

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

Кистерёв В.А.

Руководитель

канд.техн.наук, доцент

ученая степень, ученое звание

подпись, дата

Кургасов В.В.

Липецк, 2024 г.

Оглавление

Цель работы	3
Ход работы	4
Часть 1	4
Часть 2	10
Вывод.....	16
Контрольные вопросы.....	17

Цель работы

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

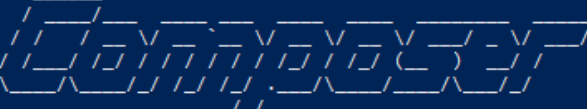
Часть 1

- `sudo apt update` (обновление списка пакетов)
- `sudo apt install git` (установка `git` для клонирования репозитория)
- `sudo apt install php` (установка `php`)

```
user@labs:~/test$ git clone https://github.com/symfony/demo
Cloning into 'demo'...
remote: Enumerating objects: 12637, done.
remote: Counting objects: 100% (190/190), done.
remote: Compressing objects: 100% (120/120), done.
remote: Total 12637 (delta 67), reused 146 (delta 60), pack-reused 12447 (from 1)
Receiving objects: 100% (12637/12637), 21.99 MiB | 16.19 MiB/s, done.
Resolving deltas: 100% (7527/7527), done.
```

С помощью документации (<https://getcomposer.org/download/>) установим composer (инструмент управления зависимостями в проектах на PHP). После установки выполним команду `composer`, чтобы убедиться, что он установлен. Результат выполнения представлен на рисунке 2.

```
user@labs:~/test/demo$ composer
```



```
Composer version 2.8.4 2024-12-11 11:57:47

Usage:
  command [options] [arguments]

Options:
  -h, --help                Display help for the given command. When no command is given display
                             this page
  -q, --quiet               Do not output any message
  -V, --version              Display this application version
                           --ansi|--no-ansi       Force (or disable) 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
                             details, 3 for debug
```

Установим **Symfony CLI** и нужные зависимости:

4

– curl -1sLf 'https://dl.cloudsmith.io/public/symfony/stable/setup.deb.sh' | sudo
-E bash

2) Установка Symfony CLI:

– sudo apt install symfony-cli

3) Установка зависимостей:

– sudo apt install php-sqlite3 (для работы с SQLite)

– sudo apt install php-xml (для обработки XML-файлов)

Перейдём в папку проекта и командой `composer install` установим все зависимости проекта, указанные в `composer.json`. Процесс установки зависимостей представлен на рисунке 3.

```
- Installing symfony/yaml (v7.2.0): Extracting archive
- Installing symfony/http-client (v7.2.1): Extracting archive
- Installing symfony/asset-mapper (v7.2.0): Extracting archive
- Installing symfonycasts/sass-bundle (v0.7.0): Extracting archive
- Installing twbs/bootstrap (v5.3.3): Extracting archive
- Installing twig/extra-bundle (v3.17.0): Extracting archive
- Installing symfony/intl (v7.2.0): Extracting archive
- Installing twig/intl-extra (v3.17.0): Extracting archive
- Installing twig/markdown-extra (v3.17.0): Extracting archive
Generating autoload files
97 packages you are using are looking for funding.
Use the 'composer fund' command to find out more!

Run composer recipes at any time to see the status of your Symfony recipes.

phpstan/extension-installer: Extensions installed
> phpstan/phpstan-doctrine: installed
> phpstan/phpstan-symfony: installed
Executing script cache:clear [OK]
Executing script assets:install public [OK]
Executing script importmap:install [OK]
Executing script sass:build [OK]
user@labs:~/test/demo$
```

