



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ЛИПЕЦКИЙ
ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет автоматизации и информатики
Кафедра автоматизированных систем управления

ЛАБОРАТОРНАЯ РАБОТА №5
по курсу “ОС Linux”

Студент ПИ-21-1

(подпись, дата)

Красиков И. А.

Руководитель

(подпись, дата)

Кургасов В.В.

Липецк 2023

Цель работы.....	3
Задание	3
I часть	5
Установка необходимых инструментов	5
Выполнение работы	9
II часть	18
Шаг №1. Установка Nginx.....	18
Шаг №2. Передача в контейнер html-файлов	19
Шаг №3. Web-разработка	20
Шаг №4. Запуск Wordpress.	25
Вопросы для самопроверки	28

Цель работы

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

Задание

I часть

С помощью Docker Compose на своем компьютере поднять сборку nginx+phpfpm+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.

II часть

Шаг №1. Установка Nginx Для начала необходимо установить один лишь Nginx. Что требует создания compose-файла включая директиву ports, иначе порт будет доступен только внутри контейнера и nginx через браузер уже будет недоступен.

Шаг №2. Передача в контейнер html-файлов. В этом нам поможет volumes, которая говорит, что происходит монтирование локальной папки в контейнер по указанному адресу. При монтировании папка по указанному адресу внутри контейнера заменяется папкой с локального компьютера. Необходимо создать папку html на одном уровне с docker-compose.yml и добавить в нее файл index.html с произвольным текстом «Ваш текст», после чего пересоздадим контейнер (docker-compose up -d).

Шаг 3. Web-разработка. Создать папку проху и в ней сборку dockercompose.yml для обращения по домену и пробросу такого домена на основной контейнер. И сборку nginx, php, mysql и phpmyadmin с использованием проху сети.

Шаг 4. Имеется работающий Web-сервер. Создайте образ с одним из движков (WordPress, Joomla). Папка для хранения внешних данных с курсами должна быть Вами определена.

I часть

Установка необходимых инструментов

Установка Docker и docker compose:

Установка пакетов позволяющих apt использовать пакеты через HTTPS

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

```
ivan_lr@intarolr:~$ sudo apt install apt-transport-https ca-certificates curl software-properties-common
Чтение списков пакетов... Готово
Построение дерева зависимостей... Готово
Чтение информации о состоянии... Готово
Уже установлен пакет ca-certificates самой новой версии (20230311ubuntu0.22.04.1).
ca-certificates помечен как установленный вручную.
Будут установлены следующие дополнительные пакеты:
  libcurl4 python3-software-properties
Следующие НОВЫЕ пакеты будут установлены:
  apt-transport-https
Следующие пакеты будут обновлены:
  curl libcurl4 python3-software-properties software-properties-common
```

Рис 1 – Установка пакетов для доступа к пакетам через HTTPS

Добавляем ключ GPG для официального репозитория Docker в нашу систему

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

```
ivan_lr@intarolr:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d instead (see apt-key(8)).
OK
ivan_lr@intarolr:~$ |
```

Рис 2 – Добавление ключа GPG

Добавляем репозиторий Docker в источник APT

```
sudo add-apt-repository "deb [arch=amd64]
```

```
https://download.docker.com/linux/ubuntu focal stable"
```

```
ivan_lr@intarolr:~$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
Repository: 'deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable'
Description:
Archive for codename: focal components: stable
More info: https://download.docker.com/linux/ubuntu
Adding repository.
Press [ENTER] to continue or Ctrl-c to cancel.
Adding deb entry to /etc/apt/sources.list.d/archive_uri-https_download_docker_com_linux_ubuntu-jammy.list
Adding disabled deb-src entry to /etc/apt/sources.list.d/archive_uri-https_download_docker_com_linux_ubuntu-jammy.list
Сущ:1 http://ru.archive.ubuntu.com/ubuntu jammy InRelease
Поп:2 http://ru.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
```

Рис 3 – Добавление репозитория Docker в источник APT

Установка Docker

`sudo apt-get install docker-ce`

```
ivan_lr@intarolr:~$ sudo apt-get install docker-ce
Чтение списков пакетов... Готово
Построение дерева зависимостей... Готово
Чтение информации о состоянии... Готово
Будут установлены следующие дополнительные пакеты:
  containerd.io docker-buildx-plugin docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libslirp0 pigz
  slirp4netns
Предлагаемые пакеты:
  aufs-tools cgroupfs-mount | cgroup-lite
Следующие НОВЫЕ пакеты будут установлены:
  containerd.io docker-buildx-plugin docker-ce docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libslirp0
  pigz slirp4netns
Обновлено 0 пакетов, установлено 9 новых пакетов, для удаления отмечено 0 пакетов, и 66 пакетов не обновлено.
Необходимо скачать 115 МВ архивов.
После данной операции объём занятого дискового пространства возрастёт на 411 МВ.
Хотите продолжить? [Д/н]
```

Рис 3 – Установка Docker

Настройка команды docker без sudo

```
ivan_lr@intarolr:~$ sudo usermod -aG docker ${USER}
ivan_lr@intarolr:~$ su - ${USER}
Password:
ivan_lr@intarolr:~$ id -nG
ivan_lr sudo docker
ivan_lr@intarolr:~$ |
```

Рис 4 – Docker без sudo

Установка Docker-compose

`sudo curl -L`

`"https://github.com/docker/compose/releases/download/1.26.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose`

```
ivan_lr@intarolr:~$ sudo curl -L "https://github.com/docker/compose/releases/download/1.26.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left    Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--    0
100 11.6M 100 11.6M    0     0 4986k      0  0:00:02  0:00:02 --:--:-- 7051k
ivan_lr@intarolr:~$ |
```

Рис 5 – Установка Docker-compose 1 этап

`sudo chmod +x /usr/local/bin/docker-compose`

```
ivan_lr@intarolr:~$ sudo chmod +x /usr/local/bin/docker-compose
```

Рис 6 – Установка Docker-compose 2 этап

Проверка успешности установки

```
ivan_lr@intarolr:~$ docker-compose --version
docker-compose version 1.26.0, build d4451659
ivan_lr@intarolr:~$ docker --version
Docker version 24.0.7, build afdd53b
ivan_lr@intarolr:~$ |
```

Рис 6 – Успешность установки

Установка PHP

`sudo apt install php8.1`

```
ivan_lr@intarolr:~$ sudo apt install php8.1
Чтение списков пакетов... Готово
Построение дерева зависимостей... Готово
Чтение информации о состоянии... Готово
Будут установлены следующие дополнительные пакеты:
apache2 apache2-bin apache2-data apache2-utils libapache2-mod-php8.1
libaprutil1-ldap liblua5.3-0 php-common php8.1-cli php8.1-fpm
```

Рис 7 – Установка php

`sudo apt install php8.1-mbstring php-sqlite3 php8.1-pgsql php8.1-xml`

```
ivan_lr@intarolr:~$ sudo apt install php8.1-mbstring php-sqlite3 php8.1-pgsql php8.1-xml
Чтение списков пакетов... Готово
Построение дерева зависимостей... Готово
Чтение информации о состоянии... Готово
Уже установлен пакет php8.1-mbstring самой новой версии (8.1.2-1ubuntu2.14).
Будут установлены следующие дополнительные пакеты:
libpq5 php8.1-sqlite3
Следующие НОВЫЕ пакеты будут установлены:
libpq5 php-sqlite3 php8.1-pgsql php8.1-sqlite3 php8.1-xml
Обновлено 0 пакетов, установлено 5 новых пакетов, для удаления отмечено 0 пакетов, и 66 пакетов не обновлено.
Необходимо скачать 361 кВ архивов.
После данной операции объём занятого дискового пространства возрастёт на 1 285 кВ.
Хотите продолжить? [д/н]
```

