

Липецкий государственный технический университет

Кафедра автоматизированных систем управления

Отчет по лабораторной работе № 5
«Контейнерезация»
по курсу «ОС Linux»

Студент

Закиров Р.Р.

Группа АС-20-1

Руководитель

Кургасов В.В.

Липецк 2022 г.

Содержание

Задание кафедры	3
1. Ход работы	4
1.1. Установка необходимого ПО	4
1.1..1 Установка Docker	4
1.1..2 Установка Docker-Compose	5
1.1..3 Установка PHP	6
1.1..4 Установка Composer	6
1.1..5 Установка Composer	7
1.2. Настройка проекта на сервере	7
1.3. Настройка конфигураций Docker-Compose для запуска мультиконтейнерного приложения	9
1.4. Проект на Wordpress.....	17
2. Контрольные вопросы	20

Задание кафедры

1. С помощью Docker Compose на своем компьютере поднять сборку nginx+php-fpm+postgres, продемонстрировать ее работоспособность, запустив внутри контейнера демо-проект на symfony.
2. Создать образ с одним из движков (WordPress, Joomla).

1. Ход работы

1.1. Установка необходимого ПО

Для запуска проекта нужно установить:

1. Docker – для работы с контейнерами
2. Docker Compose – для запуска приложения, состоящего из нескольких контейнеров
3. PHP и его необходимые библиотеки (CLI, FPM и другие)
4. Composer
5. Symfony

1.1.1 Установка Docker

Произведём установку через репозиторий Докера.

Установим пакеты, чтобы разрешить apt использовать репозитории через HTTPS-протокол

```
sudo apt-get update
sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

Добавим официальный GPG-ключ

```
curl -fsSL
https://download.docker.com/linux/debian/gpg |
sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

Добавим официальный репозиторий Докера в `sources.list`, чтобы установить его через `apt`

```
echo \  
"deb [arch=$(dpkg --print-architecture)  
signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]  
https://download.docker.com/linux/debian \  
$(lsb_release -cs) stable" |  
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Устанавливаем Docker через `apt`

```
sudo apt-get update  
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

1.1.2 Установка Docker-Compose

Есть несколько вариантов установки, но мы установим через `pip`.
Установим `pip` через `apt`.

```
sudo apt install pip
```

Далее установим через `pip` Docker-Compose

```
sudo pip install docker-compose
```

1.1.3 Установка PHP

Добавим репозиторий с PHP8

```
sudo apt install software-properties-common
sudo add-apt-repository ppa:ondrej/php
```

Далее установим сам PHP и его библиотеки

```
sudo apt install php8.2
sudo apt-get install php-sqlite3
sudo apt-get install php8.2-xml
sudo apt-get install php8.2-mbstring
sudo apt-get install php8.2-pgsql
sudo apt install postgresql postgresql-contrib
```

1.1.4 Установка Composer

Скачаем установщик с официального сайта

```
php -r "copy(' https://getcomposer.org/installer' , ' composer-setup.php' );
```

Установим Composer, сравнив с хешем

```
php -r
"if (hash_file(' sha384' , ' composer-setup.php' ) ===
' 906a84df04cea2aa72f40b5f787e49f22d4c2f19492ac
310e8cba5b96ac8b64115ac
402c8cd292b8a03482574915d1a8' )
{ echo ' Installer verified' ; }
else
```

```
{ echo ' Installer corrupt' ;  
unlink(' composer-setup.php' ); }  
echo PHP_EOL;"  
php composer-setup.php  
php -r "unlink(' composer-setup.php' );"
```

Переместим установленный файл в /usr/local/bin, чтобы обращаться к нему в любой директории

```
sudo mv composer.phar /usr/local/bin/composer
```

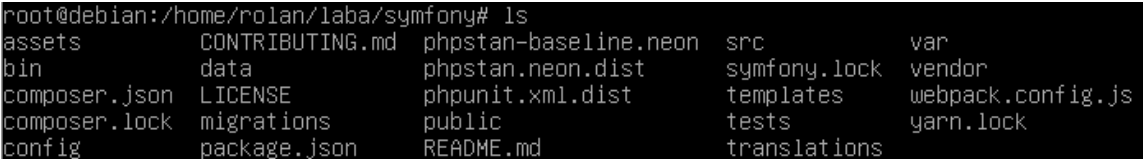
1.1.5 Установка Composer

Проделаем тоже самое, что и с Composer

```
wget https://get.symfony.com/cli/installer -O - | bash  
sudo mv /home/user/.symfony/bin/symfony /usr/local/bin/symfony
```