Рисунок 3 – Процесс установки зависимостей

Запускаем demo проект symfony командой `symfony server:start`. Запуск проекта представлен на рисунке 4.

```
user@labs:~/test/demo$ symfony server:start --allow-http --allow-all-ip

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

Following Web Server log file (/home/user/.symfony5/log/e384d51dca384791b6c55712adb2b6deec9fa4f2.log)
Following PHP log file (/home/user/.symfony5/log/e384d51dca384791b6c55712adb2b6deec9fa4f2/7daf403c7589f4927632ed3b6
a992f09b78.log)
WARNING the current dir requires PHP 8.2.0 (composer.json from current dir: /home/user/test/demo/composer.json), bu
t version is not available: fallback to 8.2

[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 8.2.26
http://127.0.0.1:8000
```

Рисунок 4 – Запуск проекта

Переходим по адресу в браузере и видим, что демо приложение успешно запустилось (рисунок 5).

Добро пожаловать в **Symfony Demo** приложение

русский Выбрать язык ▾

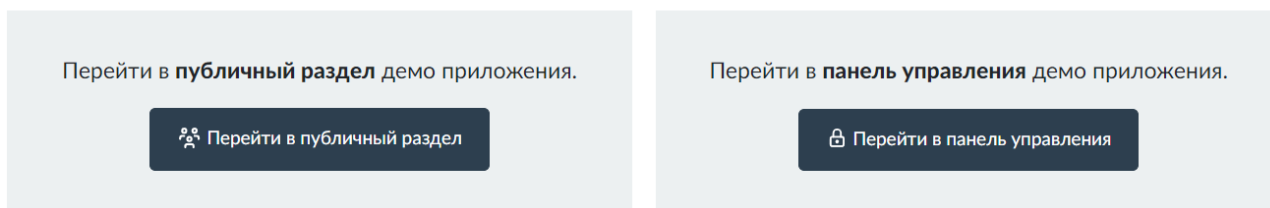


Рисунок 5 – Результат запуска приложения

Установка Docker

С помощью документации (<https://docs.docker.com/engine/install/debian/>) выполним установку Docker. Скриншот из документации с командами для установки представлен на рисунке 6.

```
# Add Docker's official GPG key:
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/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc \
$. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

Рисунок 6 – Команды для установки Docker

Убедимся, что Docker успешно установлен (рисунок 7).

```
user@labs:~/test/demo$ docker
Usage: docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Common Commands:
run           Create and run a new container from an image
exec         Execute a command in a running container
ps           List containers
build        Build an image from a Dockerfile
pull         Download an image from a registry
push         Upload an image to a registry
images       List images
login        Authenticate to a registry
logout       Log out from a registry
search       Search Docker Hub for images
version      Show the Docker version information
info         Display system-wide information
```

Рисунок 7 – Проверка установки Docker

В папке с проектом (demo) создадим Dockerfile для запуска php приложения. Содержимое Dockerfile представлено на рисунке 8.

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

Рисунок 8 – Содержимое Dockerfile

В общей папке (../demo) создадим docker-compose.yml для запуска сервисов. Его содержимое представлено на рисунке 9.

```
GNU nano 7.2                               docker-compose.yml
services:
  app:
    container_name: symfony
    restart: always
    build: ./demo
    ports:
      - "80:8000"
    links:
      - postgres
  postgres:
    container_name: postgres
    image: postgres
    ports:
      - "5432:5432"
    environment:
      POSTGRES_PASSWORD: postgres
    volumes:
      - pgdata:/var/lib/postgresql/data
volumes:
  pgdata:
```

Рисунок 9 – Содержимое docker-compose.yml

В .env файле в корневой папки проекта в переменной DATABASE_URL пропишем параметры для подключения к Postgres. Содержимое .env файла представлено на рисунке 10.

