

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**



**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Институт

компьютерных наук

Кафедра

автоматизированных систем управления

**ЛАБОРАТОРНАЯ РАБОТА №5**

По дисциплине "Операционные системы Linux"

На тему "Контейнеризация"

Студент

ПИ-22-1

подпись, дата

Клименко Н.Д.

Руководитель

канд.техн.наук, доцент

ученая степень, ученое звание

подпись, дата

Кургасов В.В.

Липецк, 2024 г.

## **Оглавление**

<b>Цель работы.....</b>	<b>3</b>
<b>Ход работы.....</b>	<b>4</b>
<b>Часть I.....</b>	<b>4</b>
<b>Часть II.....</b>	<b>14</b>
<b>Шаг 1. Установка Nginx.....</b>	<b>14</b>
<b>Шаг 2. Передача в контейнер html-файлов.....</b>	<b>14</b>
<b>Шаг 3. Web-разработка.....</b>	<b>15</b>
<b>Шаг 4. WordPress .....</b>	<b>19</b>
<b>Вывод.....</b>	<b>22</b>
<b>Контрольные вопросы .....</b>	<b>23</b>

## **Цель работы**

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

## Ход работы

### Часть I

Командой `sudo apt install git` установим git для клонирования проекта. Также командой `sudo apt install php` установим PHP для запуска Symfony.

Клонируем проект на сервер с помощью команды `"git clone https://github.com/symfony/demo"` для последующей работы. На рисунке 1 представлено клонирование проекта.

```
nikita@deb-server:~$ git clone https://github.com/symfony/demo
Клонирование в «demo»...
remote: Enumerating objects: 12611, done.
remote: Counting objects: 100% (164/164), done.
remote: Compressing objects: 100% (107/107), done.
remote: Total 12611 (delta 54), reused 126 (delta 49), pack-reused 12447 (from 1)
Получение объектов: 100% (12611/12611), 21.99 МБ | 824.00 КБ/с, готово.
Определение изменений: 100% (7514/7514), готово.
nikita@deb-server:~$ ls
demo
nikita@deb-server:~$
nikita@deb-server:~$ cd demo
nikita@deb-server:~/demo$
```

Рисунок 1 – Клонирование проекта

Командой `php bin/console server:start` пробуем запустить проект. Результат представлен на рисунке 2.

```
nikita@deb-server:~$ cd demo
nikita@deb-server:~/demo$ php bin/console server:start
PHP Fatal error:  Uncaught LogicException: Dependencies are missing. Try running "composer install". in /home/nikita/demo/bin/console:8
Stack trace:
#0 {main}
  thrown in /home/nikita/demo/bin/console on line 8
nikita@deb-server:~/demo$
```

Рисунок 2 – Попытка запуска проекта

На рисунке 2 приставлена ошибка запуска, которая возникает из-за отсутствия зависимостей. Требуется установка "Composer", который в свою очередь является инструментом для управления зависимостями в PHP-проектах. Он автоматически загружает и устанавливает библиотеки, указанные в "composer.json".

Командой `sudo apt install composer` производим установку Composer. Результат установки (выполнена команда `composer | less`) представлен на рисунке 3.

```
Composer version 2.5.5 2023-03-21 11:50:05

Usage:
  command [options] [arguments]

Options:
  -h, --help                Display help for the given command. When no command is given display help for the list command
  -q, --quiet               Do not output any message
  -V, --version             Display this application version
  --ansi|--no-ansi         Force (or disable --no-ansi) ANSI output
  -n, --no-interaction      Do not ask any interactive question
  --profile                Display timing and memory usage information
  --no-plugins              Whether to disable plugins.
  --no-scripts             Skips the execution of all scripts defined in composer.json file.
  -d, --working-dir=WORKING-DIR If specified, use the given directory as working directory.
  --no-cache               Prevent use of the cache
  -v|vv|vvv, --verbose     Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
```

Рисунок 3 - Composer

На рисунке 3 представлен composer версии 2.5.5. Обновим его версию. Для этого воспользуемся командой `sudo apt install curl` (утилита для выполнения HTTP-запросов. Она позволяет скачивать файлы или данные с веб-серверов). Теперь уже с помощью утилиты `curl` получаем скрипт установки и сверяем его хэш с указанным на официальном сайте (<https://composer.github.io/pubkeys.html>). Используемые команды:

- `curl -sS https://getcomposer.org/installer -o composer-setup.php`
- `sha384sum composer-setup.php`

Процесс скачивания скрипта и проверки его хэша продемонстрирован на рисунке 4.

```
nikita@deb-server:~$ curl -sS https://getcomposer.org/installer -o composer-setup.php
nikita@deb-server:~$ sha384sum composer-setup.php
dac665fdc30fdd8ec78b38b9800061b4150413ff2e3b6f88543c636f7cd84f6db9189d43a81e5503cda447da73c7e5b6  composer-setup.php
nikita@deb-server:~$
```

Рисунок 4 – Проверка хеш-суммы

После того, как убедились, что хэш-сумма идентична, выполним установочный скрипт:

- `sudo php composer-setup.php --install-dir=/usr/local/bin --filename=composer`

Он загрузит последнюю версию Composer с официального репозитория и настроит его для глобального использования в системе, размещая исполняемый файл в директории `/usr/local/bin`, что в свою очередь позволит вызвать Composer из любой точки командой строки без дополнительной настройки путей. На рисунке 5 можно наблюдать успешное выполнение команды. Composer обновился до версии 2.8.3.

```

nikita@deb-server:~$ sudo php composer-setup.php --install-dir=/usr/local/bin --filename=composer
All settings correct for using Composer
Downloading...

Composer (version 2.8.3) successfully installed to: /usr/local/bin/composer
Use it: php /usr/local/bin/composer

nikita@deb-server:~$ _