1.2. Настройка проекта на сервере

Инициализируем демо проект



```
root@debian:/home/rolan/laba/symfony# ls  
assets      CONTRIBUTING.md  phpstan-baseline.neon  src          var  
bin          data             phpstan.neon.dist     symfony.lock vendor  
composer.json LICENSE          phpunit.xml.dist      templates    webpack.config.js  
composer.lock migrations      public                tests        yarn.lock  
config      package.json    README.md             translations
```

Рисунок 1 - Инициализация проекта

Далее меняем конфигурацию в .env для базы данных.

```
#
# * .env contains default values for the environment variables needed by the app
# * .env.local uncommitted file with local overrides
# * .env.$APP_ENV committed environment-specific defaults
# * .env.$APP_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.
# https://symfony.com/doc/current/configuration/secrets.html
#
# Run "composer dump-env prod" to compile .env files for production use (requires symfony/flex >=1.
# https://symfony.com/doc/current/best_practices.html#use-environment-variables-for-infrastructure-

###> symfony/framework-bundle ###
APP_ENV=dev
APP_SECRET=2ca64f8d83b9e89f5f19d672841d6bb8
#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/c
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.ya
#
#DATABASE_URL=sqlite:///kernel.project_dir%/data/database.sqlite
#DATABASE_URL="mysql://db_user:db_password@127.0.0.1:3306/db_name?serverVersion=5.7",
#DATABASE_URL="postgresql://db_user:db_password@127.0.0.1:5432/db_name?serverVersion=13&charset=utf
DATABASE_URL="postgresql://postgres:12345@127.0.0.1:5432/laba_db"
###< doctrine/doctrine-bundle ###
```

Рисунок 2 - /symfony/.env

Проверим работоспособность:

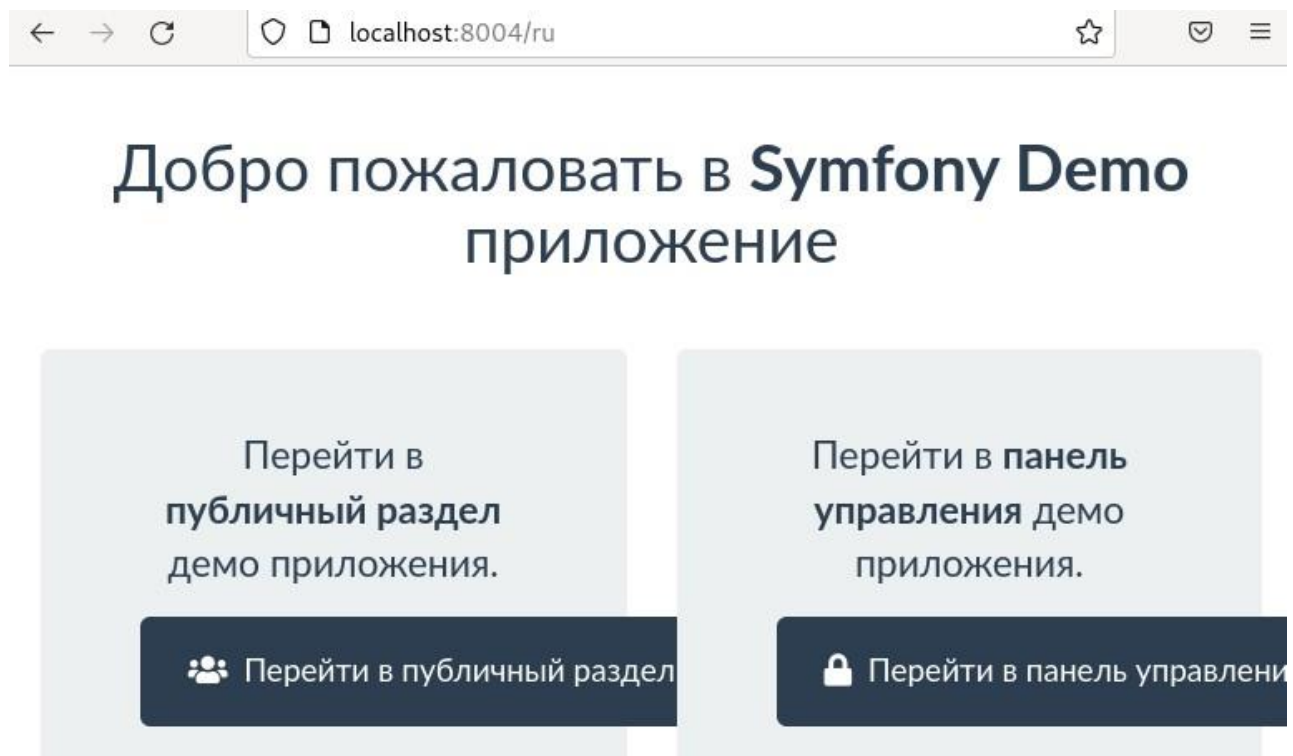


Рисунок 3 – Запуск проекта

1.3. Настройка конфигураций Docker-Compose для запуска мультиконтейнерного приложения

Создадим docker-compose.yml , по которому будет собираться наше приложение

```
GNU nano 5.4 docker-compose.yml
version: '3.8'

services:
  db:
    container_name: db
    image: postgres:12
    restart: always
    environment:
      POSTGRES_PASSWORD: 12345
      POSTGRES_DB: laba_db
    ports:
      - 15432:5432
    volumes:
      - ./data/postgresql:/var/lib/postgresql/data
      - ./laba_db_backup.sql:/docker-entrypoint-initdb.d/laba_db_backup.sql
  php-fpm:
    container_name: php-fpm
    build:
      context: ./php-fpm
    depends_on:
      - db
  nginx:
    container_name: nginx
    build:
      context: ./nginx
    volumes:
      - ../../symfony:/var/www
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf
      - ./nginx/sites:/etc/nginx/sites-available
      - ./nginx/conf.d:/etc/nginx/conf.d
    depends_on:
      - php-fpm
    ports:
      - "8080:80"
      - "443:443"
```



Рисунок 4 - docker-compose.yml

volumes отвечают за то, куда будут смонтированы файлы с сервера в контейнер и наоборот. Слева от двоеточия путь к файлам сервера, справа в контейнера.

environments – это переменные окружения контейнера. Соответственно, для симфони проекта это переменные из docker/.env(который будет описан далее), а для базы данных, её переменные окружения для создания базы данных внутри контейнера.

build отвечает за то, в какой папке лежит Dockerfile для каждого контейнера. Dockerfile отвечает за выполнение команд во время создания контейнера.

ports отвечает за занимаемые порты.

depends_on отвечает за то, какой контейнер за каким будет запускаться в Docker-compose сборке. Это нужно, чтобы не возникало ошибок. Например: контейнер с Symfony уже загрузился, а база данных к нему нет.

Далее нужно создать .env для php-fpm контейнера. Можно скопировать .env из папки symfony, но теперь айпи localhost(127.0.0.1) нужно поменять на название нашего контейнера с базой данных.

```
GNU nano 5.4 .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 >
# * .env.local uncommitted file with local overrides
# * .env.$APP_ENV committed environment-specific defaults
# * .env.$APP_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 FILE>
# https://symfony.com/doc/current/configuration/secrets.html
#
# Run "composer dump-env prod" to compile .env files for production use (requir>
# https://symfony.com/doc/current/best_practices.html#use-environment-variables>
###> symfony/framework-bundle ###
APP_ENV=dev
APP_SECRET=2ca64f8d83b9e89f5f19d672841d6bb8
#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/e>
# IMPORTANT: You MUST configure your server version, either here or in config/p>
#
#DATABASE_URL=sqlite:///kernel.project_dir%/data/database.sqlite
#DATABASE_URL="mysql://wordpress:12345@127.0.0.1:3306/wordpress?serverVersio>
#DATABASE_URL="postgresql://db_user:db_password@127.0.0.1:5432/db_name?serverVe>
DATABASE_URL="postgresql://postgres:12345@127.0.0.1:5432/laba_db"
###< doctrine/doctrine-bundle ###
```