```
###> 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/conn
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
#
DATABASE_URL="postgresql://postgres:postgres@postgres:5432/postgres?serverVersion=17.1&charset=utf8"
# DATABASE_URL="mysql://app:!ChangeMe!@127.0.0.1:3306/app?serverVersion=8&charset=utf8mb4"
# DATABASE_URL="postgresql://app:!ChangeMe!@127.0.0.1:5432/app?serverVersion=16&charset=utf8"
###< doctrine/doctrine-bundle ###

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

Рисунок 10 – Содержимое .env файла

Командой `docker compose up --build` выполним сборку и запуск Docker контейнеров (рисунок 11).

```
symfony | [WARNING] The local web server is optimized for local development and MUST ne
symfony | ver be used in a production setup.
symfony |
symfony |
symfony |
symfony |
symfony | [OK] Web server listening
symfony | The Web server is using PHP FPM 8.2.10
symfony | http://127.0.0.1:8000
symfony |
symfony |
```

Рисунок 11 – Сборка и запуск Docker контейнеров

Подключимся к терминалу Docker приложения и выполним следующие команды:

– `php bin/console doctrine:schema:update --force` (синхронизация структуры базы данных с сущностями)(рисунок 12).

```
root@4e13e6efc946:/app# php bin/console doctrine:schema:update --force
Updating database schema...

18 queries were executed

[OK] Database schema updated successfully!

root@4e13e6efc946:/app#
```

Рисунок 12 – Синхронизация структуры базы данных с сущностями

– `php bin/console doctrine:fixtures:load` (загрузка фикстур)(рисунок 13).

```
root@4e13e6efc946:/app# php bin/console doctrine:fixtures:load

Careful, database "postgres" will be purged. Do you want to continue? (yes/no) [no]:
> yes

> purging database
> loading App\DataFixtures\AppFixtures
root@4e13e6efc946:/app#
```


Рисунок 13 – Загрузка фикстур

Тестирование запущенного приложения через Docker представлено на рисунках 14, 15, 16.

Добро пожаловать в **Symfony Demo** приложение

русский Выбрать язык ▾

Перейти в публичный раздел демо приложения.

 Перейти в публичный раздел

Перейти в панель управления демо приложения.