```

Рисунок 5 – Обновление Composer

Теперь можем воспользоваться командой `composer install`, чтобы установить все зависимости проекта, указанные в файле `composer.json`. На рисунке 6 продемонстрирована ошибка, которая возникает в следствии отсутствия недостающих модулей `php`, а именно `php-xml` (для работы с `xml` файлами) и `php-sqlite3` (для использования в проекте `sqlite`).

```

Your requirements could not be resolved to an installable set of packages.

Problem 1
- Root composer.json requires PHP extension ext-pdo_sqlite * but it is missing from your system. Install or enable PHP's pdo_sqlite extension.
Problem 2
- Root composer.json requires symfony/html-sanitizer ^7 -> satisfiable by symfony/html-sanitizer[v7.0.0-BETA1, ..., 7.3.x-dev].
- symfony/html-sanitizer[v7.0.0-BETA1, ..., 7.3.x-dev] require ext-dom * -> it is missing from your system. Install or enable PHP's dom extension.
Problem 3
- Root composer.json requires symfony/debug-bundle ^7 -> satisfiable by symfony/debug-bundle[v7.0.0-BETA1, ..., 7.3.x-dev].
- symfony/debug-bundle[v7.0.0-BETA1, ..., 7.3.x-dev] require ext-xml * -> it is missing from your system. Install or enable PHP's xml extension.

Alternatively you can require one of these packages that provide the extension (or parts of it):
- nphre/php-dbus DBUS bindings for PHP language

Problem 4
- Root composer.json requires symfony/framework-bundle ^7 -> satisfiable by symfony/framework-bundle[v7.0.0-BETA1, ..., 7.3.x-dev].
- symfony/framework-bundle[v7.0.0-BETA1, ..., 7.3.x-dev] require ext-xml * -> it is missing from your system. Install or enable PHP's xml extension.

Alternatively you can require one of these packages that provide the extension (or parts of it):
- nphre/php-dbus DBUS bindings for PHP language

To enable extensions, verify that they are enabled in your .ini files:
- /etc/php/8.2/cli/php.ini
- /etc/php/8.2/cli/conf.d/10-opcache.ini
- /etc/php/8.2/cli/conf.d/10-pdo.ini
- /etc/php/8.2/cli/conf.d/20-calendar.ini
- /etc/php/8.2/cli/conf.d/20-ctype.ini
- /etc/php/8.2/cli/conf.d/20-curl.ini
- /etc/php/8.2/cli/conf.d/20-exif.ini
- /etc/php/8.2/cli/conf.d/20-ffi.ini
- /etc/php/8.2/cli/conf.d/20-fileinfo.ini
- /etc/php/8.2/cli/conf.d/20-ftp.ini
- /etc/php/8.2/cli/conf.d/20-gd.ini
- /etc/php/8.2/cli/conf.d/20-gettext.ini
- /etc/php/8.2/cli/conf.d/20-iconv.ini
- /etc/php/8.2/cli/conf.d/20-intl.ini
- /etc/php/8.2/cli/conf.d/20-mbstring.ini
- /etc/php/8.2/cli/conf.d/20-phar.ini
- /etc/php/8.2/cli/conf.d/20-posix.ini
- /etc/php/8.2/cli/conf.d/20-readline.ini
- /etc/php/8.2/cli/conf.d/20-shmop.ini
- /etc/php/8.2/cli/conf.d/20-sockets.ini
- /etc/php/8.2/cli/conf.d/20-sysvmsg.ini
- /etc/php/8.2/cli/conf.d/20-sysvsem.ini
- /etc/php/8.2/cli/conf.d/20-sysvshm.ini
- /etc/php/8.2/cli/conf.d/20-tokenizer.ini

You can also run `php --ini` in a terminal to see which files are used by PHP in CLI mode.
Alternatively, you can run Composer with `--ignore-platform-req=ext-pdo_sqlite --ignore-platform-req=ext-dom --ignore-platform-req=ext-xml` to temporarily ignore these required extensions.
nikita@deb-server:~/demo$

```

Рисунок 6 – Отсутствие php модулей

Установим недостающие модули следующими командами:

- `sudo apt install php-xml`
- `sudo apt install php-sqlite3`

Так как присутствовали проблемы при установке зависимостей, воспользуемся командой `composer clear-cache` для очистки кеша `composer`. После чего

выполним команду `sudo composer install`. На рисунке 7 продемонстрировано завершение успешной установки зависимостей.

```
Symfony operations: 3 recipes (1f4aa347b116e303c0e6dd5300c5e173)
- Configuring symfony/framework-bundle (>=7.2): From github.com/symfony/recipes:main
- Configuring symfony/form (>=7.2): From github.com/symfony/recipes:main
- Configuring symfony/ux-icons (>=2.17): From github.com/symfony/recipes:main
Executing script cache:clear [OK]
Executing script assets:install public [OK]
Executing script importmap:install [OK]
Executing script sass:build [OK]

What's next?

Some files have been created and/or updated to configure your new packages.
Please review, edit and commit them: these files are yours.

symfony/framework-bundle instructions:

* Run your application:
  1. Go to the project directory
  2. Create your code repository with the git init command
  3. Download the Symfony CLI at https://symfony.com/download to install a development web server

* Read the documentation at https://symfony.com/doc

nikita@deb-server:~/demo$ _
```

Рисунок 7 – Успешное выполнение `composer install`

Снова пробуем выполнить запуск сервера – `php bin/console server:start`, но сталкиваемся с очередной ошибкой, представленной на рисунке 8.

```
nikita@deb-server:~/demo$ php bin/console server:start

Command "server:start" is not defined.

Did you mean one of these?
    server:dump
    server:log

nikita@deb-server:~/demo$ _
```

Рисунок 8 – Ошибка запуска сервера

Причиной ошибки является команда `server:start`, которая устарела или отсутствует в текущей версии Symfony. Для решения этой проблемы необходимо произвести установку Symfony CLI, который предлагает современный подход запуска сервера.

Сперва загрузим и выполним установочный скрипт для настройки репозитория Symfony. Скрипт автоматически добавит репозиторий в список источников пакетов и обновит информацию о доступных пакетах на системе:

```
- curl -sLf 'https://dl.cloudsmith.io/public/symfony/stable/setup.deb.sh' |  
sudo -E bash
```

После произведем установку пакета из только что добавленного репозитория:

```
- sudo apt install symfony-cli
```

Чтобы сервер был доступен из вне виртуальной машины необходимо реализовать проброс портов. В разделе "Устройства"->"Сеть"->"Настроить сеть"->"Адаптер 1"->"Проброс портов" указываем порт хоста и порт гостя равный 8000. И можем воспользоваться командой `symphony --listen-ip=0.0.0.0 serve`, благодаря чему наш сервер будет доступен по адресу <http://127.0.0.1:8000>.

