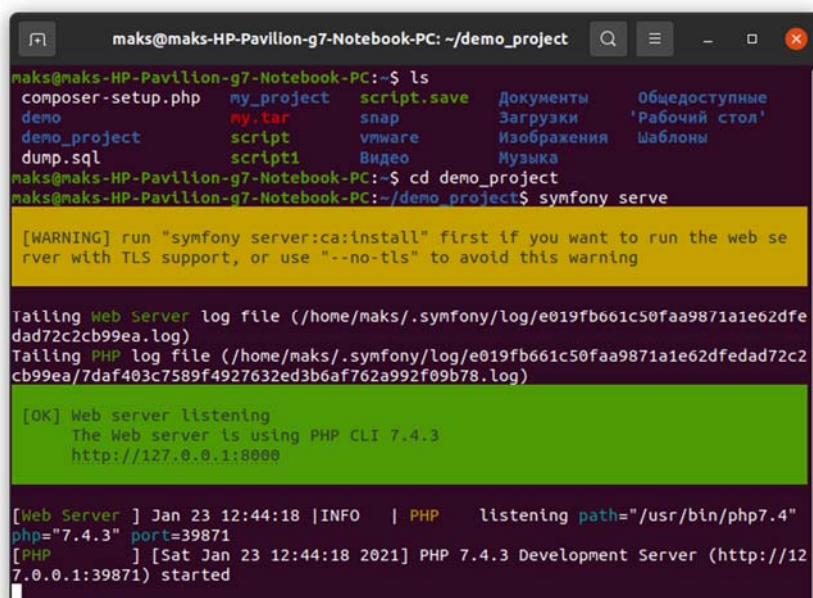
Затем создадим новый демо-проект с помощью команды **symfony new -demo my\_project**. Теперь можем посмотреть структуру проекта:



```
maks@maks-HP-Pavillon-g7-Notebook-PC: ~/demo
maks@maks-HP-Pavillon-g7-Notebook-PC:~$ ls
composer-setup.php  my_project  script.save  Документы  Общедоступные
demo               my.tar      snap         Загрузки   'Рабочий стол'
demo_project       script      vmware       Изображения  Шаблоны
dump.sql           script1     Видео        Музыка
maks@maks-HP-Pavillon-g7-Notebook-PC:~$ cd demo
maks@maks-HP-Pavillon-g7-Notebook-PC:~/demo$ ls
assets      CONTRIBUTING.md  phpunit.xml.dist  templates      yarn.lock
bin         data            public            tests
composer.json  LICENSE        README.md        translations
composer.lock  migrations     src              var
config        package.json   symfony.lock     webpack.config.js
maks@maks-HP-Pavillon-g7-Notebook-PC:~/demo$
```

Рисунок 1 – Структура демо-проекта

После этого зайдём в папку с проектом и выполним команду **symfony serve**, чтобы запустить локальный сервер:



```
maks@maks-HP-Pavillon-g7-Notebook-PC: ~/demo_project
maks@maks-HP-Pavillon-g7-Notebook-PC:~$ ls
composer-setup.php  my_project  script.save  Документы  Общедоступные
demo               my.tar      snap         Загрузки   'Рабочий стол'
demo_project       script      vmware       Изображения  Шаблоны
dump.sql           script1     Видео        Музыка
maks@maks-HP-Pavillon-g7-Notebook-PC:~$ cd demo_project
maks@maks-HP-Pavillon-g7-Notebook-PC:~/demo_project$ symfony serve

[WARNING] run "symfony server:ca:install" first if you want to run the web server with TLS support, or use "--no-tls" to avoid this warning

Tailing Web Server log file (/home/maks/.symfony/log/e019fb661c50faa9871a1e62fedad72c2cb99ea.log)
Tailing PHP log file (/home/maks/.symfony/log/e019fb661c50faa9871a1e62fedad72c2cb99ea/7daf403c7589f4927632ed3b6af762a992f09b78.log)

[OK] Web server listening
The Web server is using PHP CLI 7.4.3
http://127.0.0.1:8000

[Web Server ] Jan 23 12:44:18 |INFO | PHP listening path="/usr/bin/php7.4"
php="7.4.3" port=39871
[PHP ] [Sat Jan 23 12:44:18 2021] PHP 7.4.3 Development Server (http://127.0.0.1:39871) started
```

Рисунок 2 – Поднимаем локальный хост

Проверим работоспособность нашего проекта, для этого перейдём по адресу localhost:8000:

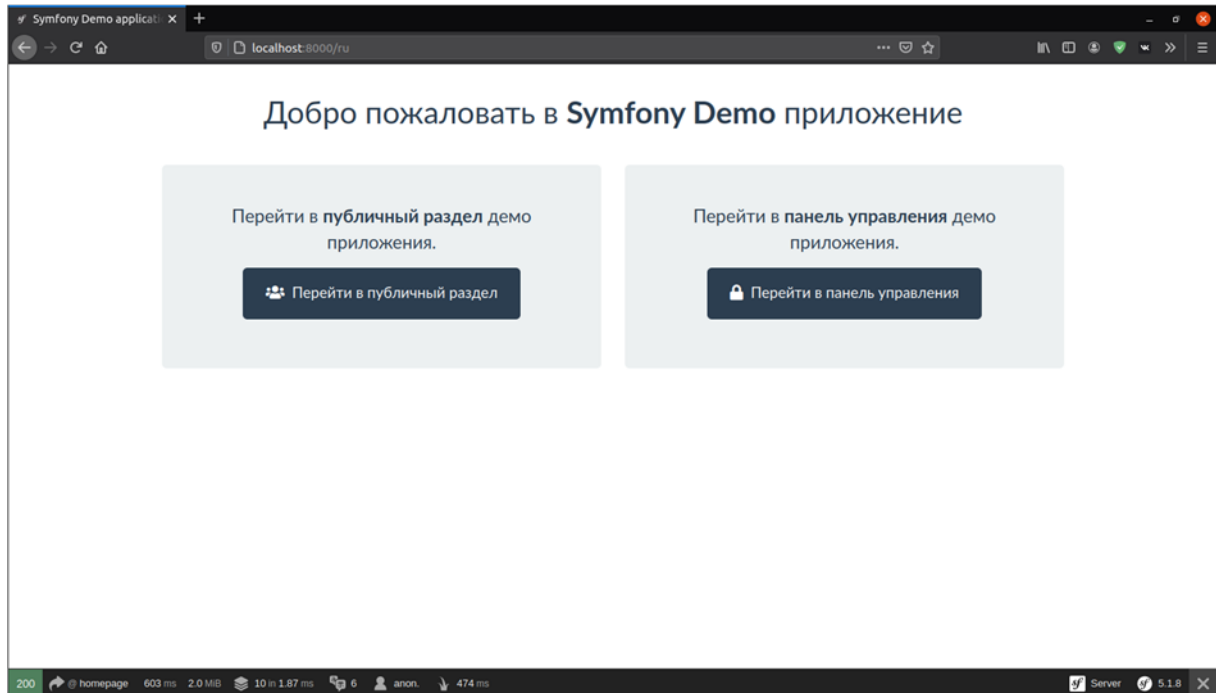


Рисунок 3 – Стартовая страница

Следующим шагом будет создание базы данных в postgres и подключение её к проекту:

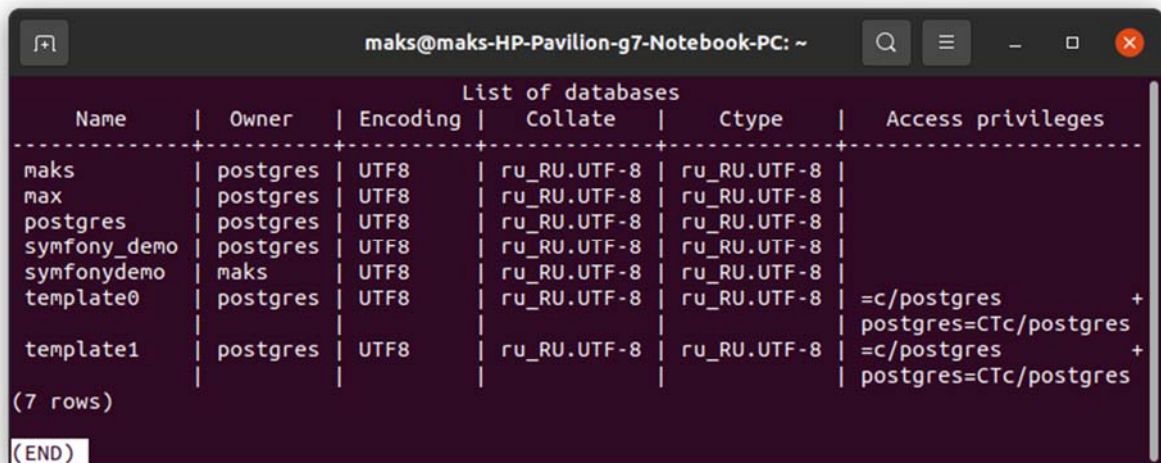
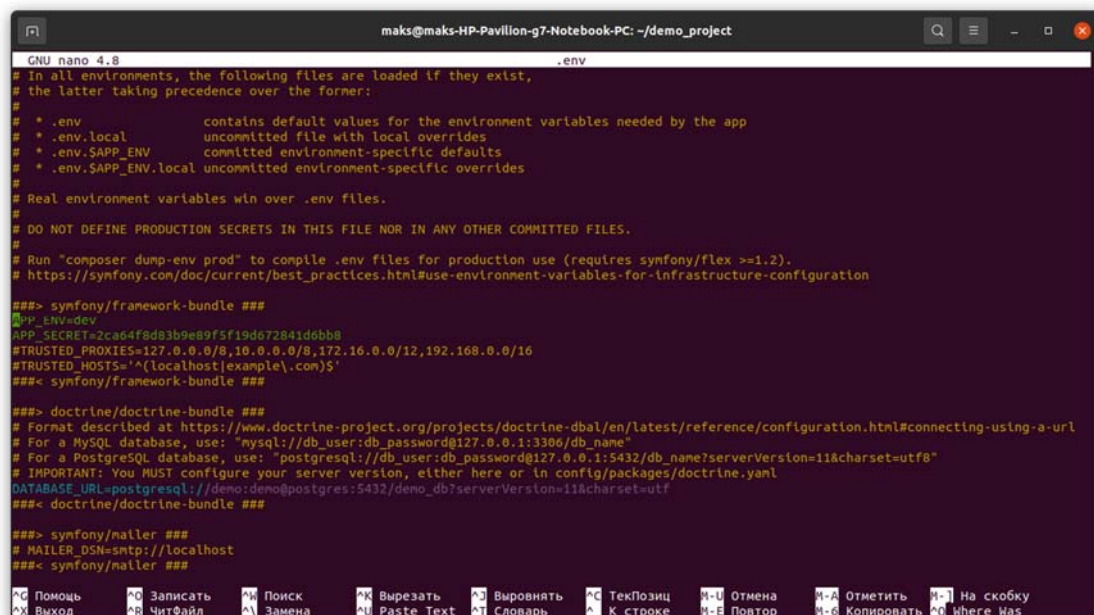