Рис 8 – Установка дополнительных пакетов для php

Установка Postgresql

`sudo apt install postgresql postgresql-contrib`

```
ivan_lr@intarolr:~$ sudo apt install postgresql postgresql-contrib
Чтение списков пакетов... Готово
Построение дерева зависимостей... Готово
Чтение информации о состоянии... Готово
Будут установлены следующие дополнительные пакеты:
libcommon-sense-perl libjson-perl libjson-xs-perl libllvm14 libsensors-config libsensors5 libtypes-serialiser-perl
postgresql-14 postgresql-client-14 postgresql-client-common postgresql-common sysstat
```

Рис 9 – Установка Postgresql

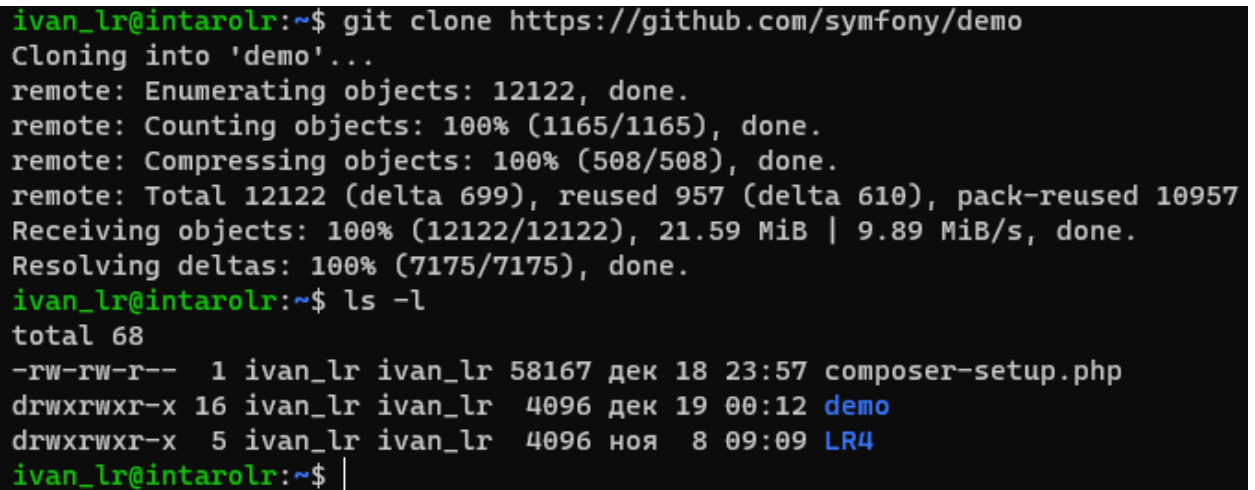
Установка Symfony

```
sudo curl -LsS https://symfony.com/installer -o /usr/local/bin/symfony
```

```
sudo chmod a+x /usr/local/bin/symfony
```

Выполнение работы

1) Клонирование тестовый проект



```
ivan_lr@intarolr:~$ git clone https://github.com/symfony/demo
Cloning into 'demo'...
remote: Enumerating objects: 12122, done.
remote: Counting objects: 100% (1165/1165), done.
remote: Compressing objects: 100% (508/508), done.
remote: Total 12122 (delta 699), reused 957 (delta 610), pack-reused 10957
Receiving objects: 100% (12122/12122), 21.59 MiB | 9.89 MiB/s, done.
Resolving deltas: 100% (7175/7175), done.
ivan_lr@intarolr:~$ ls -l
total 68
-rw-rw-r-- 1 ivan_lr ivan_lr 58167 дек 18 23:57 composer-setup.php
drwxrwxr-x 16 ivan_lr ivan_lr 4096 дек 19 00:12 demo
drwxrwxr-x  5 ivan_lr ivan_lr 4096 ноя  8 09:09 LR4
ivan_lr@intarolr:~$ |
```

Рис 13 – Клонирование тестового проекта

2) Переходим в папку с проектом

С помощью команды `cd demo`

3) Запустим тестовый проект

Но перед тем как запустить проект нам нужно настроить проброс портов в VirtualBox, чтобы мы могли увидеть тестовый сайт