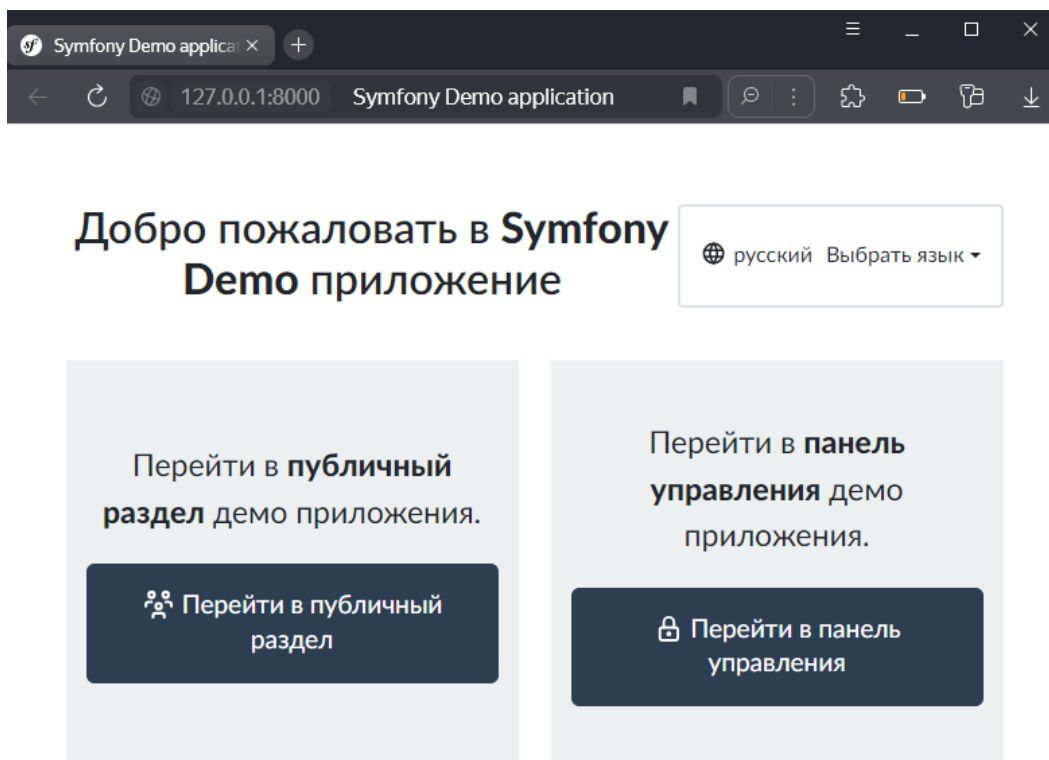


Рисунок 9 – Страница проекта

Перейдем к части с докером. Для установки Docker воспользуемся официальной инструкцией с сайта <https://docs.docker.com/engine/install/debian/>. Последовательно выполним следующие команды:

```
- sudo apt-get update  
- sudo apt-get install ca-certificates curl  
- sudo install -m 0755 -d /etc/apt/keyrings
```



```

- sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o /etc/apt/key-
rings/docker.asc

- sudo chmod a+r /etc/apt/keyrings/docker.asc

- echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/key-
rings/docker.asc] https://download.docker.com/linux/debian $(. /etc/os-release &&
echo "$VERSION_CODENAME") stable" | sudo tee /etc/apt/sources.list.d/docker.list
> /dev/null

- sudo apt-get update

```

Теперь подготовим проект к работе с Docker. Удалим запись pdo\_sqlite из файла composer.json. Это необходимо, так как в проекте используется PostgreSQL и использование SQLite становится избыточным. Также данная процедура поможет избежать возможных конфликтов зависимостей и упростит настройку базы данных в Docker-контейнере. Нужная строка для удаления продемонстрирована на рисунке 10.

```

GNU nano 7.2
{
  "name": "symfony/symfony-demo",
  "license": "MIT",
  "type": "project",
  "description": "Symfony Demo Application",
  "minimum-stability": "dev",
  "prefer-stable": true,
  "replace": {
    "symfony/polyfill-php72": "*",
    "symfony/polyfill-php73": "*",
    "symfony/polyfill-php74": "*",
    "symfony/polyfill-php80": "*",
    "symfony/polyfill-php81": "*",
    "symfony/polyfill-php82": "*"
  },
  "require": {
    "php": ">=8.2",
    "ext-pdo_sqlite": "*",
    "doctrine/dbal": "^4.0",
    "doctrine/doctrine-bundle": "^2.11",
    "doctrine/doctrine-migrations-bundle": "^3.3",
    "doctrine/orm": "^3.0",
    "league/commonmark": "^2.1",
    "symfony/apache-pack": "^1.0",
    "symfony/asset": "^7",
  }
}

```

Рисунок 10 – Редактирование composer.json

Затем с помощью команды `sudo apt install postgresql` установим PostgreSQL для работы с проектом. После установки, чтобы избежать конфликтов портов

между локальной базой данных и контейнером Docker, останавливаем PostgreSQL перед запуском контейнера с помощью команды `sudo systemctl stop postgresql`. Данный процесс продемонстрирован на рисунке 11.

```
nikita@deb-server:~/demo$ sudo systemctl stop postgresql
nikita@deb-server:~/demo$ sudo systemctl status postgresql
* postgresql.service - PostgreSQL RDBMS
   Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; preset: enabled)
   Active: inactive (dead) since Thu 2024-12-12 09:03:53 MSK; 3s ago
     Duration: 1min 47.573s
    Main PID: 3556 (code=exited, status=0/SUCCESS)
       CPU: 6ms

дек 12 09:02:06 deb-server systemd[1]: Starting postgresql.service - PostgreSQL RDBMS...
дек 12 09:02:06 deb-server systemd[1]: Finished postgresql.service - PostgreSQL RDBMS.
дек 12 09:03:53 deb-server systemd[1]: postgresql.service: Deactivated successfully.
дек 12 09:03:53 deb-server systemd[1]: Stopped postgresql.service - PostgreSQL RDBMS.
nikita@deb-server:~/demo$ _
```

Рисунок 11 – Остановка PostgreSQL

Перейдем к редактированию файла конфигураций переменных окружения проекта. Откроем данный файл (`nano .env`) и добавим следующую строку:

- `DATABASE_URL="postgresql://postgres:postgres@postgres:5432/app?serverVersion=17.1&charset=utf8"`

Переменная `DATABASE_URL` указывает параметры подключения сервиса к базе данных. В данном случае мы указываем подключение к PostgreSQL, который работает в Docker-контейнере. Остальные переменные, отвечающие за подключение к базе данных, необходимо закомментировать. Содержимое отредактированного файла представлено на рисунке 12.

```
GNU nano 7.2                                .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.$APP_ENV     committed environment-specific defaults
# * .env.$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 >=1.2).
# https://symfony.com/doc/current/best_practices.html#use-environment-variables-for-infrastructure-configuration

###> symfony/framework-bundle ###
APP_ENV=dev
APP_SECRET=
###< 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
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
#
# DATABASE_URL=sqlite:///kernel.project_dir%/data/database.sqlite
# DATABASE_URL="mysql://app:ICchangeMe!@127.0.0.1:3306/app?serverVersion=8&charset=utf8mb4"
# DATABASE_URL="postgresql://app:ICchangeMe!@127.0.0.1:5432/app?serverVersion=16&charset=utf8"
DATABASE_URL="postgresql://postgres:postgres@postgres:5432/app?serverVersion=17.1&charset=utf8"
###< doctrine/doctrine-bundle ###

###> symfony/mailer ###
# MAILER_DSN=null://null
###< symfony/mailer ###
```

Рисунок 12 – Редактирование файла конфигураций переменных окружения

В корневой папке проекта создаем Dockerfile, который в свою очередь будет содержать инструкции для создания среды, в которой будет работать сервис. Его содержимое представлено на рисунке 13.

```
GNU nano 7.2
FROM wyveo/nginx-php-fpm:php82
WORKDIR /var/www/html/demo
COPY composer.json ./
COPY . .
RUN \
    wget https://get.symfony.com/cli/installer -O - | bash \
    && mv /root/.symfony5/bin/symfony /usr/local/bin/symfony \
    && symfony composer install
EXPOSE 8000
CMD ["symfony", "--listen-ip=0.0.0.0", "serve"]
```

Рисунок 13 – Содержимое Dockerfile

Также для управления контейнерами и описания взаимодействия сервисов необходимо создать файл docker-compose.yml. Его содержимое представлено на рисунке 14.

```
GNU nano 7.2
version: "3"
services:
  app:
    container_name: docker-node-mongo
    restart: always
    build: .
    ports:
      - "81:8000"
    links:
      - postgres
  postgres:
    container_name: postgres
    image: postgres
    ports:
      - "5432:5432"
    environment:
      POSTGRES_PASSWORD: postgres
    volumes:
      - pgdata:/var/lib/postgresql/data
volumes:
  pgdata:
```

Рисунок 14 – Содержимое docker-compose.yml

Для корректной работы PostgreSQL была создана директория pgdata.