 Перейти в панель управления

Рисунок 14 – Тестирование приложения

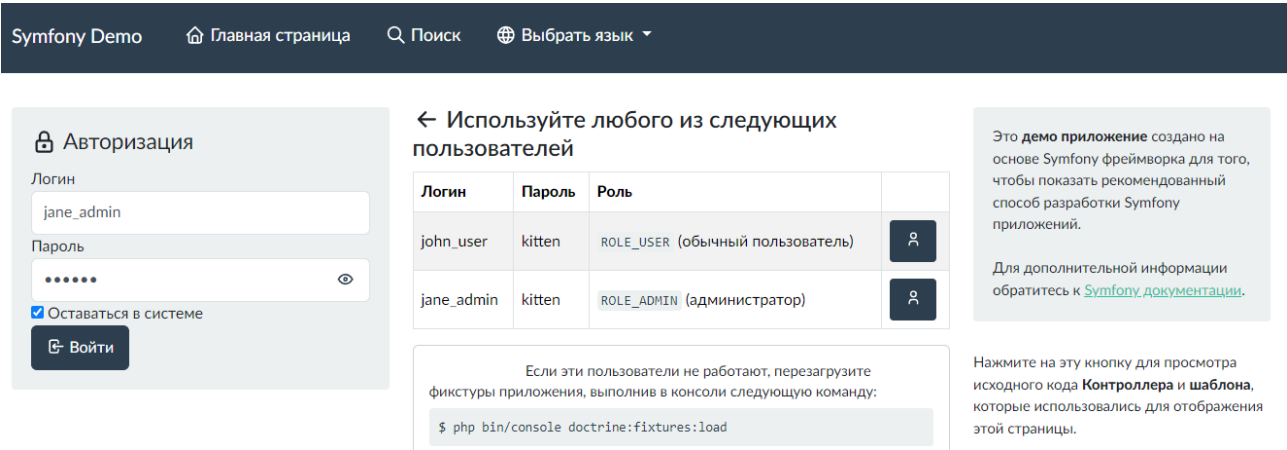


Рисунок 15 – Тестирование приложения

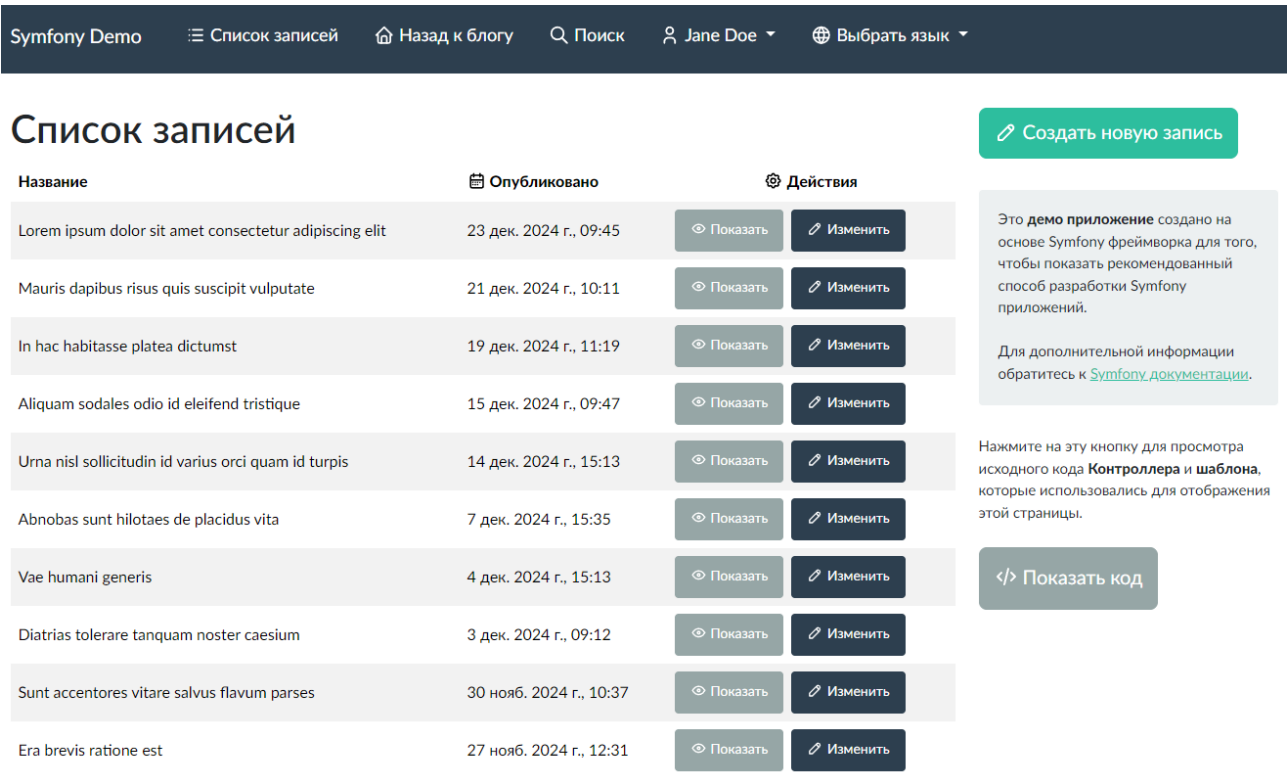


Рисунок 16 – Тестирование приложения

Часть 2

Для второй части задания была создана новая директория, в которой был создан docker-compose файл, содержимое которого представлено на рисунке 17, для запуска nginx.

```
GNU nano 7.2 docker-compose.yml
services:
  nginx:
    image: nginx
    ports:
      - "8080:80"
```

Рисунок 17 – Содержимое docker-compose

Командой `docker compose up` запустим nginx. Приветственная страница Nginx представлена на рисунке 18.