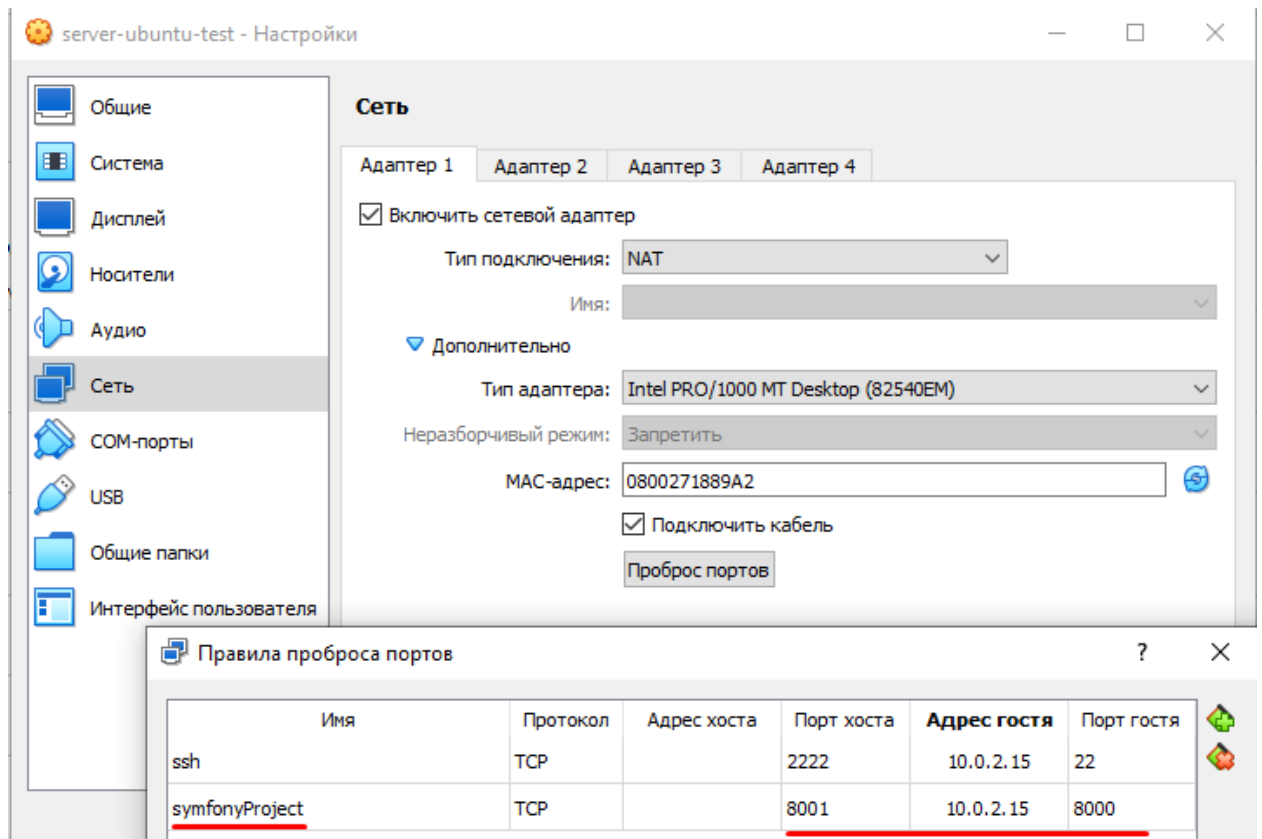


Рис 14 – Проброс портов

С помощью команды `symfony serve:start` запускаем наш сервер

```
ivan_lr@intarolr:~/demo$ symfony serve:start

[WARNING] run "symfony server:ca:install" first if you want to run the web server with TLS support, or use "--p12" or
"--no-tls" to avoid this warning

Following Web Server log file (/home/ivan_lr/.symfony5/log/a41ba87df31de2d922e74bd2c49af1eb7823eb0a.log)
Following PHP log file (/home/ivan_lr/.symfony5/log/a41ba87df31de2d922e74bd2c49af1eb7823eb0a/7daf403c7589f4927632ed3b6af762a9
WARNING the current dir requires PHP 8.1.0 (composer.json from current dir: /home/ivan_lr/demo/composer.json), but this versi

[WARNING] The local web server is optimized for local development and MUST never be used in a production setup.

[OK] Web server listening
The Web server is using PHP CLI 8.1.2
http://127.0.0.1:8000
```

Рис 15 – Запуск сервера

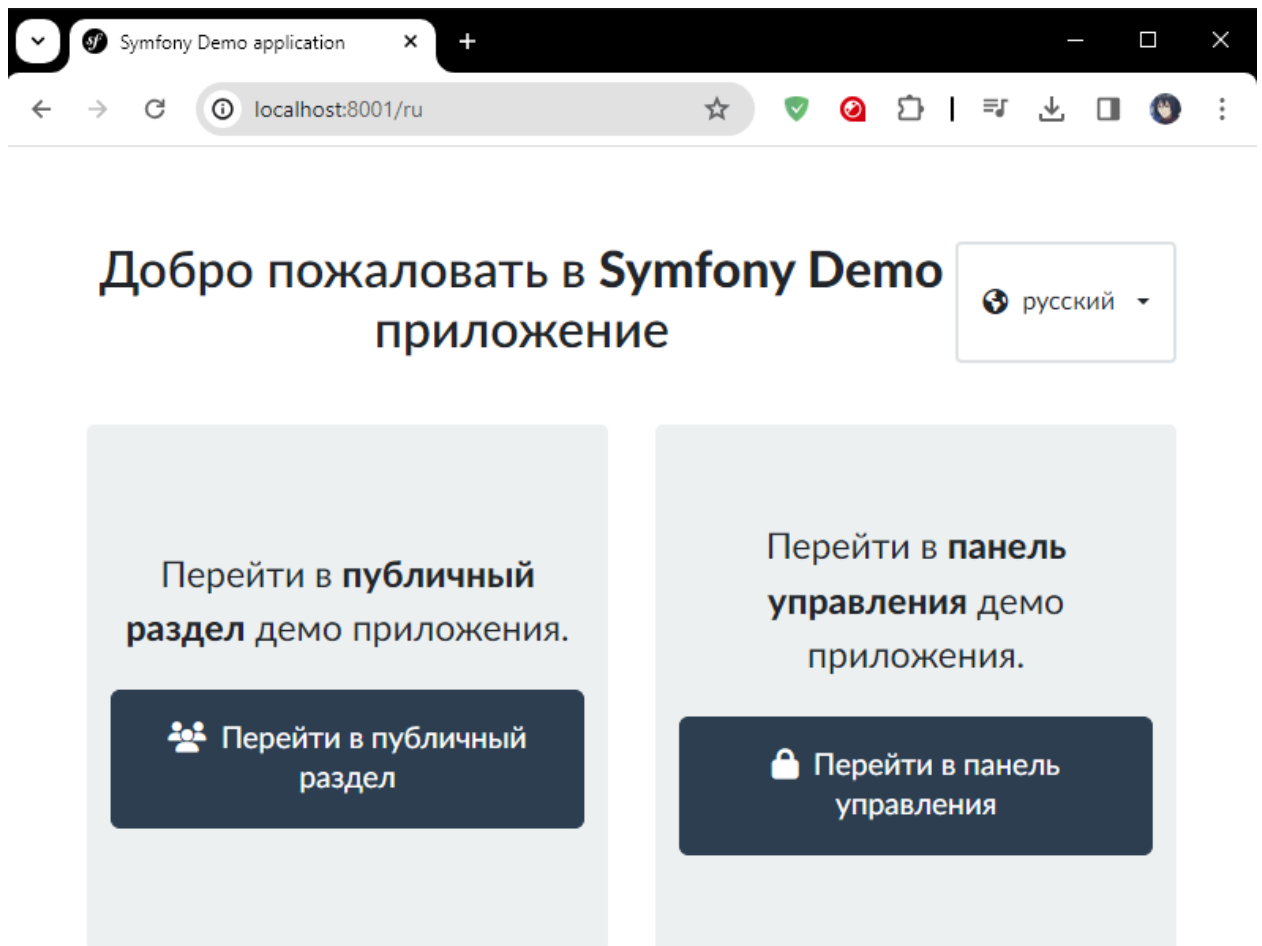


Рис 16 – Наш запущенный сайт

5) Docker и Docker-compose установили

5.1) Настройка Postgres

```
ivan_lr@intarolr:~$ sudo -i -u postgres
[sudo] password for ivan_lr:
postgres@intarolr:~$ psql
psql (14.10 (Ubuntu 14.10-0ubuntu0.22.04.1))
Type "help" for help.

postgres=# CREATE USER admin WITH PASSWORD '0000';
CREATE ROLE
postgres=# CREATE DATABASE db;
CREATE DATABASE
postgres=# /q
postgres=# \q
postgres@intarolr:~$ su - ivan_lr
Password:
```

Рис 17 – Создание БД

```
DATABASE_URL="postgres://admin:0000@localhost:5432/db?serverVersion=15&charset=utf8"
```

Рис 18 – Замена DATABASE_URL

6) Создадим файл docker-compose.yml и файл Dockerfile для php-fpm

Вместе с созданием файлов для docker создадим папки для конфига nginx и для dockerfile php-fpm, а также папку с нашим демо-проектом

```
ivan_lr@intarolr:~/LR5$ ls
app  docker-compose.yml  nginx  php
ivan_lr@intarolr:~/LR5$ |
```

Рис 19 – Структура проекта

В папке nginx – хранится конфигурация сервера

В папке php – хранится Dockerfile

В папке app – хранится наш сайт

```

version: "3.8"
services:
  nginx-service:
    image: nginx
    container_name: nginx-container
    ports:
      - "8080:80"
    volumes:
      - ./app:/var/www/project
      - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
    depends_on:
      - php82-service
      - pgsql-service
  php82-service:
    build:
      context: .
      dockerfile: ./php/Dockerfile
    container_name: php82-container
    ports:
      - "9000:9000"
    volumes:
      - ./app:/var/www/project
  pgsql-service:
    image: postgres:14
    container_name: pgsql-container
    ports:
      - "5432:5432"
    volumes:
      - /var/lib/postgresql/14/main:/var/lib/postgresql/data
    command:
      - "postgres"
      - "-c"
      - "port=5432"
    restart: always
    environment:
      - POSTGRES_USER=admin
      - POSTGRES_PASSWORD=0000