В разделе "Устройства"->"Сеть"->"Настроить сеть"->"Адаптер 1"->"Проброс портов" указываем порт хоста и порт гостя равный 81.

Собираем и запускаем контейнеры с помощью команды `sudo docker compose up --build`. После успешного запуска Docker выполняем следующие команды в новом терминале для инициализации базы данных:

- `docker ps -a`

- sudo docker exec -it "id" bash
- php bin/console doctrine:database:create
- php bin/console doctrine:schema:create
- php bin/console doctrine:fixtures:load

Теперь переходим по адресу <http://127.0.0.1:81> и тестируем работу сайта.

Главная страница представлена на рисунке 15. Работа контрольной панели представлена на рисунке 16. Пробуем зайти в учетную запись администратора. На рисунке 17 представлена успешная авторизация учетной записи администратора. Из чего можно сделать вывод, что сервис и база данных в контейнере работают корректно.

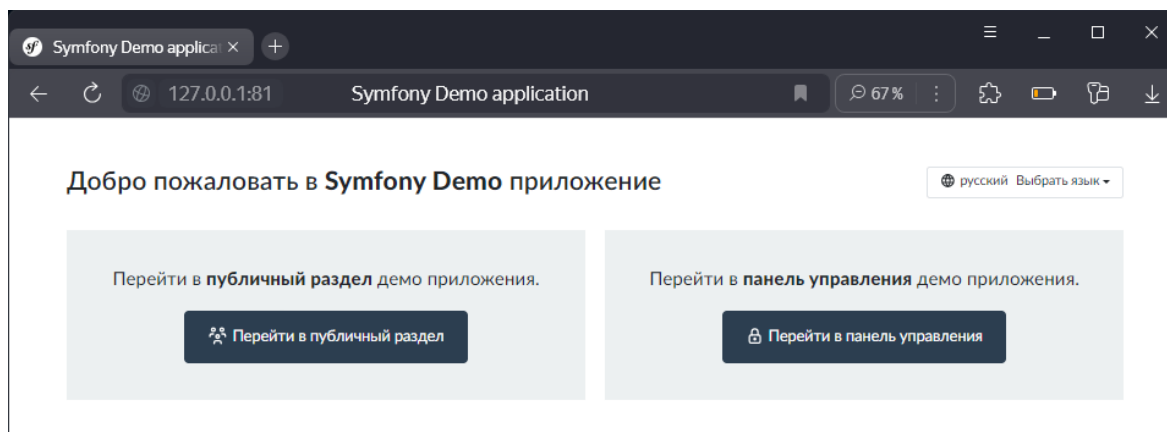


Рисунок 15 – Главная страница

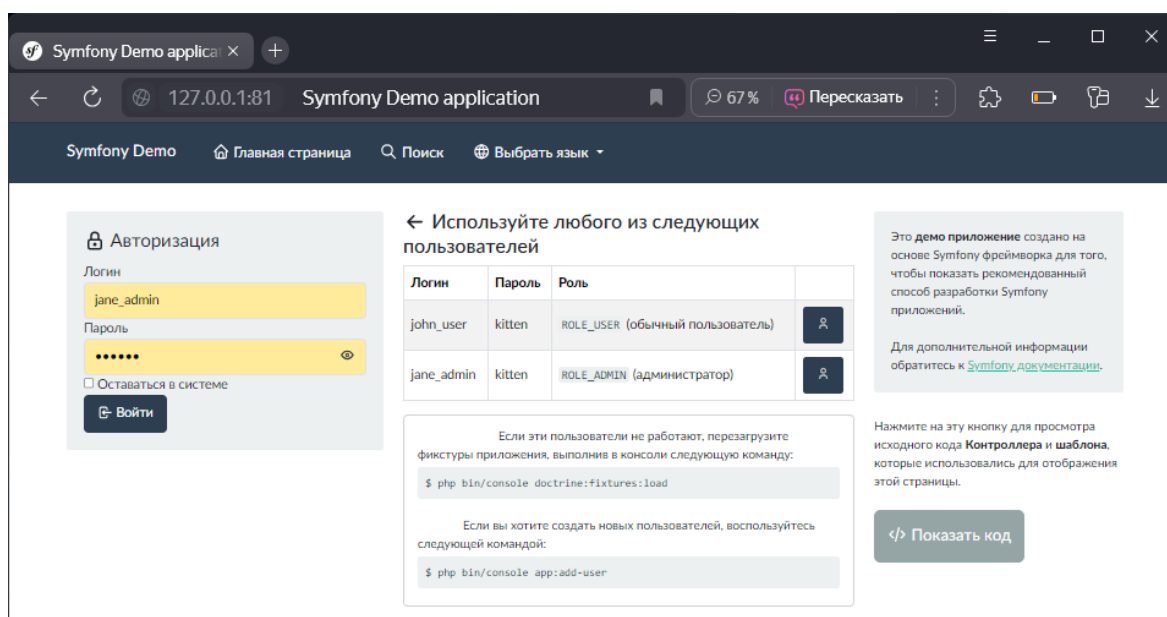


Рисунок 16 – Контрольная панель

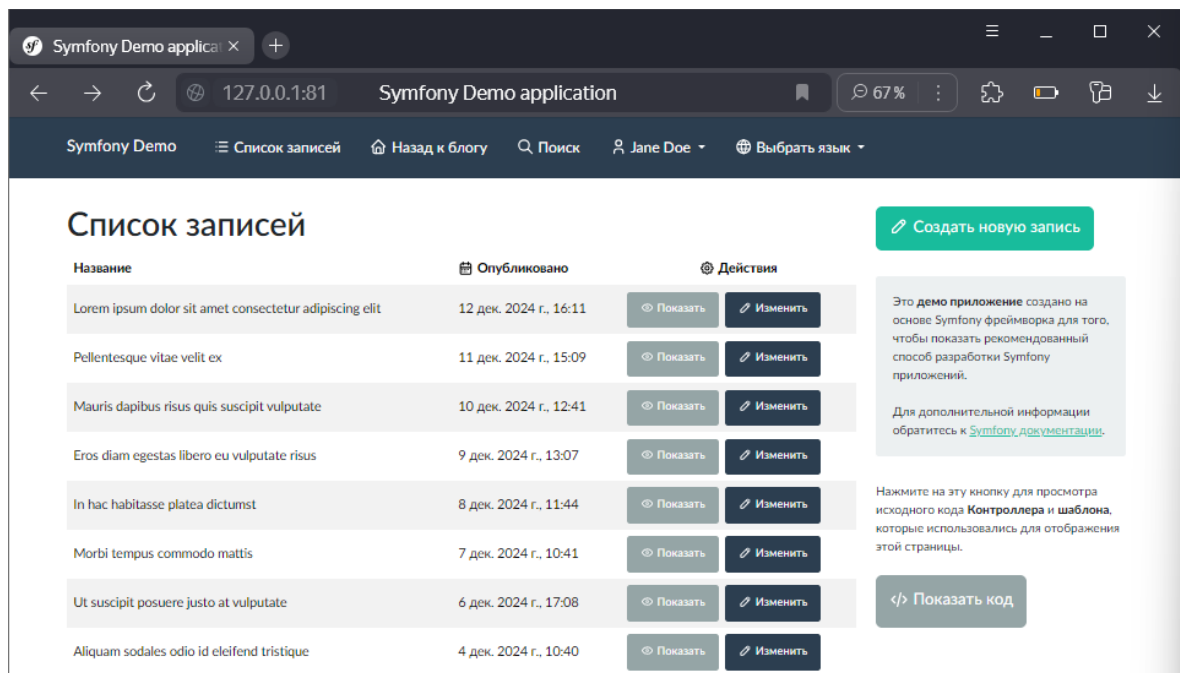
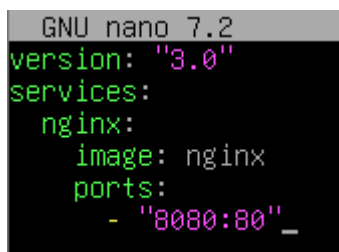