Рисунок 18 – Приветственная страница Nginx

В корневой папке проекта была создана директория `html`, в которую был помещён `html` файл. В `docker-compose` файл были добавлены инструкции по подключению внешнего каталога. Содержимое `docker-compose` файла представлено на рисунке 19.

```
GNU nano 7.2 docker-compose.yml
services:
  nginx:
    image: nginx
    ports:
      - "8080:80"
    volumes:
      - type: bind
        source: ./html
        target: /usr/share/nginx/html
```

Рисунок 19 – Содержимое docker-compose

Содержимое страницы после запуска docker-compose файла представлено на рисунке 20.

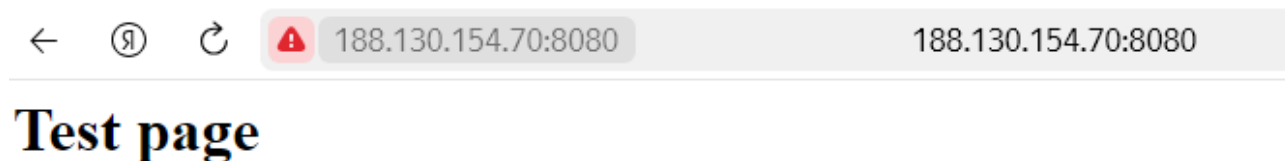


Рисунок 20 – Страница после запуска docker-compose файла

Создадим новую директорию проху, в которую поместим docker-compose файл, содержимое которого представлено на рисунке 21.

```
GNU nano 7.2 docker-compose.yml
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 21 – Содержимое docker-compose

После запуска контейнера командой `docker network ls` проверим список сетей, результат представлен на рисунке 22.

```
user@labs:~$ sudo docker network ls
[sudo] password for user:
NETWORK ID          NAME                DRIVER              SCOPE
7399bb1f67ce        bridge              bridge              local
78e9144c1793        env_default         bridge              local
69da03df4762        env_symfony_network bridge              local
1b12a0245585        host                host                local
a2e4f5da19fb        lb_default          bridge              local
14e95e83619d        none                null                local
b25514c727cd        proxy_proxy         bridge              local
bd6f1b80485c        test_default        bridge              local
eff827800a2c        test_symfony_network bridge              local
```

Рисунок 22 – Список сетей

Добавим сервисы `php`, `mysql` и `phpmyadmin` в docker-compose файл, находящийся в корневой папке проекта. Содержимое docker-compose файла представлено на рисунке 23.

```

GNU nano 7.2                                     docker-compose.yml
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:

```

Рисунок 23 – Содержимое docker-compose

Создадим файл конфигурации nginx по следующему пути: docker/nginx/conf.d. Содержимое файла конфигурации представлено на рисунке 24.

```

GNU nano 7.2                                     default.nginx
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 24 – Содержимое файла конфигурации

Создадим Dockerfile для php (docker/php). Его содержимое представлено на рисунке 25.

```
GNU nano 7.2 Dockerfile
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 25 – Содержимое Dockerfile для php

В директории html создадим файл index.php, содержимое которого представлено на рисунке 26.

```
GNU nano 7.2 index.php
<?php

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

Рисунок 26 – Содержимое index.php

После сборки и запуска контейнеров (docker compose up --build) содержимое страницы (по порту 8081) представлено на рисунке 27.

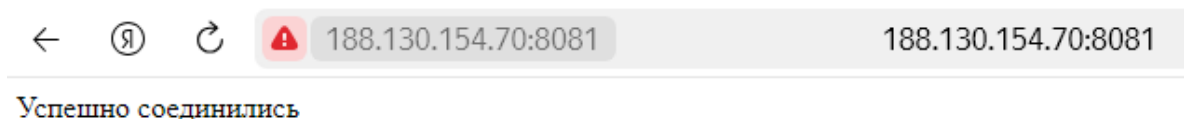


Рисунок 27 – Содержимое страницы

Подключимся к phpmyadmin (по порту 8082). Результат представлен на рисунке 28.