Рисунок 5 - docker/.env

Далее настраиваем php-fpm/Dockerfile

```

GNU nano 5.4 Dockerfile
FROM php:8.2-fpm

RUN apt-get update && \
    apt-get install -y --no-install-recommends libssl-dev zlib1g-dev curl git unzip netcat libxml2-dev \
    pecl install apcu && \
    docker-php-ext-configure pgsql -with-pgsql=/usr/local/pgsql && \
    docker-php-ext-install -j$(nproc) zip opcache intl pdo_pgsql pgsql && \
    docker-php-ext-enable apcu pdo_pgsql sodium && \
    apt-get clean && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*

COPY --from=composer /usr/bin/composer /usr/bin/composer

WORKDIR /var/www

CMD composer i -o ; php-fpm

EXPOSE 9000

```

Рисунок 6 - php-fpm/Dockerfile

В FROM мы указываем название образа с DockerHub, в нашем случае php:8.2-fpm. Скачиваем все утилиты, которые нужны для корректной работы указываем рабочую папку и обновляем зависимости.

Далее настраиваем nginx/Dockerfile

```

GNU nano 5.4 Dockerfile
FROM nginx:alpine

WORKDIR /var/www

CMD ["nginx"]

EXPOSE 80 443

```

Рисунок 7 - nginx/Dockerfile

Далее настраиваем nginx/conf.d/default.conf

```

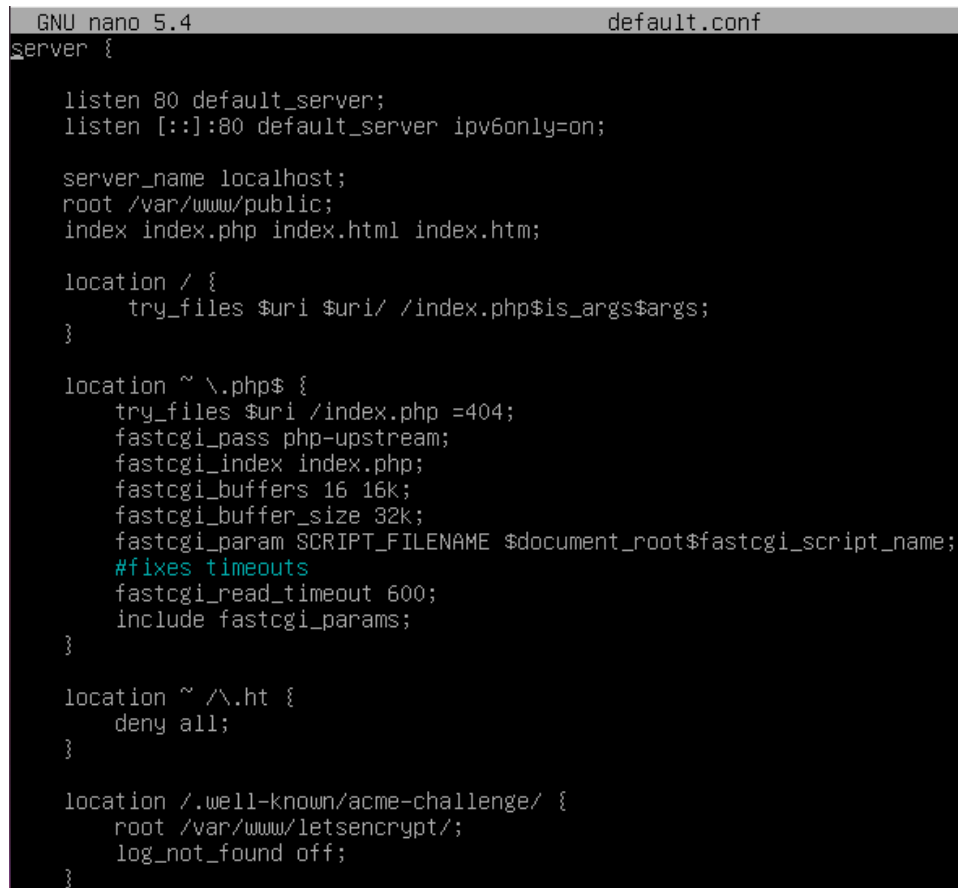
GNU nano 5.4 default.conf
upstream php-upstream {
    server php-fpm:9000;
}