Рисунок 17 – Учетная запись администратора

## Часть II

### Шаг 1. Установка Nginx

Для данной части задания была создана новая директория для проекта "my-demo". В корневой папке проекта создаем файл `docker-compose.yml`, его содержимое представлено на рисунке 18



```
GNU nano 7.2
version: "3.0"
services:
  nginx:
    image: nginx
    ports:
      - "8080:80"
```

Рисунок 18 – Содержимое `docker-compose.yml`

В разделе "Устройства"-"Сеть"-"Настроить сеть"-"Адаптер 1"-"Проброс портов" указываем порт хоста и порт гостя равный 81. С помощью команды `sudo compose up` разворачиваем контейнер. Переходим по адресу <http://localhost:8080>. Приветственная страница Nginx представлена на рисунке 19.

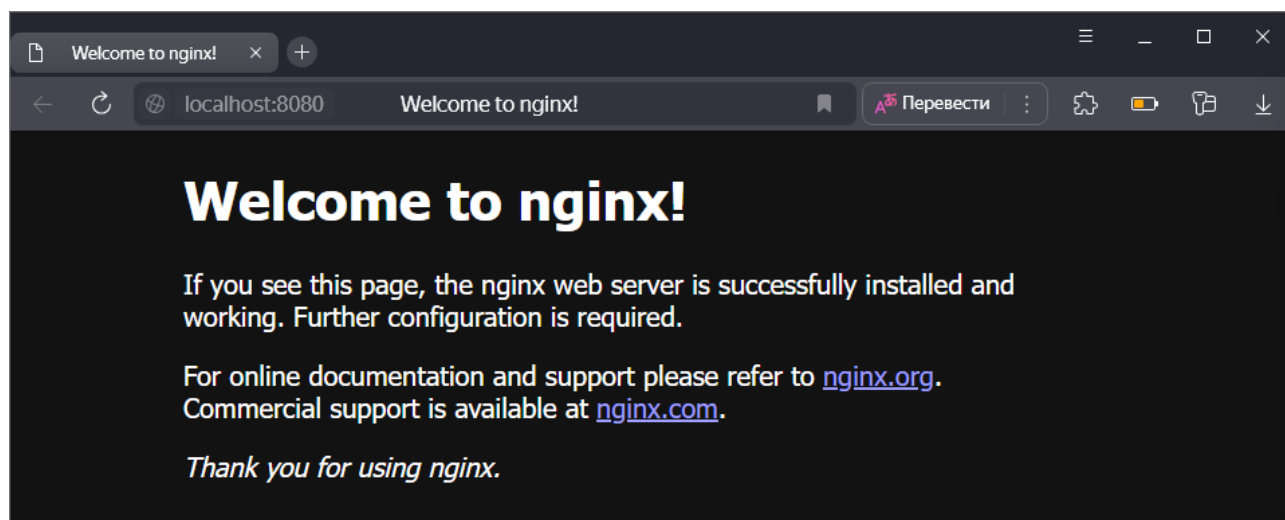


Рисунок 19 – Главная страница Nginx

### Шаг 2. Передача в контейнер html-файлов

Для выполнения данного шага была создана директория `html` в корневой папке проекта, в ней размещен файл `index.html`. В уже имеющийся `docker-compose.yml` файл добавляем инструкции по подключению внешнего каталога. Содержимого `docker-compose.yml` представлено на рисунке 20.

```

GNU nano 7.2
version: "3.0"
services:
  nginx:
    image: nginx
    ports:
      - "8080:80"
    volumes:
      - type: bind
        source: ./html
        target: /usr/share/nginx/html

```

Рисунок 20 – Содержимое docker-compose.yml

Пересоздадим контейнер с помощью `sudo docker compose up -d` и проверим работу перейдя на тот же адрес <http://localhost:8080>. Результат представлен на рисунке 21.

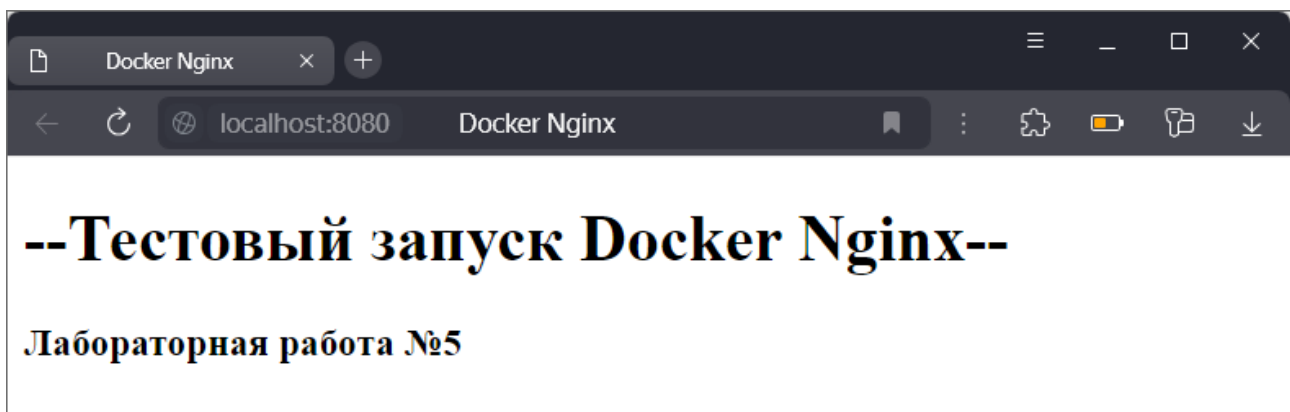


Рисунок 21 – Вывод собственного html файла

### Шаг 3. Web-разработка

Прежде чем приступать к данному шагу настроим сетевое подключение в virtual box. В разделе "Устройства"->"Сеть"->"Настроить сеть"->"Адаптер 1" выбрать подключение "Сетевой мост". После выполнить команду `ip a`, чтобы узнать, как `ip` был присвоен виртуальной машине.

```

2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
link/ether 08:00:27:28:ab:54 brd ff:ff:ff:ff:ff:ff
inet 172.20.10.11/28 brd 172.20.10.15 scope global dynamic enp0s3
    valid_lft 3327sec preferred_lft 3327sec
inet6 fe80::a00:27ff:fe28:ab54/64 scope link
    valid_lft forever preferred_lft forever

```

Рисунок 22 – Проверка ip

Из рисунка 22 видно, что был присвоен следящий `ip` адрес: 172.20.10.11. Данный адрес будет использоваться для доступа к сервису в веб-браузере.