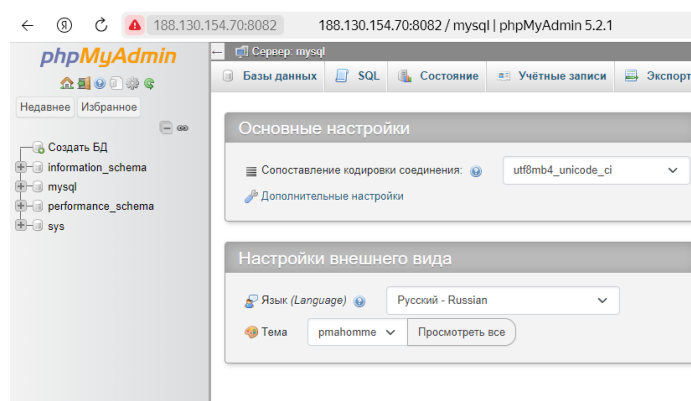


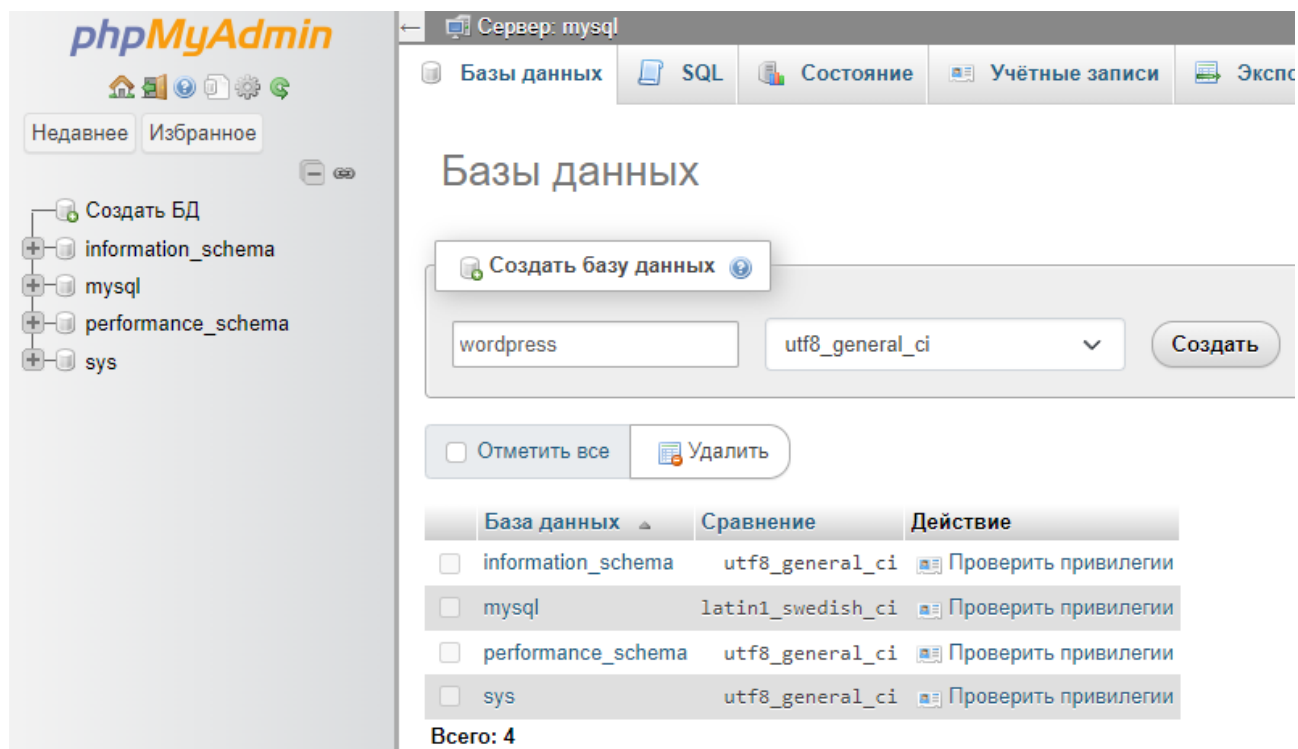
Рисунок 28 – Результат подключения к phpmyadmin