```

Рисунок 20 – файл docker-compose.yml

```

FROM php:8.2-fpm
RUN apt-get update && apt-get install -y libpq-dev zlib1g-dev g++ git libicu-dev zip libzip-dev zip \
    && docker-php-ext-install intl opcache pdo \
    && pecl install apcu \
    && docker-php-ext-enable apcu \
    && docker-php-ext-configure zip \
    && docker-php-ext-install zip \
    && docker-php-ext-install pdo pdo_pgsql pgsql
WORKDIR /var/www/project
RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer
RUN curl -sS https://get.symfony.com/cli/installer | bash
RUN mv /root/.symfony5/bin/symfony /usr/local/bin/symfony

```

Рисунок 21 – Dockerfile для php-fpm

```

server {
    listen 80;
    index index.php;
    server_name localhost;
    root /var/www/project/public;
    location / {
        try_files $uri /index.php$is_args$args;
    }
    location ~ ^/index\.php(/|$) {
        fastcgi_pass php82-service:9000;
        fastcgi_split_path_info ^(.+\.php)(/.*)$;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;
        fastcgi_param DOCUMENT_ROOT $realpath_root;
        fastcgi_buffer_size 128k;
        fastcgi_buffers 4 256k;
        fastcgi_busy_buffers_size 256k;
        internal;
    }
    location ~ \.php$ {
        return 404;
    }
    error_log /var/log/nginx/project_error.log;
    access_log /var/log/nginx/project_access.log;
}

```

Рисунок 22 – файл default.conf (конфигурация сервера)

7) Настроим DATABASE_URL и заполним бд

```

#
DATABASE_URL="postgres://admin:0000@172.20.0.2:5432/admin?serverVersion=14&charset=utf8"
# DATABASE_URL="mysql://app:!ChangeMe!@127.0.0.1:3306/app?serverVersion=8&charset=utf8mb4"

```

Рисунок 23 – Содержимое DATABASE_URL

И заполним БД с помощью команд

```
php bin/console doctrine:schema:create
```

```
php bin/console doctrine:fixtures:load
```

8) Запустим команду docker-compose up -d

```
ivan_lr@intarolr:~$ cd LR5
ivan_lr@intarolr:~/LR5$ docker-compose up -d
Starting php82-container ...
Starting php82-container ... done
Starting nginx-container ... done
ivan_lr@intarolr:~/LR5$ docker ps
```

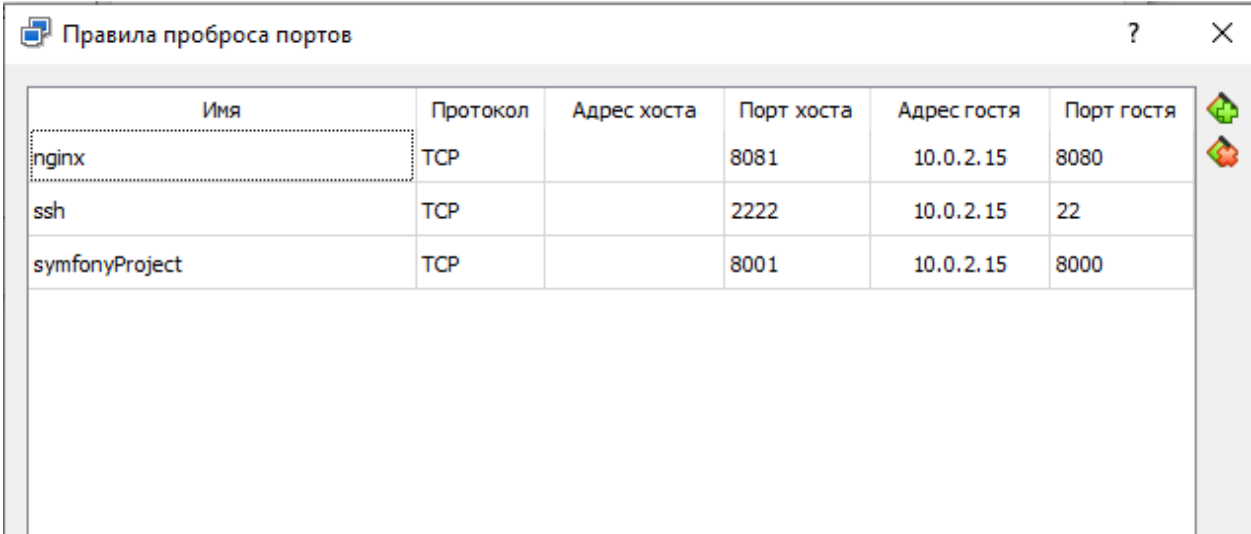
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d60274e5c572	postgres:14	"docker-entrypoint.s..."	39 minutes ago	Up 2 minutes	0.0.0.0:5432->5432/tcp, :::5432->5432/tcp	pgsql-container
ce6946ee84a1	nginx	"/docker-entrypoint..."	2 hours ago	Up 5 seconds	0.0.0.0:8080->80/tcp, :::8080->80/tcp	nginx-container
85448c7147b1	lr5_php82-service	"docker-php-entrypoi..."	2 hours ago	Up 6 seconds	0.0.0.0:9000->9000/tcp, :::9000->9000/tcp	php82-container

```
ivan_lr@intarolr:~/LR5$
```

Рисунок 24 – Запуск команды docker-compose up -d

9) Откроем наш сайт в браузере

Перед этим нужно настроить проброс портов в VirtualBox



Имя	Протокол	Адрес хоста	Порт хоста	Адрес гостя	Порт гостя
nginx	TCP		8081	10.0.2.15	8080
ssh	TCP		2222	10.0.2.15	22
symfonyProject	TCP		8001	10.0.2.15	8000