Теперь создаем новую директорию proxy, в ней новый docker-compose.yml файл. Содержимое данного файла представлено на рисунке 23.

```
GNU nano 7.2
version: '3.0'
services:
  proxy:
    image: jwilder/nginx-proxy
    ports:
      - "80:80"
    volumes:
      - /var/run/docker.sock:/tmp/docker.sock:ro
    networks:
      - proxy
networks:
  proxy:
    driver: bridge
```

Рисунок 23 – Содержимое docker-compose.yml

Производим сборку и запуск контейнера из этой же директории с помощью команды `sudo docker compose up -d`. Результат представлен на рисунке 24. С помощью команды `sudo docker network ls` проверим список сетей, результат представлен на рисунке 25.

```
nikita@deb-server: ~/my-demo/proxy$ sudo docker compose up -d
[sudo] пароль для nikita:
WARN[0000] /home/nikita/my-demo/proxy/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion

[+] Running 14/14
 * proxy Pulled                                17.55s
 * bc0965b23a04 Already exists                  0.00s
 * 650ee30bbe5e Pull complete                   13.05s
 * 8cc1569e58f5 Pull complete                   13.11s
 * 362f35df001b Pull complete                   13.11s
 * 13e320bf29cd Pull complete                   13.11s
 * 7b50399908e1 Pull complete                   13.11s
 * 57b64962dd94 Pull complete                   13.22s
 * 886092e0261e Pull complete                   13.22s
 * ae399b1f8c35 Pull complete                   13.33s
 * c37f81e412db Pull complete                   13.55s
 * 05f3cee9d717 Pull complete                   13.65s
 * b2bfd419a37e Pull complete                   13.65s
 * 4f4fb700ef54 Pull complete                   13.65s
[+] Running 2/2
 * Network proxy_proxy Created                  0.25s
 * Container proxy-proxy-1 Started              0.95s
nikita@deb-server: ~/my-demo/proxy$
```

Рисунок 24 – Результат запуска proxy

```
nikita@deb-server: ~/my-demo/proxy$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
28ba6e462381        bridge              bridge              local
c7801fbd032f        host                host                local
b351d7fc7e62        my-demo_backend     bridge              local
0e9f76764fcc        none                null                local
1096bfb6796f        proxy_proxy         bridge              local
nikita@deb-server: ~/my-demo/proxy$ _
```

Рисунок 25 – Список созданных сетей

Изменяем файл docker-compose.yml, лежащий в корневой директории проекта. Содержимое данного файла:

```
version: '3.0'
```



services:

nginx:

image: nginx

environment:

VIRTUAL\_HOST: site.local

depends\_on:

- php

volumes:

- ./docker/nginx/conf.d/default.nginx:/etc/nginx/conf.d/default.conf

- ./html:/var/www/html/

ports:

- "8081:80"

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:

- "8082:80"

networks:

- frontend

- backend

networks:

frontend:

external:

name: proxy\_proxy

backend:

Добавляем файл конфигурации для nginx:

Создаем новую следующую иерархию в корневой папке проекта: docker/nginx/conf.d. Создаем файл конфигурации nginx (nano default.nginx). Содержимое данного файла представлено на рисунке 26.

```
GNU nano 7.2
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 26 – Содержимое default.nginx

Затем создаем директорию docker/php, и создаем Dockerfile для php. Его содержимое представлено на рисунке 27.

```
GNU nano 7.2
FROM php:fpm

RUN apt-get update && apt-get install -y libzip-dev zip
RUN docker-php-ext-configure zip
RUN docker-php-ext-install zip
RUN docker-php-ext-install mysqli

COPY --from=composer:latest /usr/bin/composer /usr/bin/composer
WORKDIR /var/www/html
```

Рисунок 27 – Содержимое Dockerfile

Также в директории html необходимо создать файл index.php. Его содержимое представлено на рисунке 28.

```
GNU nano 7.2
<?php

$link=mysqli_connect('mysql', 'root', 'root'); if
(! $link) {
    die('Ошибка соединения: ' . mysqli_error());
}
echo 'Успешно соединились';
mysqli_close($link);
```

Рисунок 28 – Содержимое index.php

Наконец производим сборку и запуск контейнера из корневой директории проекта: `sudo docker compose up -d`. На рисунке 29 представлена главная страница, а на рисунке 30 – контрольная панель базы данных.

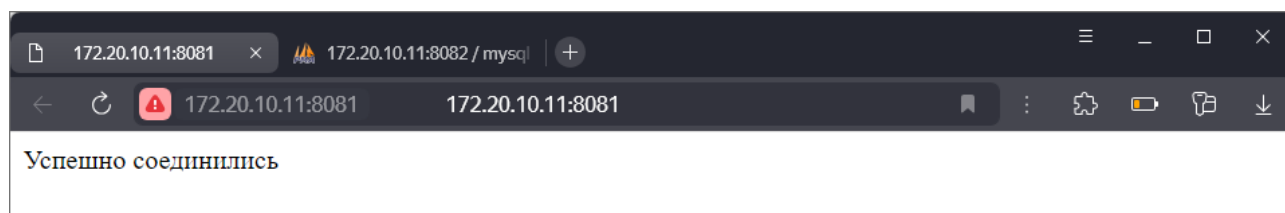


Рисунок 29 – Главная страница

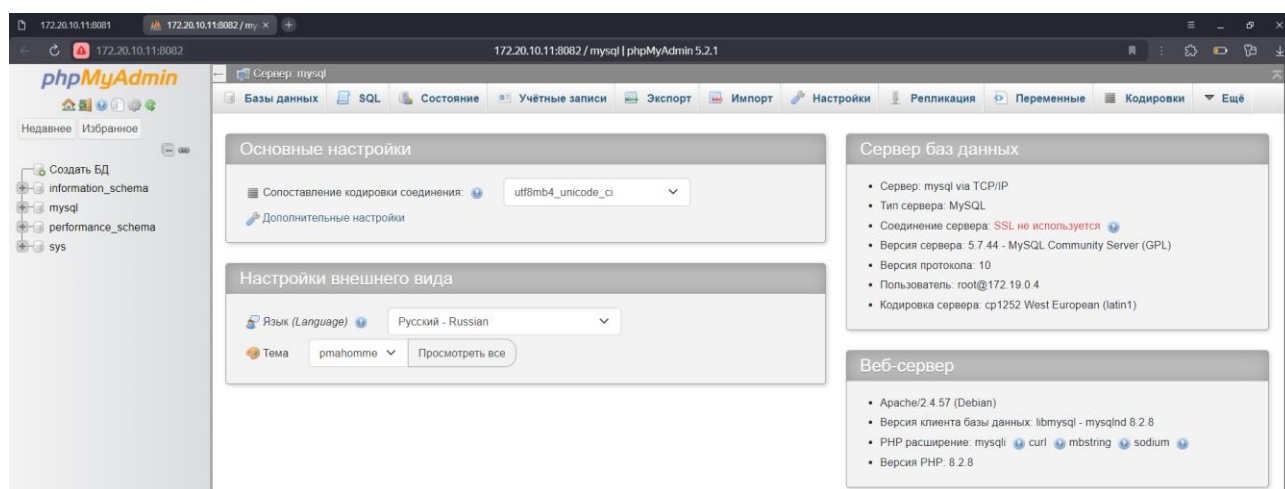


Рисунок 30 – Контрольная панель базы данных

## Шаг 4. WordPress

Произведем остановку и удаление созданного контейнера с помощью `sudo docker compose down`. После чего добавляем новый сервис WordPress в `docker-compose.yml`. Добавленный сервис представлен на рисунке 31.