Рисунок 4 – Создание БД symfony\_demo

После создания базы данных изменим файл окружения `.env`, где изменим строку подключения к БД:



```
GNU nano 4.8 .env
# In all environments, the following files are loaded if they exist,
# the latter taking precedence over the former:
#
# * .env contains default values for the environment variables needed by the app
# * .env.local uncommitted file with local overrides
# * .env.SAPP_ENV committed environment-specific defaults
# * .env.SAPP_ENV.local uncommitted environment-specific overrides
#
# Real environment variables win over .env files.
#
# DO NOT DEFINE PRODUCTION SECRETS IN THIS FILE NOR IN ANY OTHER COMMITTED FILES.
#
# Run "composer dump-env prod" to compile .env files for production use (requires symfony/flex >=1.2).
# https://symfony.com/doc/current/best_practices.html#use-environment-variables-for-infrastructure-configuration

###> symfony/framework-bundle ###
APP_ENV=dev
APP_SECRET=2ca64f8d03b9e89f5f19d672841d6bb8
#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#connecting-using-a-url
# 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://demo:demo@postgres:5432/demo_db?serverVersion=11&charset=utf
###< doctrine/doctrine-bundle ###

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

Рисунок 5 – Изменение файла `.env`

Теперь необходимо загрузить схему БД с помощью команды `php bin/console doctrine:schema:create` и заполняем БД с помощью команды `php bin/console doctrine:fixtures:load`.

Снова запустим локальный сервер командой `symfony serve` и проверим работоспособность проекта по адресу `localhost:8000`:

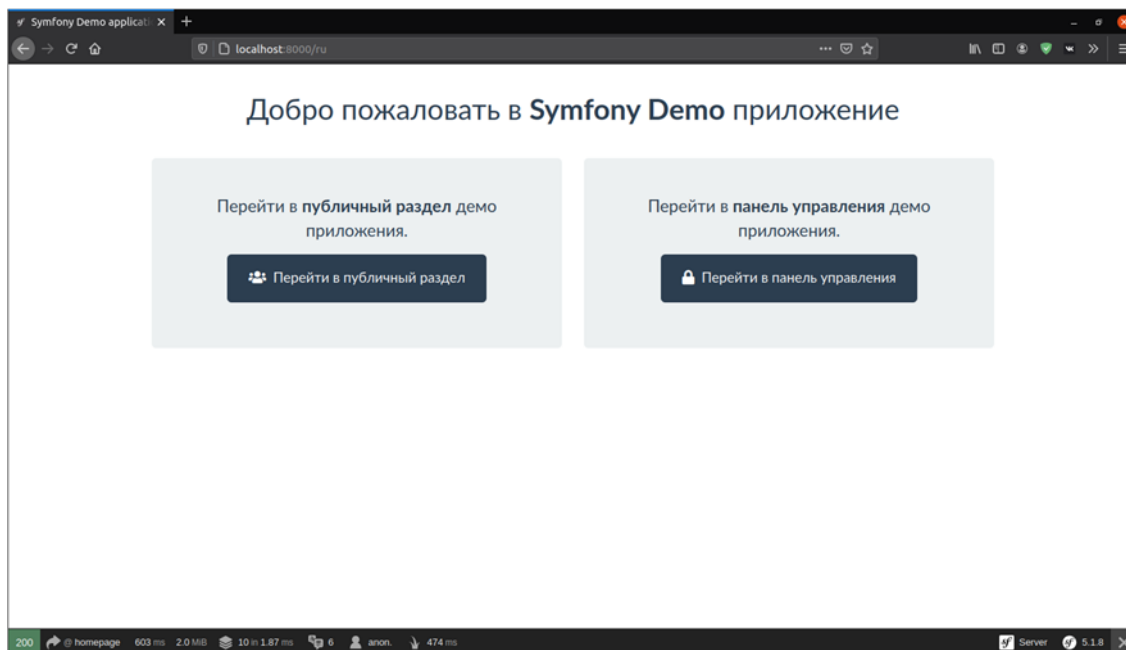


Рисунок 6 – Стартовая страница на postgres

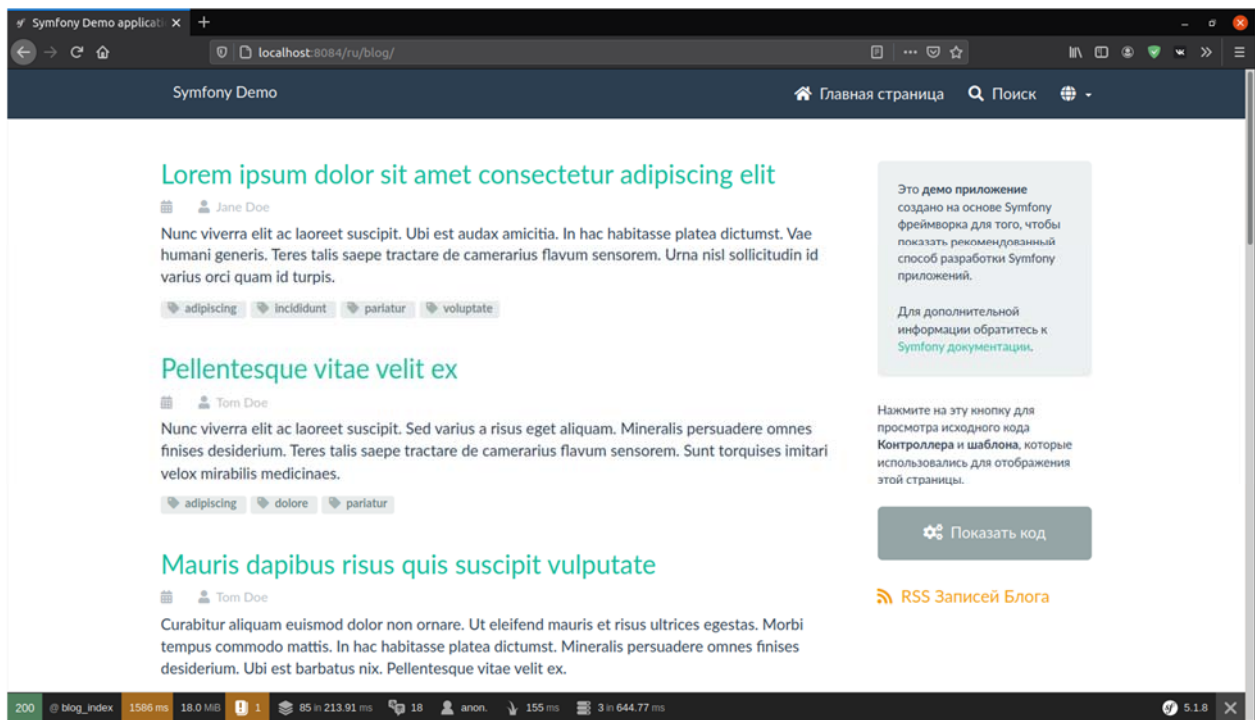


Рисунок 7 – Публичный раздел демо-приложения

Перейдём к настройке контейнеров. Для начала создадим файл `docker-compose.yml` и заполним его следующим образом:

```

GNU nano 4.8                                docker-compose.yml
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

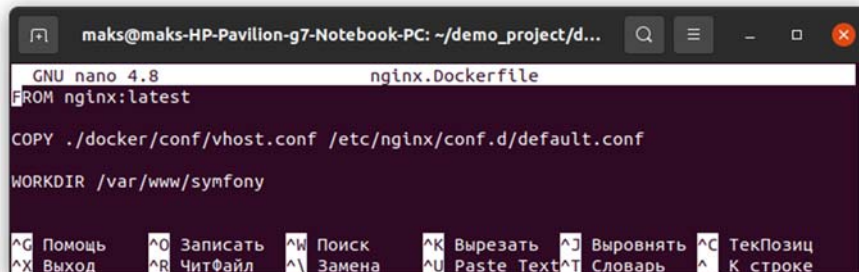
  nginx:
    container_name: nginx
    build:
      context: .
      dockerfile: docker/nginx.Dockerfile
    ports:
      - "8084:80"
    volumes:
      - ./var/www/symfony
      - ./logs/nginx:/var/log/nginx
    links:
      - php-fpm