```

Рисунок 8 - nginx/conf.d/default.conf

Директива `upstream` позволяет распределять запросы по прокси на другие серверы, в нашем случае это контейнер `php-fpm`. В настройке самого сервера `php-upstream` будет установлен в `fastcgi_pass`.

Далее установим конфигурации для `nginx` внутри контейнера



```
GNU nano 5.4 default.conf
server {

    listen 80 default_server;
    listen [::]:80 default_server ipv6only=on;

    server_name localhost;
    root /var/www/public;
    index index.php index.html index.htm;

    location / {
        try_files $uri $uri/ /index.php$is_args$args;
    }

    location ~ \.php$ {
        try_files $uri /index.php =404;
        fastcgi_pass php-upstream;
        fastcgi_index index.php;
        fastcgi_buffers 16 16k;
        fastcgi_buffer_size 32k;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        #fixes timeouts
        fastcgi_read_timeout 600;
        include fastcgi_params;
    }

    location ~ /\.ht {
        deny all;
    }

    location /.well-known/acme-challenge/ {
        root /var/www/letsencrypt/;
        log_not_found off;
    }

}
```

Рисунок 10 - `nginx/sites/default.conf`

```

GNU nano 5.4                               nginx.conf
user  nginx;
worker_processes  4;
daemon off;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include        /etc/nginx/mime.types;
    default_type   application/octet-stream;

    access_log    /var/log/nginx/access.log;
    #access_log   /dev/stdout;
    #error_log    /dev/stderr;

    sendfile      on;
    #tcp_nopush   on;

    keepalive_timeout  65;

    gzip  on;

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-available/*.conf;
}

```

Рисунок 11 - nginx/nginx.conf

Далее установим конфигурацию для запуска сервера nginx из контейнера. Указав разрешённые файлы, переменный FastCGI и другие настройки.

В принципе, у нас уже есть рабочий проект, но на нём нет баз данных, т.к. мы не добавили её в наш контейнер. Есть 2 пути решения, собрать проект и добавить её в уже работающий контейнер или смонтировать и добавить её на стадии сборки приложения. В нашем случае используем второй вариант.

Вернёмся к docker-compose.yml и увидим в volumes у db строчку начинающуюся с ./laba_db_backup.sql . Это дампы базы данных, который необходимо смонтировать к нашему контейнеру. Давайте создадим его.

Для этого зайдём под пользователем postgres и запустим утилиту pg_dump и по URL базы данных сделаем дамп в файл laba_db_backup.sql

```

root@debian:/home/rolan/symfony/docker/nginx# sudo su - postgres
postgres@debian:~$ pg_dump postgresql://postgres:12345@127.0.0.1:5432/laba_db > laba_db_backup.sql
postgres@debian:~$ ls
13 docker laba_db_backup.sql

```

Рисунок 11 - Создание дампа базы данных

Копируем её в /docker.

Проект настроен, осталось только собрать. Используем команду docker-compose build

```
-->709db286ed54
Step 7/8 : CMD composer i -o ; wait-for-it db:5432; php-fpm
--> Using cache
--> 98bd55aa4062
Step 8/8 : EXPOSE 9000
--> Using cache
--> 71889290e580
Successfully built 71889290e580
Successfully tagged docker_php-fpm:latest
Building nginx
Sending build context to Docker daemon 7.168kB
Step 1/4 : FROM nginx:alpine
-->cc44224bfe20
Step 2/4 : WORKDIR /var/www
--> Using cache
--> c663063eb9d9
Step 3/4 : CMD ["nginx"]
--> Using cache
--> 3fd249c2a558
Step 4/4 : EXPOSE 80 443
--> Using cache
--> 5d6349aa1a0d
Successfully built 5d6349aa1a0d
Successfully tagged docker_nginx: latest
Creating db ... done
Creating php-fpm ...
done
Creating nginx
done
```