```
wordpress:
  image: wordpress:latest
  environment:
    WORDPRESS_DB_HOST: mysql:3306
    WORDPRESS_DB_USER: root
    WORDPRESS_DB_PASSWORD: root
    WORDPRESS_DB_NAME: wordpress
  depends_on:
    - mysql
  volumes:
    - ./html/courses_data:/var/www/html/wp-content/uploads/courses_data
  ports:
    - "8083:80"
  networks:
    - frontend
    - backend
```

Рисунок 31 – Сервис WordPress

Заново производим сбор и запуск контейнеров. После переходим в контрольную панель управления базами данных и создаем новую базу данных wordpress. Результат представлен на рисунке 32.

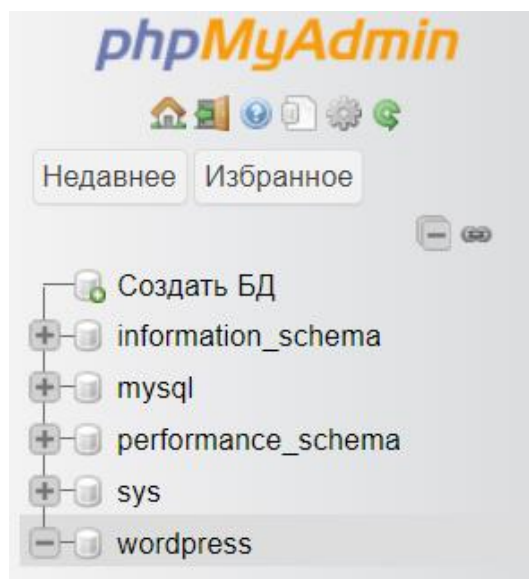


Рисунок 32 – Добавление базы данных

Теперь можем подключаться к сервису WordPress по адресу: <http://172.20.10.11:8083>. Нас встречает окно настройки (рисунок 33). На рисунке 34 представлена главная страница панели администрации, на которую можно попасть пройдя процесс настройки.

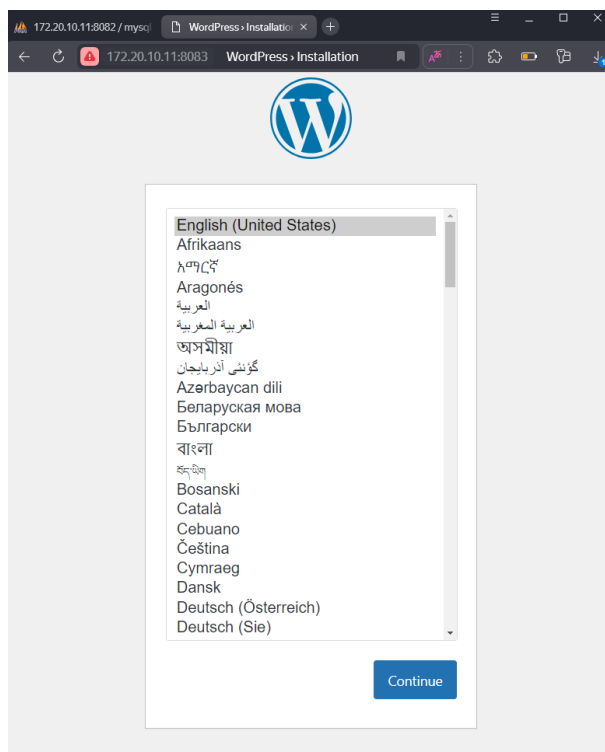


Рисунок 33 – Окно настройки

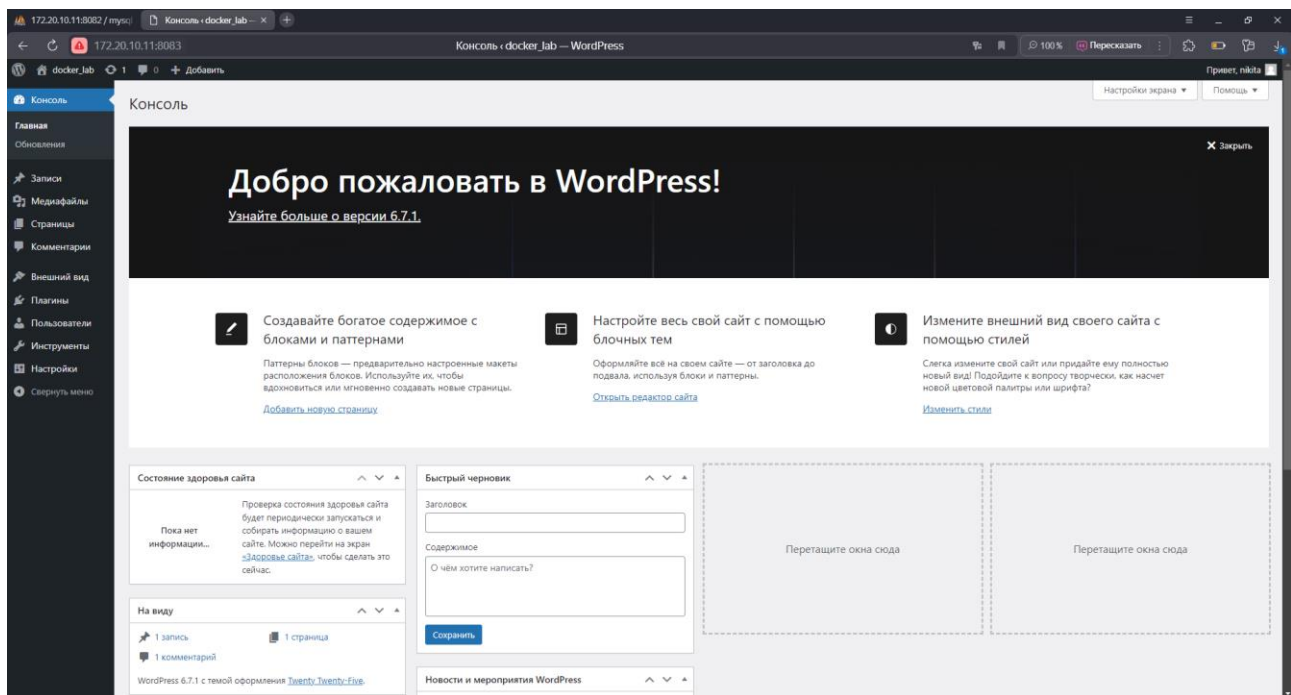


Рисунок 34 – Панель администратора WordPress

## **Вывод**

В ходе выполнения данной лабораторной работы я освоил технологии контейнеризации Docker, научился использовать контейнеры для развертки собственных сервисов. Также приобрел навыки по настройке внутренних виртуальных сетей между контейнерами и хостовой машиной.

## **Контрольные вопросы**

**1. Назовите отличия использования контейнеров по сравнению с виртуализацией.**

- Меньшие накладные расходы на инфраструктуру.

**2. Назовите основные компоненты Docker.**

- Контейнеры;
- Образы виртуальных машин;
- Реестры.

**3. Какие технологии используются для работы с контейнерами?**

- Пространство имен (Linux Namespaces)
- Контрольные группы (cgroups).

**4. Найдите соответствия между компонентами и его описанием.**

- Контейнеры: изолированные при помощи ОС 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?**

Виртуализация использует гипервизор, который создает виртуальные машины, каждая из которых содержит свою копию операционной системы (гостевой ОС), ядро и все необходимые зависимости. Это обеспечивает полную изоляцию, но требует значительных вычислительных ресурсов и большего времени на запуск.

Контейнеры работают на уровне операционной системы, используя общее ядро хостовой ОС. Контейнеры содержат только приложение и его зависимости, что делает их легковесными и обеспечивает быстрый запуск.

Таким образом, контейнеры более эффективны для развертывания и масштабирования приложений, тогда как виртуализация подходит для запуска изолированных систем с разными ОС на одном сервере.

**6. Перечислите основные команды утилиты Docker с их кратким описанием.**

`docker pull <image>`: загрузка образа из реестра;  
`docker run <image>`: запуск контейнера из образа;  
`docker ps`: список запущенных контейнеров (-а включая остановленные);  
`docker stop <container>`: остановка контейнера;  
`docker build -t <name>`: создание образа из Dockerfile;  
`docker push`: загрузка контейнера в реестр;  
`docker compose up`: запуск многоконтейнерного приложения;  
`docker network ls`: список сетей Docker;  
`docker rm <containers>`: удаление контейнеров;  
`docker images`: список локальных образов;  
`docker rmi <image>`: удаление образа.

## **7. Каким образом осуществляется поиск образов контейнеров?**

Поиск образов производится через реестры – хранилища контейнерных образов, например, Docker Hub. Команда: `docker search <image>`.

## **8. Каким образом осуществляется запуск контейнера?**

С помощью команды `docker run <image>` или `docker compose up` для многоконтейнерного приложения.

## **9. Что значит управлять состоянием контейнеров?**

Контейнеры имеют следующие состояния:

- Создан: начальное состояние контейнеров после их сборки;