Рисунок 25 – Проброс портов в Virtualbox

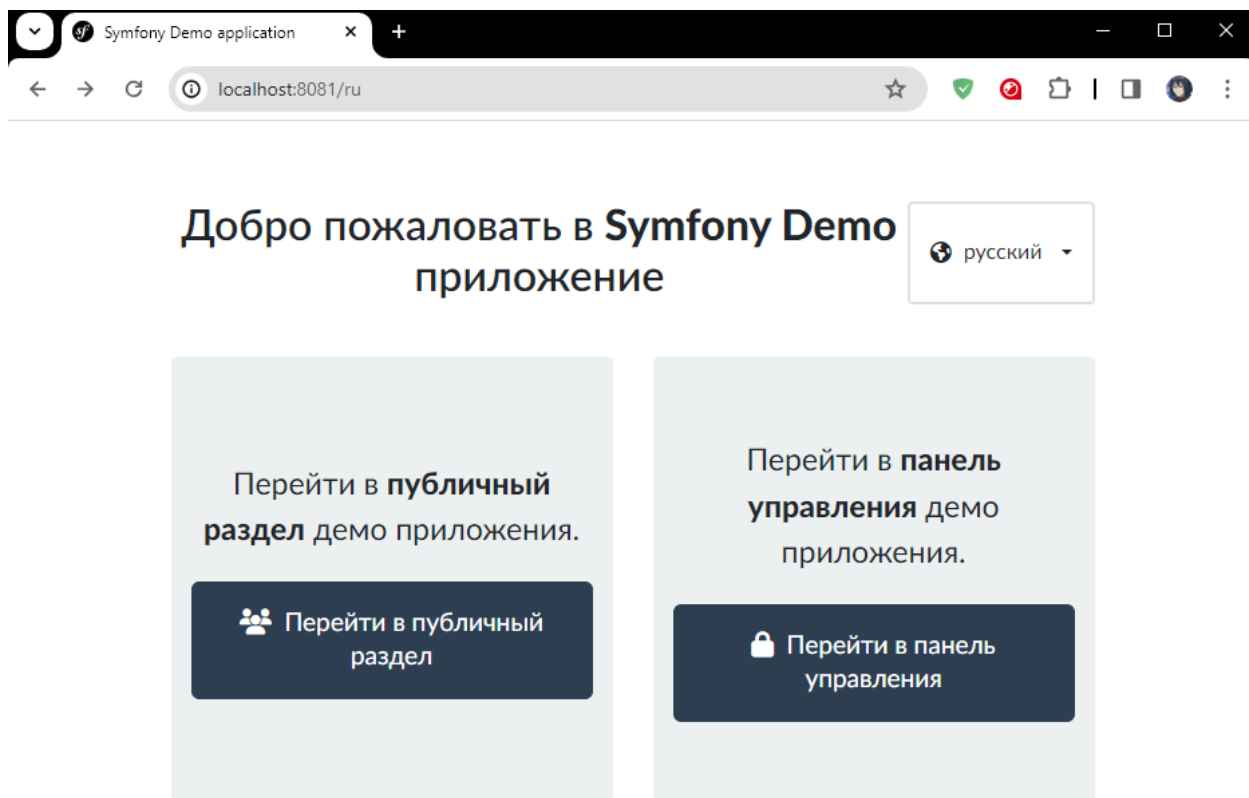


Рисунок 26 – Главная страница сайта

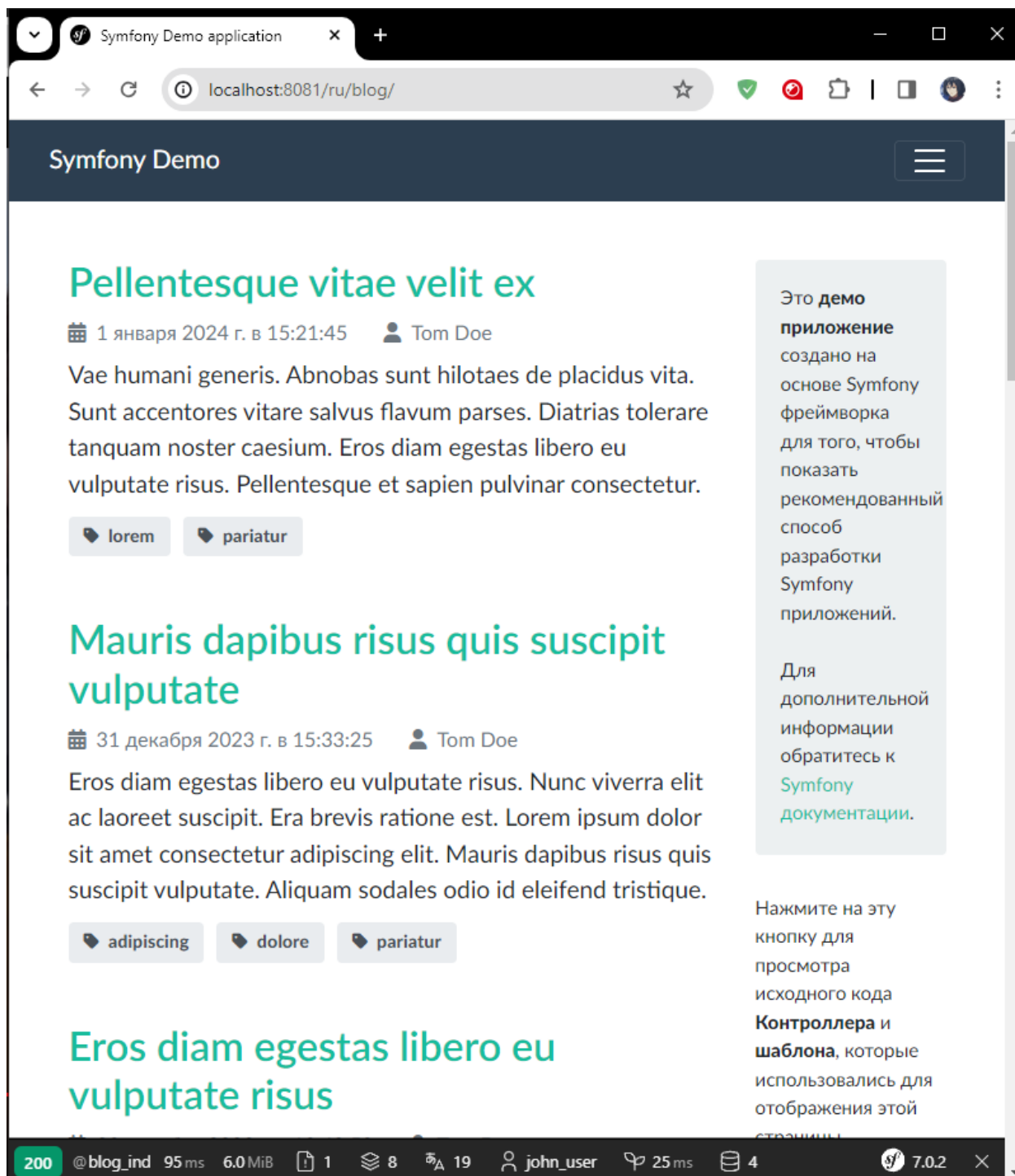


Рисунок 27 – Публичный раздел сайта (демонстрация работы postgres)

II часть

Шаг №1. Установка Nginx

Создадим docker-compose.yml

```
version: "3.8"
services:
  nginx-serv:
    image: nginx
    container_name: nginx-container
    ports:
      - "8080:80"
~
~
~
```

Рисунок 28 – Содержимое docker-compose.yml

```
ivan_lr@intarolr:~/LR5_2$ docker-compose up
Creating nginx-container ... done
Attaching to nginx-container
nginx-container | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
nginx-container | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
nginx-container | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
nginx-container | 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
nginx-container | 10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
nginx-container | /docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
nginx-container | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
nginx-container | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
nginx-container | /docker-entrypoint.sh: Configuration complete; ready for start up
nginx-container | 2024/01/02 13:38:42 [notice] 1#1: using the "epoll" event method
nginx-container | 2024/01/02 13:38:42 [notice] 1#1: nginx/1.25.3
nginx-container | 2024/01/02 13:38:42 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
nginx-container | 2024/01/02 13:38:42 [notice] 1#1: OS: Linux 5.15.0-91-generic
nginx-container | 2024/01/02 13:38:42 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
nginx-container | 2024/01/02 13:38:42 [notice] 1#1: start worker processes
nginx-container | 2024/01/02 13:38:42 [notice] 1#1: start worker process 29
nginx-container | 2024/01/02 13:38:42 [notice] 1#1: start worker process 30
```