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

Рисунок 8 – Содержимое файла `docker-compose.yml`



После этого необходимо создать папку docker внутри нашего проекта и в этой папке создать файлы nginx.Dockerfile и php.Dockerfile и папку config, внутри которой будет располагаться файл vhost.conf. Перечисленные файлы имеют следующее содержание:

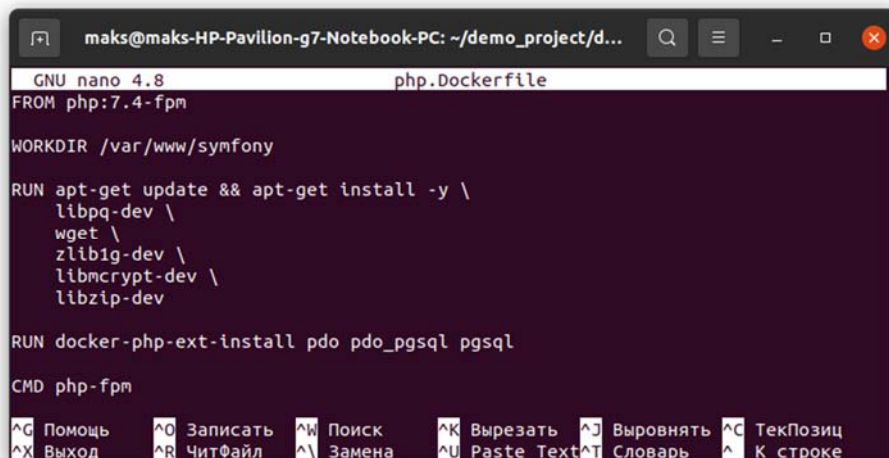


```
GNU nano 4.8 nginx.Dockerfile
FROM nginx:latest

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

WORKDIR /var/www/symfony
```

Рисунок 9 – Содержимое файла nginx.Dockerfile



```
GNU nano 4.8 php.Dockerfile
FROM php:7.4-fpm

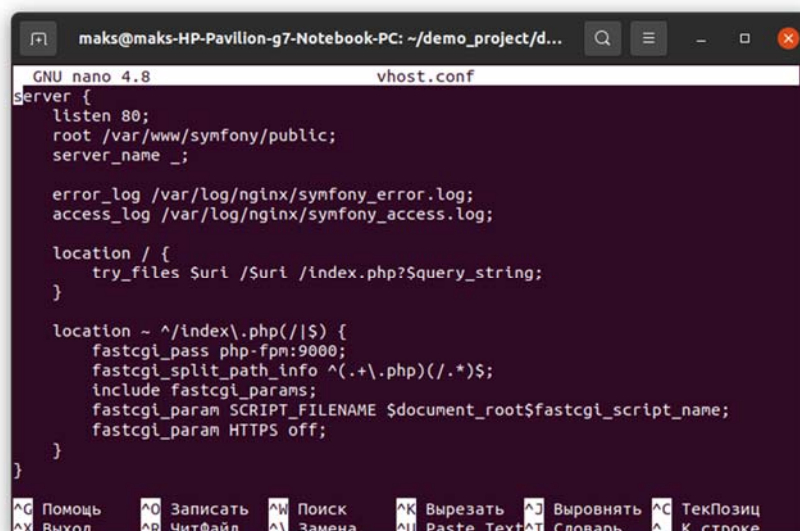
WORKDIR /var/www/symfony

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
```

Рисунок 10 – Содержимое файла php.Dockerfile



```
GNU nano 4.8 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;
    }
}
```

Рисунок 11 – Содержимое файла vhost.conf



Теперь запустим проект с помощью команды **docker-compose up -d** и проверим работоспособность (проект теперь находится по адресу localhost:8084):