В docker-compose файл добавим новый сервис – wordpress. Добавленные строки представлены на рисунке 29.

```
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_date:/var/www/html/wp-content/uploads/courses_data
  ports:
    - "8083:80"
  networks:
    - frontend
    - backend
```

Рисунок 29 – Сервис wordpress

В phpmyadmin создадим новую бд с названием wordpress (рисунок 30).



После сборки и запуска контейнеров (docker compose up --build) подключимся к wordpress (по порту 8083), результат представлен на рисунке 31.

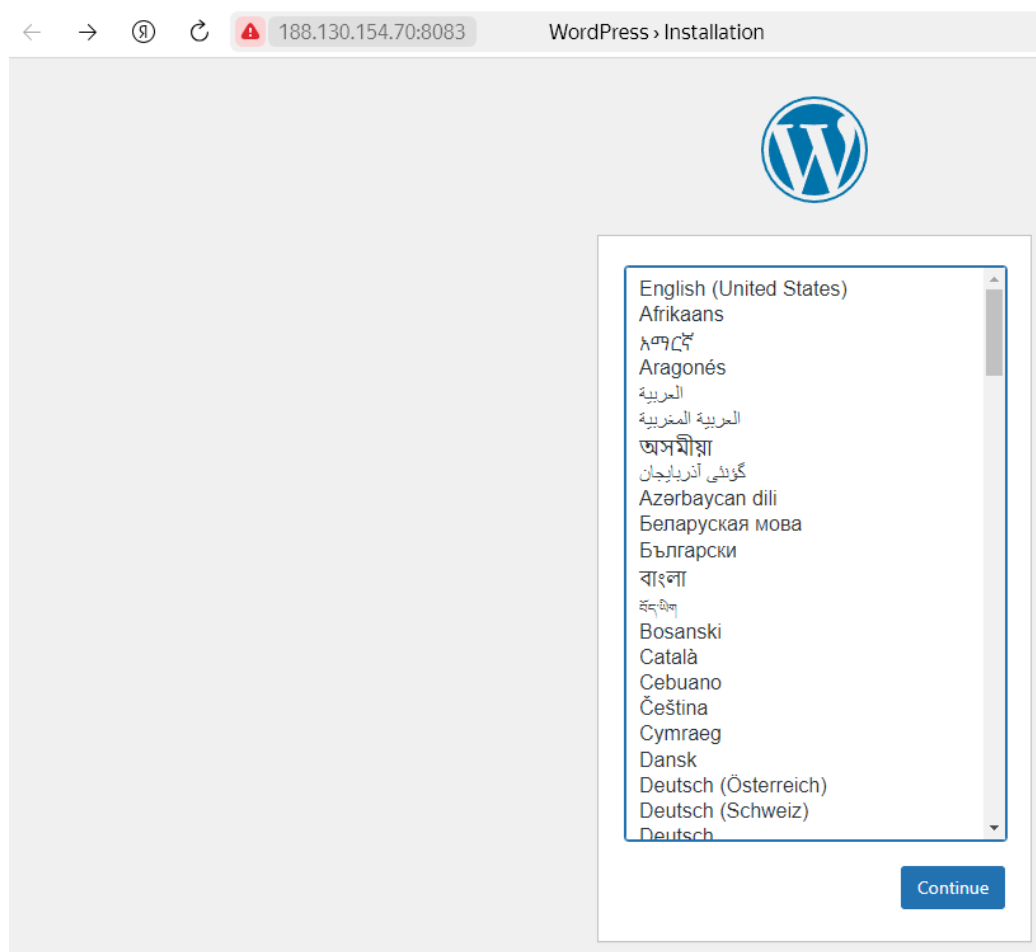


Рисунок 31 – Результат подключения к wordpress

Вывод

В ходе выполнения данной лабораторной работы были освоены технологии контейнеризации Docker, приобретены навыки по настройке внутренних виртуальных сетей между контейнерами и хостовой машиной.