Рисунок 12 - Сборка проекта

Теперь запускаем, чтобы проверить. Чтобы запустить наше приложение пишем docker-compose up -d.

```
db is up-to-date
php-fpm is up-to-date
nginx is up-to-date
```

Рисунок 13 - Инициализация БД

Далее зайдём по айпи nginx и порту, указанному в docker-compose.yml на наш сайт my-server.local

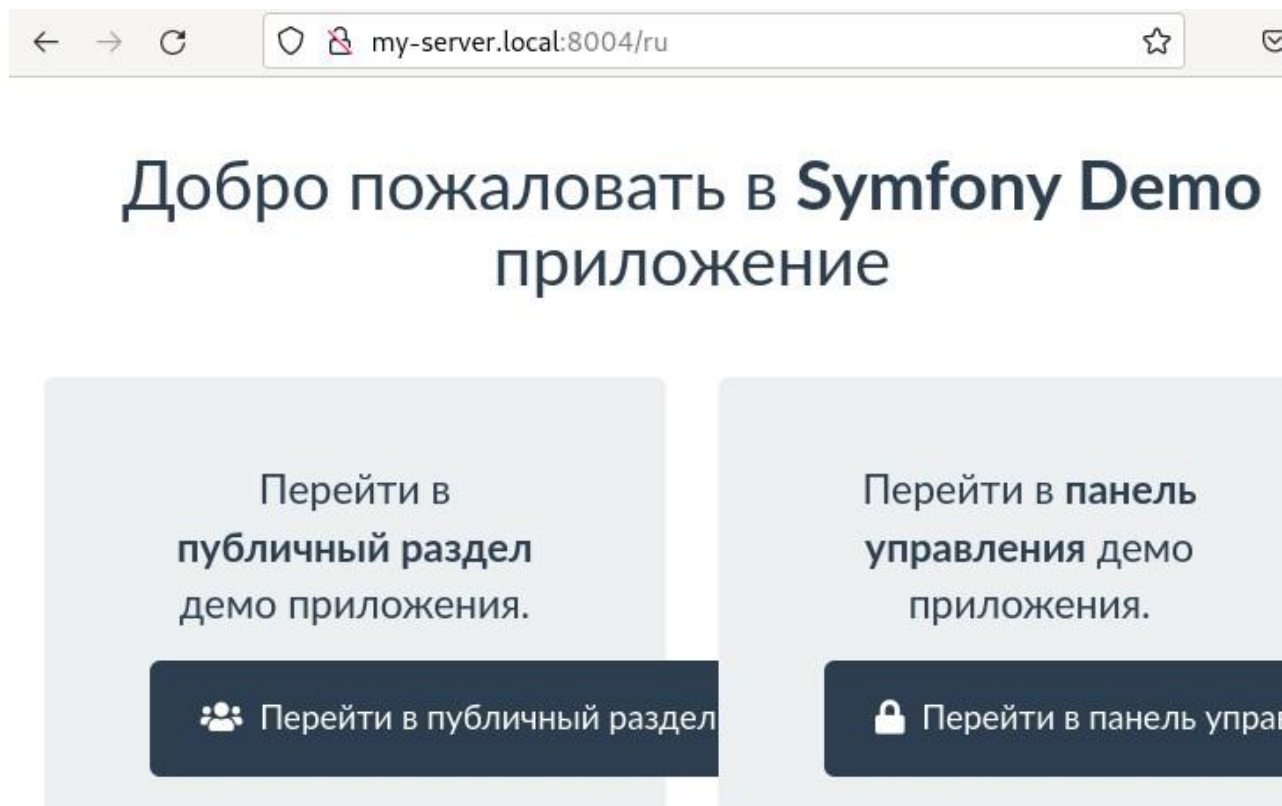


Рисунок 14 - Результат 1

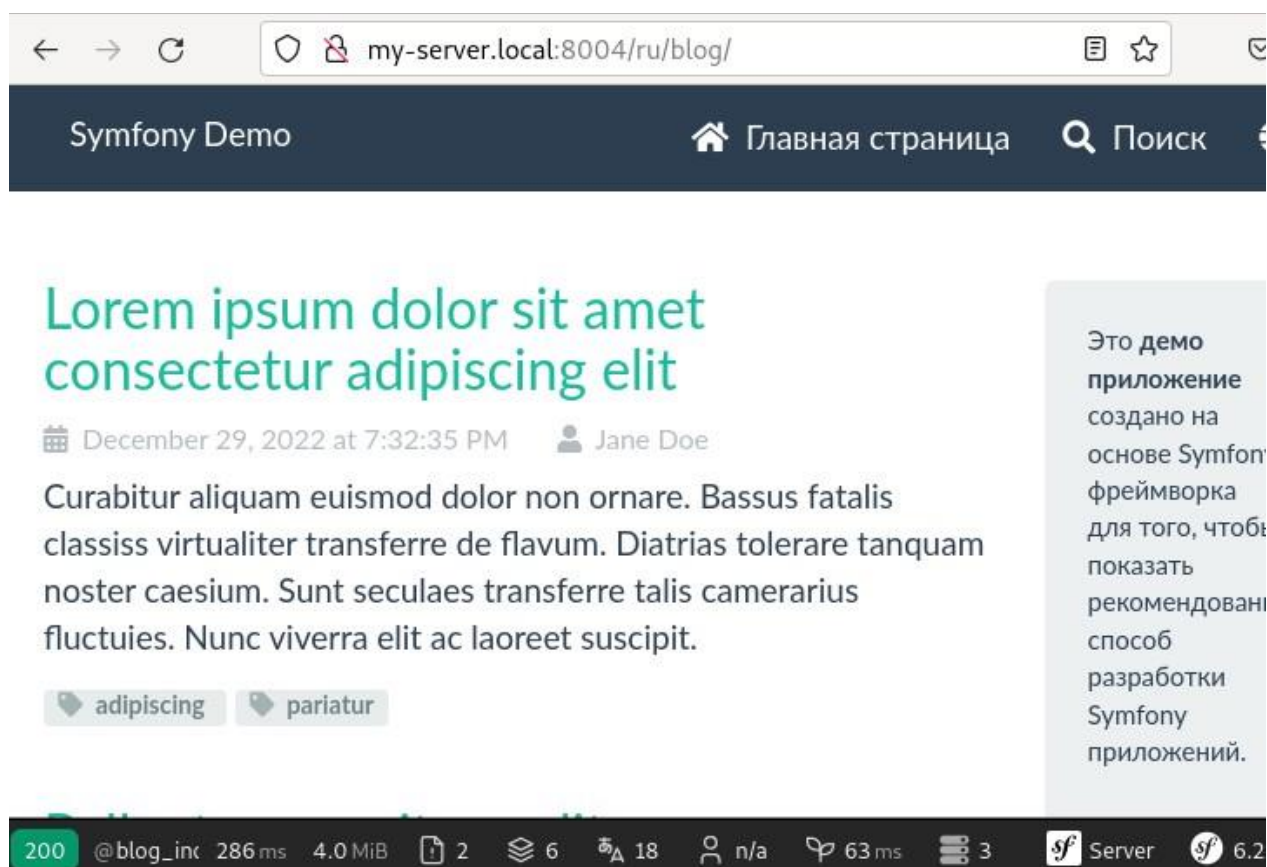


Рисунок 15 - Результат 2

1.4. Проект на Wordpress

Делаем аналогично предыдущим шагам

```
GNU nano 5.4                                docker-compose.yml
version: '3.8'

services:
  wordpress:
    image: wordpress:latest
    restart: always
    links:
      - db:mysql
    ports:
      - 8080:80
    working_dir: /var/www/html
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: 12345
      WORDPRESS_DB_NAME: wp_lab
    volumes:
      - wordpress:/var/www/html

  db:
    image: mysql:latest
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: neskagu
      MYSQL_DATABASE: wp_lab
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: 12345
    volumes:
      - db:/var/lib/mysql
```

Рисунок 16 - docker-compose.yml

Далее собираем и запускаем проект командой `docker-compose up -d --build`

Результат:

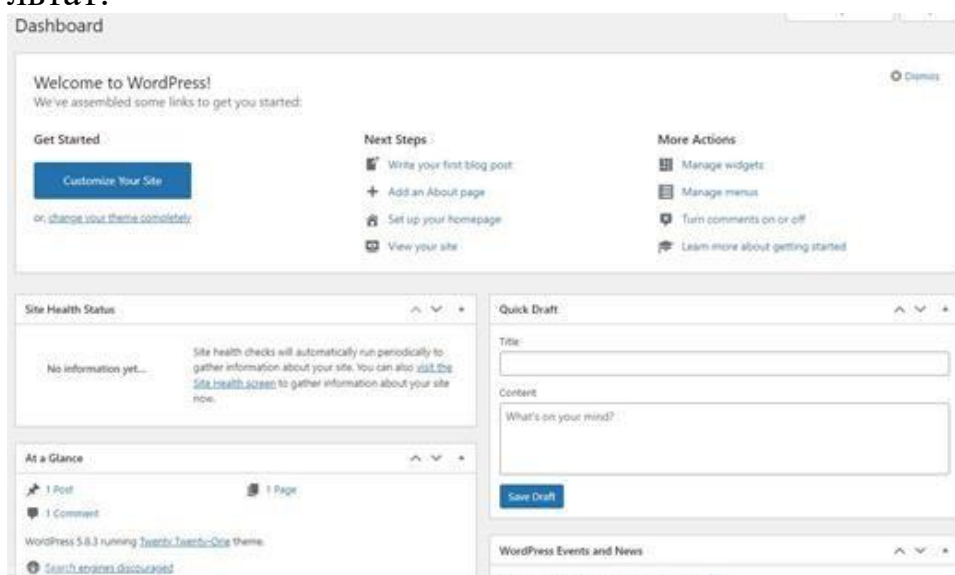


Рисунок 19 - Результат

2. Контрольные вопросы

1. Вопрос: Назовите отличия использования контейнеров по сравнению с виртуализацией.

Ответ: Меньшие накладные расходы на инфраструктуру

2. Вопрос: Назовите основные компоненты Docker.

Ответ: Контейнеры

3. Вопрос: Какие технологии используются для работы с контейнерами?

Ответ: Контрольные группы (cgroups)

4. Вопрос: Найдите соответствие между компонентом и его описанием

Ответ:

- 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. Вопрос: В чем отличие контейнеров от виртуализации?

Ответ: Контейнер обеспечивает виртуализацию на уровне операционной системы, а аппаратная виртуализация обеспечивает виртуализацию на уровне машины.

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

Ответ: — `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. Вопрос: Каким образом осуществляется поиск образов контейнеров?

Ответ: Сначала проверяется локальный репозиторий на наличия нужного контейнера, если он не найден локально, то поиск производится в репозитории Docker Hub.

8. Вопрос: Каким образом осуществляется запуск контейнера?

Ответ: Команда `docker run (container_name)`

9. Вопрос: Что значит управлять состоянием контейнеров?

Ответ: Это означает, что в любой момент времени есть возможность запустить, остановить или выполнить команды внутри контейнера.

10. Вопрос: Как изолировать контейнер?

Ответ: Контейнеры уже по сути своей являются изолированными единицами, поэтому достаточно без ошибок сконфигурировать файлы Dockerfile и/или docker-compose.yml.

11. Вопрос: Опишите последовательность создания новых образов, назначение Dockerfile?

Ответ: 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 иденти-

кацию 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. Вопрос: Возможно ли работать с контейнерами Docker без одноименного движка?

Ответ: Да. Существует Kubernetes.

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

Ответ: Оркестрация — обеспечение совместной работы всех элементов системы. Запуск контейнеров на соответствующих хостах и установление соединений между ними. Организационная система также может включать поддержку масштабирования, автоматического восстановления после критических сбоев и инструменты изменения балансировки нагрузки на узлы. Kubernetes – это высокоуровневое решение оркестровки, в которое по умолчанию встроены функции восстановления после критических сбоев и масштабирования и которое может работать поверх других решений кластеризации.

— 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.

НПО