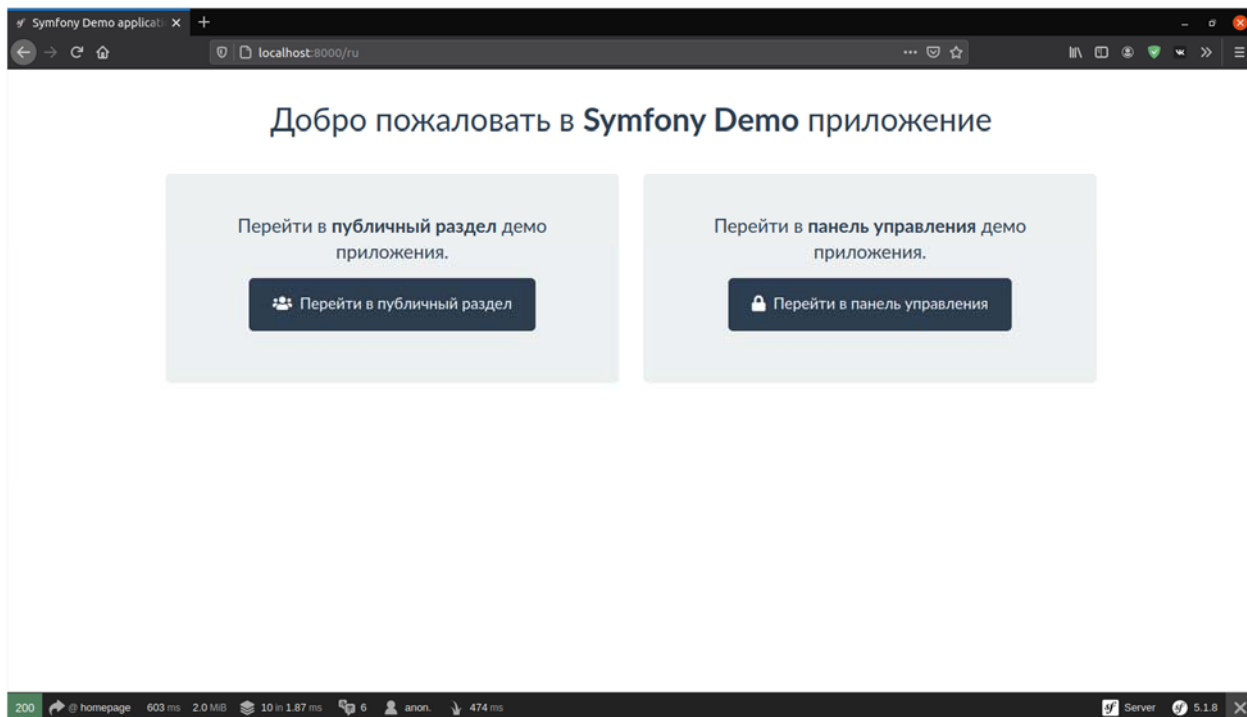


Рисунок 12 – Стартовая страница (запуск через контейнеры)

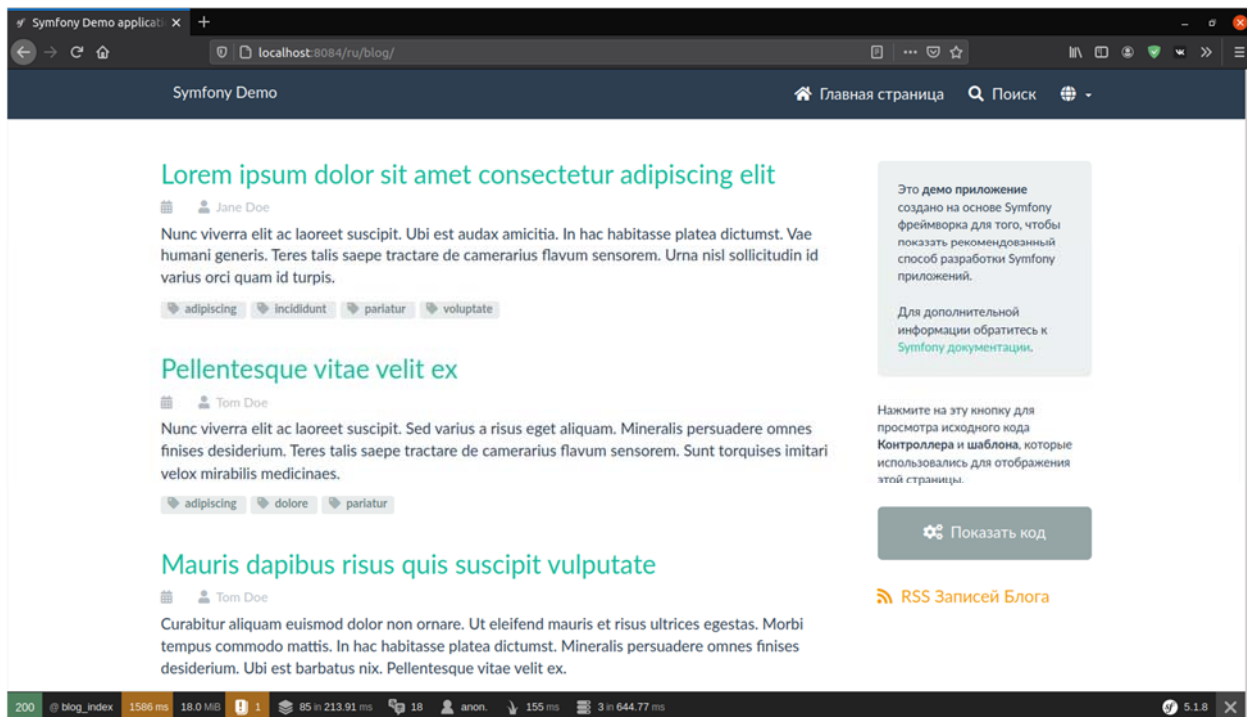
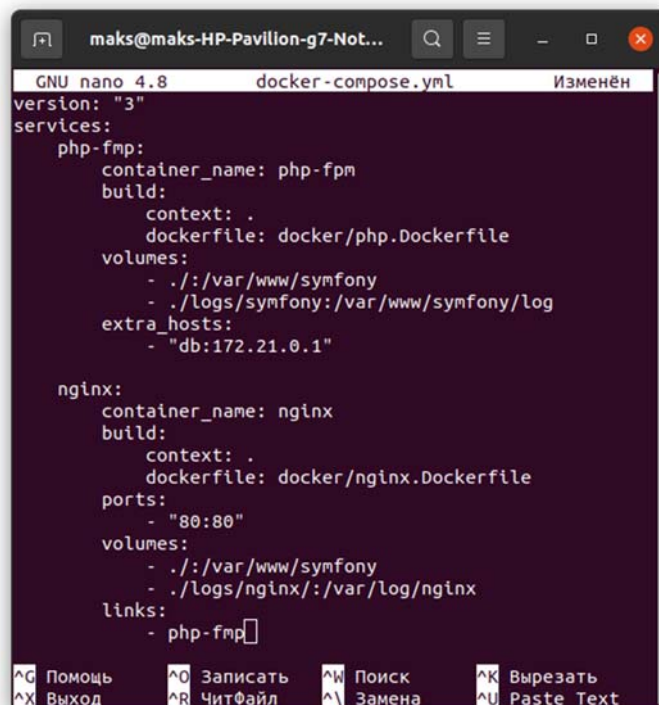