Контрольные вопросы

1. Назовите отличия использования контейнеров по сравнению с виртуализацией.

Контейнеры используют ядро хостовой ОС, обеспечивая лёгкую изоляцию и меньшие затраты ресурсов.

2. Назовите основные компоненты Docker.

Docker Engine: движок, управляющий контейнерами.

Docker CLI: интерфейс командной строки для работы с Docker.

Docker Hub: репозиторий для хранения и обмена образами.

Docker Compose: инструмент для управления многоконтейнерными приложениями

3. Какие технологии используются для работы с контейнерами?

cgroups: ограничение ресурсов.

namespaces: изоляция процессов.

OverlayFS: работа с многослойными файловыми системами.

4. Найдите соответствия между компонентами и его описанием.

Docker Engine – управление контейнерами.

Docker Hub – репозиторий образов.

Docker Compose – управление многоконтейнерными приложениями.

Docker CLI – взаимодействие через терминал.

5. В чем отличие контейнеров от виртуализации?

Контейнеры изолируют приложения в пределах одной ОС, тогда как виртуализация создает полные копии ОС для каждого приложения.

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

`docker run`: запуск контейнера.

`docker ps`: просмотр запущенных контейнеров.

`docker pull`: загрузка образа из Docker Hub.

`docker build`: создание образа.

`docker stop`: остановка контейнера.

`docker rm`: удаление контейнера.

7. Каким образом осуществляется поиск образов контейнеров?

Осуществляется через команду `docker search`, которая ищет образы в Docker Hub или других репозиториях.

8. Каким образом осуществляется запуск контейнера?

Используется команда `docker run`, например: `docker run -d -p 80:80 nginx`. Или через `docker compose up` для многоконтейнерного приложения.

9. Что значит управлять состоянием контейнеров?

Управление состоянием контейнеров включает в себя обработку переходов между состояниями.

Запуск (`docker start`), остановка (`docker stop`), перезапуск (`docker restart`) и удаление (`docker rm`) контейнеров.

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 (K8s) – это платформа для оркестрации контейнеров, обеспечивающая автоматизацию следующих задач:

- Развёртывание приложений: управление жизненным циклом контейнеров, их запуск и обновление.

- Масштабирование: горизонтальное масштабирование (увеличение или уменьшение количества контейнеров в зависимости от нагрузки).

- Балансировка нагрузки: автоматическое распределение запросов между работающими контейнерами.

- Самовосстановление: перезапуск упавших контейнеров, замена или удаление нерабочих узлов.

- Управление конфигурациями и секретами: обеспечение безопасного и удобного хранения конфиденциальных данных и переменных среды.

- Организация сети: встроенные механизмы для обеспечения связи между контейнерами и их доступности извне.

Основные образы Kubernetes:

- kube-apiserver: центральный компонент, который предоставляет API для взаимодействия с кластером.

- kube-scheduler: компонент, отвечающий за распределение подов на узлы в кластере.

- kube-controller-manager: управляет различными контроллерами (например, контроллером репликации, узлов, конечных точек).
- kube-proxy: сетевой прокси для маршрутизации запросов между подами и службами.
- etcd: распределённое хранилище конфигураций и состояния кластера.
- coredns: предоставляет DNS-сервис для поиска сервисов и подов.
- kubectl: утилита командной строки для управления Kubernetes-кластером.
- kubelet: агент, работающий на каждом узле, следит за состоянием контейнеров и выполняет команды API-сервера.

Эти компоненты и образы совместно обеспечивают полный цикл управления контейнерами и их взаимодействие в распределённых системах.