- Выполняется: основное состояние рабочего контейнера;
- Приостановлен: работа процессов временно заморожена;
- Остановлен: состояние после завершения главного процесса;
- Удален.

Управление состоянием контейнеров включает в себя обработку переходов между вышеприведенными состояниями.

## **10. Как изолировать контейнер?**

Для изоляции контейнера используются:

- Пространства имен (Linux Namespaces): изоляция ресурсов (процессы, сеть);



- Контрольные группы (cgroups): управление ресурсами;
- Проброс портов или изолированная сеть;
- Сторонние программные решения для более жесткого контроля и изоляции: SELinux, AppArmor, Seccomp.

## **11. Опишите последовательность создания новых образов, назначение Dockerfile?**

- Создание Dockerfile (текстовый файл, содержащий набор инструкций, которые Docker использует для создания образа. Этот файл определяет, что будет установлено и настроено в образе);

- Написание инструкций в Dockerfile:
  - FROM: указывает базовый образ;
  - RUN: выполняет команды, например, установка пакетов;
  - COPY: копирует файлы из локальной системы в образ;
  - CMD или ENTRYPOINT: определяет команду, которая будет выполняться при запуске контейнера;
- Дополнительные инструкции: ENV, EXPOSE, WORKDIR, VOLUME.

- Сборка образов: `docker build -t <name> <path>;`

- Запуск контейнера: `docker run <images>.`

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

Для создания контейнеров без самого docker можно использовать альтернативное программное обеспечение, например, Kubernetes, rkt, LXC, LXD

## **13. Опишите назначение системы оркестрации контейнеров Kubernetes. Перечислите основные образы Kubernetes?**

Kubernetes – это система оркестрации контейнеров, предназначенная для автоматизации развертывания, управления, масштабирования и мониторинга контейнеризированных приложений. Она позволяет управлять кластерами, состоящими из множества узлов, на которых работают контейнеры.

Основные задачи Kubernetes:

- Управление контейнерами: автоматизация запуска, остановки и перезапуска контейнеров;
- Оркестрация: распределение контейнеров по узлам в кластере для оптимального использования ресурсов;
- Масштабирование: автоматическое увеличение или уменьшение числа запущенных контейнеров в зависимости от нагрузки;
- Мониторинг и самовосстановление: отслеживание состояния приложений, перезапуск упавших контейнеров и замена неработающих узлов;
- Сетевая связь: организация сети между контейнерами и управление доступом к приложениям через балансировщики нагрузки;
- Управление конфигурациями: хранение параметров конфигураций и данных, таких как пароли и ключи, в безопасной форме.

Kubernetes использует несколько базовых образов для своей работы. Среди них:

- kube-apiserver: главный компонент управления, предоставляющий API для взаимодействия с кластером.
- kube-controller-manager: отвечает за фоновые процессы, такие как управление состоянием приложений, поддержание реплик и обработка событий.
- kube-scheduler: распределяет поды (группы контейнеров) по узлам на основе доступных ресурсов и требований.
- kube-proxy: компонент, обеспечивающий сетевую маршрутизацию между подами и узлами.
- kubelet: агенты, работающие на каждом узле кластера, которые управляют запуском контейнеров и следят за их состоянием.
- etcd: распределённое хранилище данных для хранения состояния кластера.
- coredns: DNS-сервис для внутреннего именования подов и сервисов.
- pause: легковесный образ, использующийся в качестве контейнера-хоста для сетевой инфраструктуры пода.

- kubectl: утилита командной строки для взаимодействия с кластером Kubernetes.

Эти образы работают вместе, обеспечивая полное управление контейнеризованными приложениями в кластере Kubernetes.