Рисунок 13 – Публичный раздел (запуск через контейнеры)

Далее необходимо подключить проект к локальной базе данных. Изменяем содержимое файла docker-compose.yml:

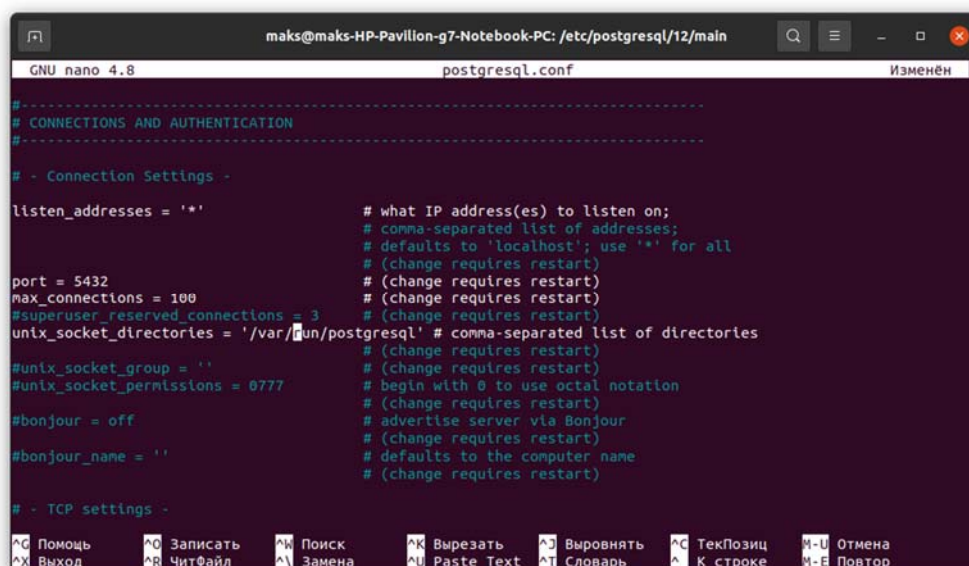


```
GNU nano 4.8 docker-compose.yml Изменён
version: "3"
services:
  php-fpm:
    container_name: php-fpm
    build:
      context: .
      dockerfile: docker/php.Dockerfile
    volumes:
      - ./var/www/symfony
      - ./logs/symfony:/var/www/symfony/log
    extra_hosts:
      - "db:172.21.0.1"

  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
```

Рисунок 14 – Содержимое файла docker-compose.yml при переносе на локальную базу данных

Теперь изменим конфигурационные файлы /etc/postgresql/10/main/postgresql.conf и pg\_hba.conf, чтобы локальная база данных поддерживала запуск из контейнеров:



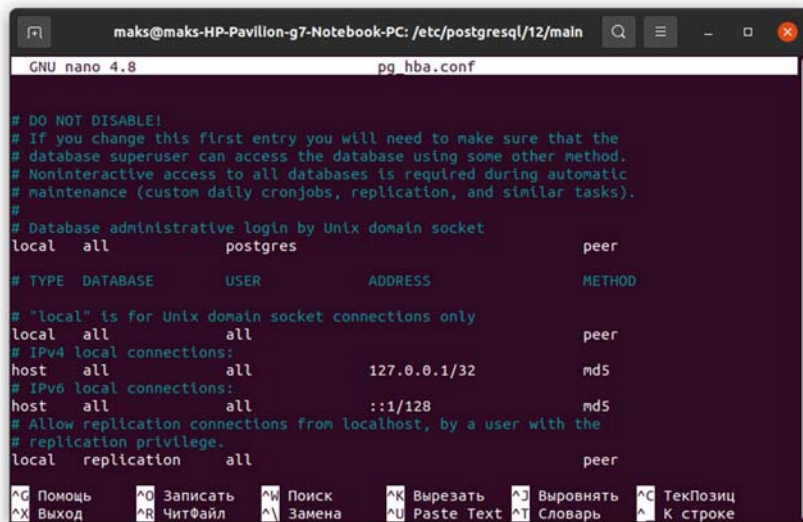
```
GNU nano 4.8 postgresql.conf Изменён
#-----
# CONNECTIONS AND AUTHENTICATION
#-----

# - 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
                                # (change requires restart)

# - TCP settings -
```

Рисунок 15 – Содержимое файла postgresql.conf



```
maks@maks-HP-Pavilion-g7-Notebook-PC: /etc/postgresql/12/main
GNU nano 4.8 pg_hba.conf

# DO NOT DISABLE!
# If you change this first entry you will need to make sure that the
# database superuser can access the database using some other method.
# Noninteractive access to all databases is required during automatic
# 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
# 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

^O Помощь ^O Записать ^M Поиск ^K Вырезать ^O Выводить ^C ТекПозиц
^X Выход ^R ЧитФайл ^\ Замена ^U Paste Text ^T Словарь ^_ К строке
```

Рисунок 16 – Содержимое файла pg\_hba.conf

Теперь перезапустим postgres с помощью команды **sudo service postgres restart** и проверим работоспособность проекта:

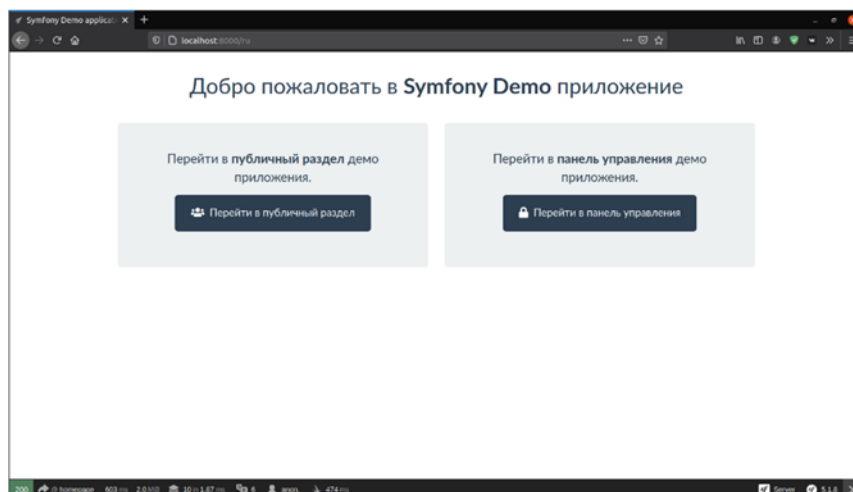


Рисунок 17 – Стартовая страница при запуске на локальной базе данных

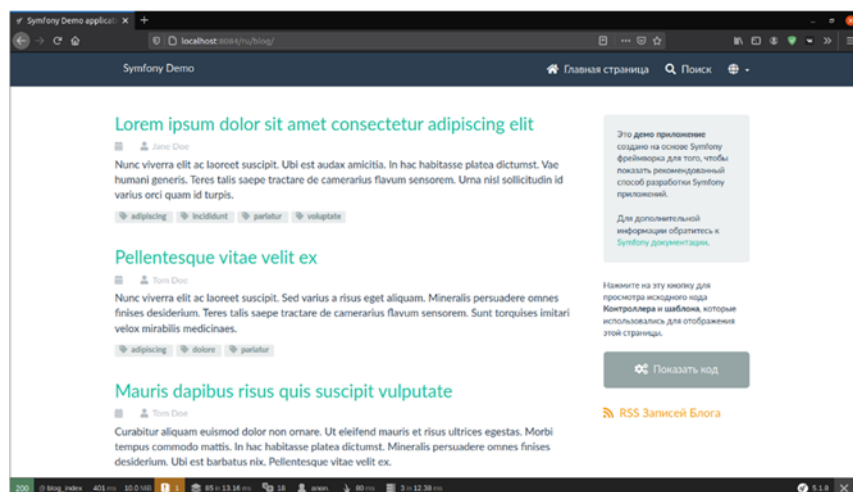


Рисунок 18 – Публичный раздел при запуске на локальной базе данных

## Вывод

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

## 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 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 предпочтительным 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 наличие 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.