Рисунок 29 – Запуск docker-compose up

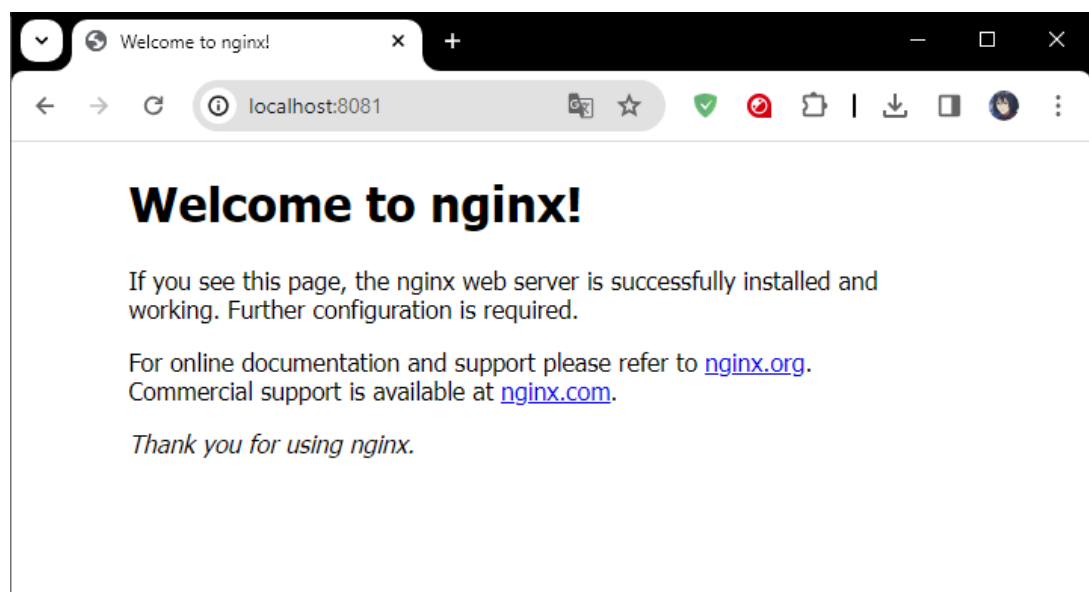


Рисунок 30 – Страничка Welcome to nginx!

Шаг №2. Передача в контейнер html-файлов

Пропишем в docker-compose.yml пути для монтирования конфига nginx и html файла.

```
version: "3.8"
services:
  nginx-serv:
    image: nginx
    container_name: nginx-container
    ports:
      - "8080:80"
    volumes:
      - ./html:/var/www/helloworld
      - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
```

Рисунок 31 – Файл docker-compose.yml

```
server {
    listen 80;
    server_name helloworld;

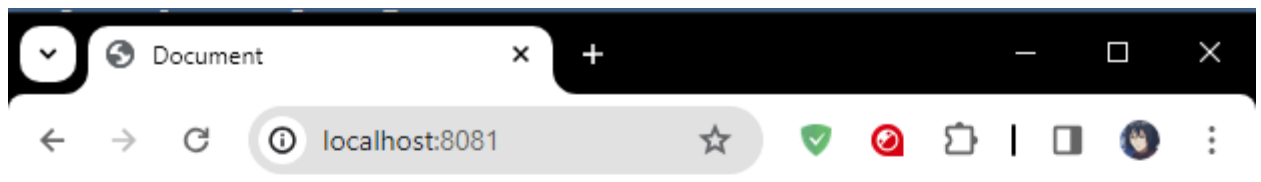
    root /var/www/helloworld;
    index index.html;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

Рисунок 32 – Файл default.conf (конфиг nginx)

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <h1>Красиков Иван</h1>
  <p>Привет, мир!!</p>
</body>
</html>
```

Рисунок 33 – Содержимое файла index.html



Красиков Иван

Привет, мир!!!

Рисунок 34 – Страница сайта

Шаг №3. Web-разработка

```
version: "3.0"
services:
  proxy:
    image: jwilder/nginx-proxy
    ports:
      - "8080:80"
    volumes:
      - /var/run/docker.sock:/tmp/docker.sock:ro
    networks:
      - proxy
networks:
  proxy:
    driver: bridge
```

Рисунок 35 – docker-compose.yml для nginx-проxy

NETWORK ID	NAME	DRIVER	SCOPE
504c75857f79	bridge	bridge	local
977bb8de7952	host	host	local
a5dd3d412884	lr5_2_default	bridge	local
9c1ac3582896	lr5_default	bridge	local
046f4b58af4a	nginx-php82-pgsql	bridge	local
542e1ab0454d	nginx-php82-pgsql-default	bridge	local
887358148d7e	none	null	local
d4d9c9a67a82	proxy_proxy	bridge	local

Рисунок 36 – docker network ls

```

version: "3"
services:
  nginx:
    image: nginx
    environment:
      - VIRTUAL_HOST=site.local
    depends_on:
      - php
    ports:
      - "8082:80"
    volumes:
      - ./html:/var/www/html/
      - ./docker/nginx/conf.d/default.conf:/etc/nginx/conf.d/default.conf
    networks:
      - frontend
      - backend
  php:
    build:
      context:
        ./docker/php
    volumes:
      - ./docker/php/php.ini:/usr/local/etc/php/php.ini
      - ./html:/var/www/html/
    networks:
      - backend
  mysql:
    image: mysql:5.7
    volumes:
      - ./docker/mysql/data:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=root
    networks:
      - backend
  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    environment:
      - VIRTUAL_HOST=phpmyadmin.local
      - PMA_HOST=mysql
      - PMA_USER=root
      - PMA_PASSWORD=root
    ports:
      - "8083:80"
    networks:
      - frontend
      - backend
networks:
  frontend:
    external:
      name: proxy_proxy
  backend:

```

Рисунок 37 – Основной docker-compose.yml

```

FROM php:8.2-fpm
RUN apt-get update && apt-get install -y \
    curl \
    wget \
    git \
    libfreetype6-dev \
    libjpeg62-turbo-dev \
    libpng-dev \
    libonig-dev \
    libzip-dev \
    libmcrypt-dev \
    && pecl install mcrypt \
    && docker-php-ext-enable mcrypt \
    && docker-php-ext-install -j$(nproc) iconv mbstring mysqli pdo_mysql zip \
    && docker-php-ext-configure gd --with-freetype --with-jpeg \
    && docker-php-ext-install -j$(nproc) gd
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer
WORKDIR /var/www/html

```

Рисунок 38 – Dockerfile для php

```

server {
    listen 80;
    server_name_in_redirect off;
    access_log /var/log/nginx/host.access.log main;
    root /var/www/html/;
    location / {
        try_files $uri /index.php$is_args$args;
    }
    location ~ /\.php$ {
        try_files $uri =404;
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        fastcgi_pass php:9000;
        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }
    location ~ /\.ht {
        deny all;
    }
}

```

Рисунок 39 – Файл default.conf

```

<?php
//phpinfo();
$link = mysqli_connect('mysql', 'root', 'root');
if (!$link) {
    die('Ошибка соединения: ' . mysqli_error());
}
echo 'Успешно соединились';
mysqli_close($link);
~

```

Рисунок 40 – Файл index.php

```

ivan_lr@intarolr:~/LR5_2$ docker-compose up -d
WARNING: Found orphan containers (nginx-container) for thi
Recreating lr5_2_phpmyadmin_1 ... done
Starting lr5_2_php_1 ... done
Starting lr5_2_mysql_1 ... done
Recreating lr5_2_nginx_1 ... done

```

Рисунок 41 – Запуск docker-compose

```

ivan_lr@intarolr:~/LR5_2$ tree -L 3
.
├── docker
│   ├── mysql
│   │   └── data
│   ├── nginx
│   │   └── conf.d
│   └── php
│       ├── Dockerfile
│       └── php.ini
├── docker-compose.yml
├── html
│   └── index.php
└── proxy
    └── docker-compose.yml

9 directories, 4 files

```

Рисунок 42 – Структура проекта

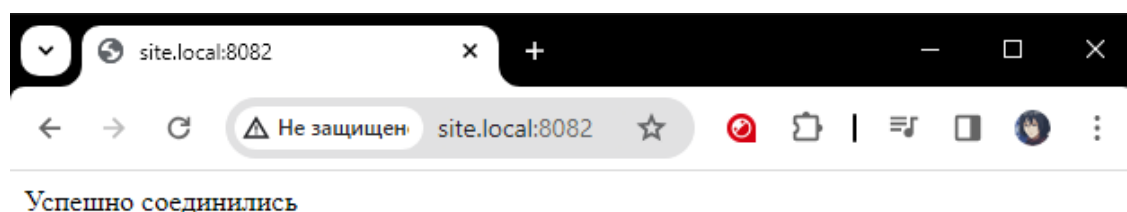


Рисунок 43 – Проверка работоспособности

Шаг №4. Запуск Wordpress.

Для запуска Wordpress на уже готовом нашем сервере, добавим папку wordpress и допишем пару моментов в docker-compose.yml, добавив туда загрузку wordpress.

```
version: "3"
services:
  nginx:
    image: nginx
    environment:
      - VIRTUAL_HOST=site.local
    depends_on:
      - php
    ports:
      - "8082:80"
    volumes:
      - ./wordpress:/var/www/html/
      - ./docker/nginx/conf.d/default.conf:/etc/nginx/conf.d/default.conf
    networks:
      - frontend
      - backend
  php:
    build:
      context:
        ./docker/php
    volumes:
      - ./docker/php/php.ini:/usr/local/etc/php/php.ini
      - ./wordpress:/var/www/html/
    networks:
      - backend
  mysql:
    image: mysql:5.7
    volumes:
      - ./docker/mysql/data:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_DATABASE=wordpress
    networks:
      - backend
  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    ports:
      - "8083:80"
    environment:
      - VIRTUAL_HOST=phpmyadmin.local
      - PMA_HOST=mysql
      - PMA_USER=root
      - MYSQL_ROOT_PASSWORD=root
    networks:
      - frontend
      - backend
  wordpress:
    depends_on:
```

```

        - mysql
image: wordpress:5.1.1-fpm-alpine
container_name: wordpress
restart: unless-stopped
env_file: .env
environment:
    - WORDPRESS_DB_HOST=mysql:3306
    - WORDPRESS_DB_USER=root
    - WORDPRESS_DB_PASSWORD=root
    - WORDPRESS_DB_NAME=wordpress
volumes:
    - ./wordpress:/var/www/html/
networks:
    - frontend
    - backend
networks:
    frontend:
        external:
            name: proxy_proxy
    backend:

```

После чего запустим docker-compose и перезагрузим наш сервер nginx

```

ivan_lr@intarolr:~/LR5_2$ docker-compose up -d
WARNING: Found orphan containers (nginx-container) for this project. If you removed or renamed this service in
lr5_2_php_1 is up-to-date
lr5_2_mysql_1 is up-to-date
Recreating lr5_2_phpmyadmin_1 ...
lr5_2_nginx_1 is up-to-date
Recreating lr5_2_phpmyadmin_1 ... done
ivan_lr@intarolr:~/LR5_2$ docker compose restart nginx
WARN[0000] network frontend: network.external.name is deprecated. Please set network.name with external: true
[+] Restarting 1/1
✓ Container lr5_2_nginx_1 Started 11.3s
ivan_lr@intarolr:~/LR5_2$ |

```

Рисунок 44 – Запуск docker-compose и перезагрузка nginx

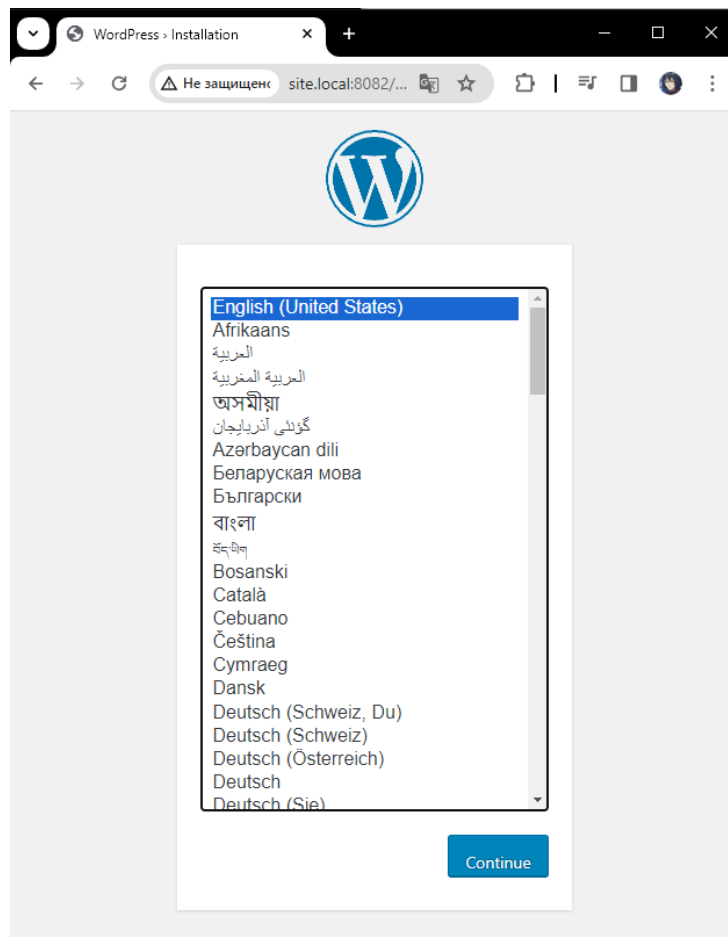


Рисунок 45 – Окно загрузки wordpress

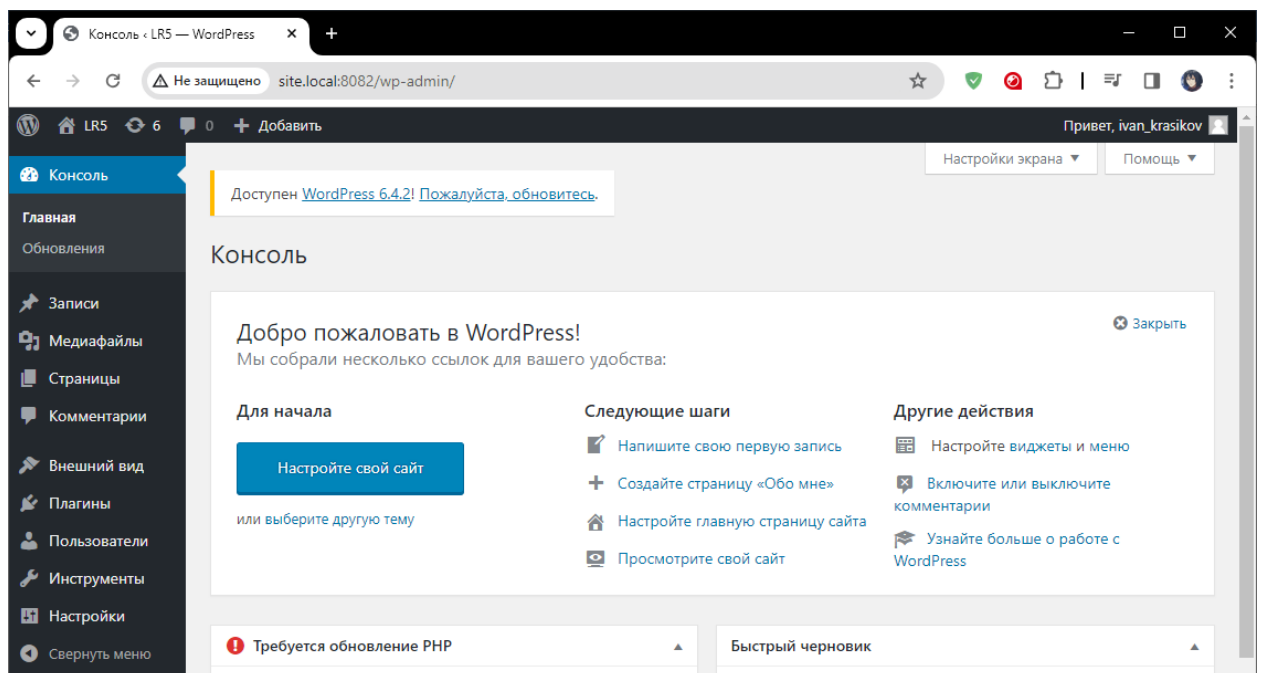


Рисунок 46 – Wordpress

Вопросы для самопроверки

1. Назовите отличия использования контейнеров по сравнению с виртуализацией:

A. Меньшие накладные расходы на инфраструктуру

C. Невозможность запуска GNU/Linux- и Windows-приложений на одном хосте

2. Назовите основные компоненты Docker:

B. Контейнеры

D. Реестры – так называются публичные и приватные хранилища докер образов, например, Docker Hub.

3. Какие технологии используются для работы с контейнерами?

A. Пространства имен (Linux Namespaces) – изоляция в виртуализации обеспечивается через ограничения со стороны процессора, а в контейнеризации это делается через пространство имен: при создании контейнера докер создает набор пространства имен для данного контейнера.

C. Контрольные группы (cgroups) – позволяют разделить доступные ресурсы железа и создавать ограничения при необходимости

4. Найдите соответствие между компонентом и его описанием:

Контейнеры – изолированные при помощи технологий операционной системы пользовательские окружения, в которых выполняются приложения.

Образы – доступные только для чтения шаблоны приложений.

Реестры (репозитории) – сетевые хранилища образов.

5. В чем отличие контейнеров от виртуализации?

Контейнеры обеспечивают легковесную виртуализацию, работая на уровне операционной системы и разделяя ее ядро с хост-системой. Они обеспечивают быстрое развертывание и экономичное использование ресурсов, так как контейнеры делят общие ресурсы и изолируют приложения на уровне пользовательского пространства. В отличие от этого, виртуальные машины полностью изолированы, имеют собственные виртуальные ресурсы и требуют больше времени на запуск, что делает их более подходящими для

приложений, где требуется высокая степень изоляции. Контейнеры облегчают масштабирование и управление приложениями, в то время как виртуальные машины предоставляют более высокий уровень изоляции, но могут быть более тяжеловесными в использовании ресурсов.

6. Перечислите основные команды утилиты Docker с их кратким описанием

`run <название образа>` – запуск контейнера из образа

- `-it` – позволяет войти в контейнер, чтобы работать изнутри. Например, в контейнер с Ubuntu мы можем работать на `bash`.
- `--name` – позволяет задать имя контейнеру
- `-v` или `--mount` – указывает для контейнера папки монтирования
- `-p` – позволяет пробросить порты из контейнера на хост
- `-d` – запуск контейнера в фоновом режиме
- `-e` – устанавливает переменную окружения
- `--link` – позволяет связать контейнер с другим

`pull <название образа>` – скачивание образа из реестра

`stop` – остановка контейнера

`ps` – просмотр списка работающих контейнеров

- `-a` – весь список контейнеров (запущенных и нет)

`rm` – удаляет остановленный или жестко завершает контейнер

`images` – просмотр списка образов на хосте

`rmi` – удаляет образ (перед удалением обязательно остановить все контейнеры этого образа)

`exec <контейнер> <команда>` – позволяет выполнить команду на работающем контейнере

`build` – сборка образа на основе файлов

`push` – позволяет отправить образ в реестр (нужны имя и адрес)

`inspect` – просмотр полной информации о контейнере

`network ls` – просмотр сетей на хосте

`compose up` – запуск мультиконтейнерного приложения

- `-d` – запуск в фоновом режиме
- `--scale <name>=<num>` – создает кол-во реплик контейнера 29

compose build – сборка мультиконтейнерного приложения на основе файла docker-compose.yml

7. Каким образом осуществляется поиск образов контейнеров?

Сначала докер ищет указанный образ на хосте, и если его не находит, то идет на доступные реестры и скачивает их оттуда. Реестр может быть частным, или запрашиваемый образ может быть закрытым – для доступа к таким образам нужно авторизоваться – команда `docker login`.

8. Каким образом осуществляется запуск контейнера?

Контейнер запускается на основе образа. Образ может быть скачен из реестра или создан на хосте. Созданные на хосте образы обычно включает в себя инструкцию FROM, указывающую на основе какого образа осуществляется создание текущего. В образе указаны все необходимые зависимости и другие инструкции. При запуске контейнера из образа на хосте в первый раз его сначала нужно собрать командой `docker build`. Сборка осуществляется по слоям, каждый слой имеет свой размер. Если осуществляется пересборка образа, то идентичные инструкции будут взяты из кеша.

9. Что значит управлять состоянием контейнеров?

Управление состоянием контейнеров включает в себя контроль за запуском, остановкой, мониторингом и обновлением контейнеризованных приложений. Этот процесс позволяет эффективно использовать ресурсы, обеспечивает стабильность работы приложения и предоставляет возможность динамической конфигурации, такой как изменение переменных среды и настроек сети. Управление состоянием также включает в себя миграцию контейнеров между хостами и масштабирование приложения путем добавления или удаления экземпляров контейнеров в зависимости от потребностей. Общее управление состоянием обеспечивает гибкость и надежность в эксплуатации контейнеризованных сред.

10. Как изолировать контейнер?

Контейнеры изолированы по умолчанию, но есть возможность отключить все механизмы изоляции Docker – в таком случае запуск 30 приложения в контейнере не будет отличаться от запуска на хосте. Две основы изоляции – это Linux namespace и cgroup.

11. Опишите последовательность создания новых образов, назначение Dockerfile?

Создание новых образов в Docker включает в себя использование файла, называемого Dockerfile. Dockerfile представляет собой текстовый файл, в котором определяются инструкции и шаги для построения контейнера. В начале Dockerfile обычно указывается базовый образ, на основе которого будет строиться новый контейнер. Затем добавляются инструкции для установки зависимостей, копирования файлов приложения, настройки переменных среды и других параметров. Каждая инструкция Dockerfile создает новый слой образа, оптимизируя кэширование и упрощая процесс обновления. В конечном итоге, Dockerfile служит для автоматизации процесса сборки контейнера, делая его воспроизводимым и легко управляемым.

12. Возможно ли работать с контейнерами Docker без одноименного движка?

Да. Например, облачные сервисы: Fly.io, Stackpath, Deno.land, Vercel.app.

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

Система оркестрации контейнеров (не обязательно Kubernetes) нужны для управления контейнерами. Она позволяет их создавать (планировать время создания тоже), масштабировать, распределять между несколькими хостами, выделять ресурсы, следить за статусами контейнеров, перезапускать и многое другое. Kubernetes – одна из систем оркестрации, но очень хорошая. Kubernetes не зависит от языка программирования, платформы и операционной системы, он предлагает широкий спектр вариантов развертывания. Кроме того, он предоставляет множество разных удобных функций, которых нет, например, в Docker Swarm – бесплатной встроенной в докер системе оркестрации. Основные объекты Kubernetes:

- Кластеры: пул для вычислений, хранения и сетевых ресурсов.
- Ноды: хост-машины, работающие в кластере.

- Пространства имен: логические разделы кластера.
- Поды: единицы развертывания.
- Метки и селекторы: пары «ключ-значение» для идентификации и обнаружения сервисов.
- Сервисы: коллекция подов, принадлежащих одному и тому же приложению.
- Набор реплик: обеспечивает доступность и масштабируемость.
- Развертывание: управляет жизненным циклом приложения.