**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

A logo of a triangle

Description automatically generated**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 | Первеева Е. Ю. |
| Руководитель | |  |  |  |
| канд.техн.наук, доцент  ученая степень, ученое звание | |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись, дата | Кургасов В.В. |

Липецк, 2024 г.

Оглавление

[Цель работы 3](#_Toc187886380)

[Ход работы 4](#_Toc187886381)

[1. Часть I 4](#_Toc187886382)

[2. Часть I – Установка Docker 9](#_Toc187886383)

[Часть II 14](#_Toc187886384)

[1. Установка Nginx 14](#_Toc187886385)

[2. Передача в контейнер html-файлов 14](#_Toc187886386)

[3. Web-разработка 15](#_Toc187886387)

[4. WordPress 19](#_Toc187886388)

[Вывод 22](#_Toc187886389)

[Контрольные вопросы 23](#_Toc187886390)

**Цель работы**

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

**Ход работы**

**1. Часть I**

Клонируем проект на сервер с помощью команды "git clone <https://github.com/symfony/demo>" и перейдем в папку (demo) с проектом командой cd.

Попытаемся запустить проект при помощи команды php bin/console server:start. Результат представлен на рисунке 1.

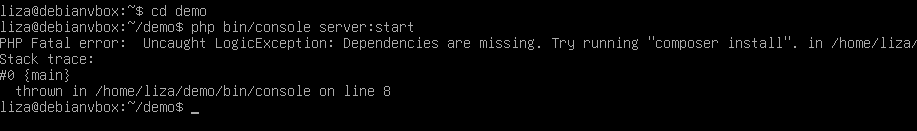


Рисунок 1 - Попытка запуска проекта

На рисунке 1 мы видим, что возникла ошибка запуска, так как при попытке запустить проект отсутствовали его зависимости. Требуется установить "Composer", который автоматически загружает и устанавливает библиотеки, указанные в "composer.json".

Результат установки Composer представлен на рисунке 2.



Рисунок 2 – Composer

На рисунке 2 видно, что версия composer 2.5.5. Необходимо обновить его до более новой версии. Для этого установим curl - он позволяет скачивать файлы или данные с веб-серверов. Теперь использованные команды для проверки хэша скрипта с указанным на официальном сайте composer:

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

Процесс продемонстрирован на рисунке 3.



Рисунок 3 - Проверка хеш-суммы

После этого выполним установочный скрипт: sudo php composer-setup.php --install-dir=/usr/local/bin --filename=composer

С помощью него будет загружена последняя версия Composer с официального репозитория. На рисунке 4 можно наблюдать успешное выполнение команды. Composer обновился до версии 2.8.4.

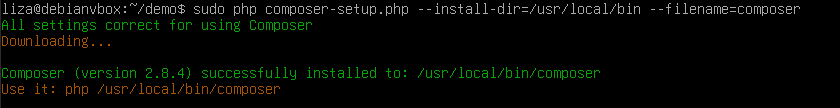


Рисунок 4 - Обновление Composer

Далее попытаемся установить все зависимости проекта, указанные в файле composer.json. На рисунке 5 продемонстрирована ошибка в процессе установки - отсутствуют модули php.

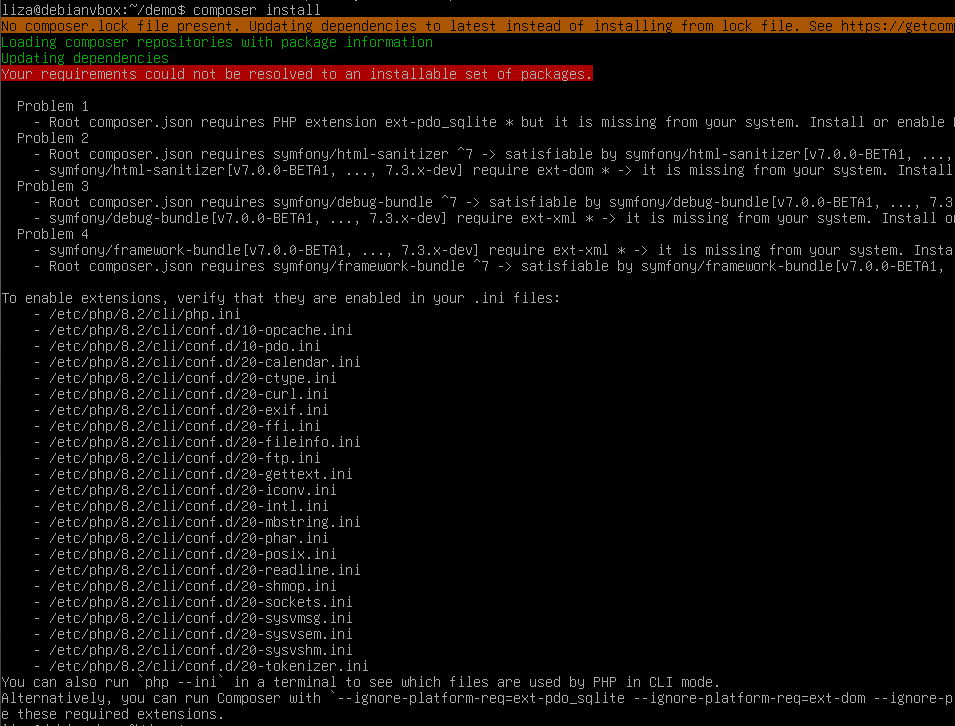


Рисунок 5 - Отсутствие php модулей

Установим недостающие модули, а именно php-xml (для работы с xml файлами) и php-sqlite3 (для использования в проекте sqlite).

Так как присутствовали проблемы при установке зависимостей, воспользуемся командой composer clear-cache для очистки кеша composer. После чего снова попытаемся подтянуть библиотеки из файла composer.json командой sudo composer install. На рисунке 6 продемонстрировано завершение успешной установки зависимостей.

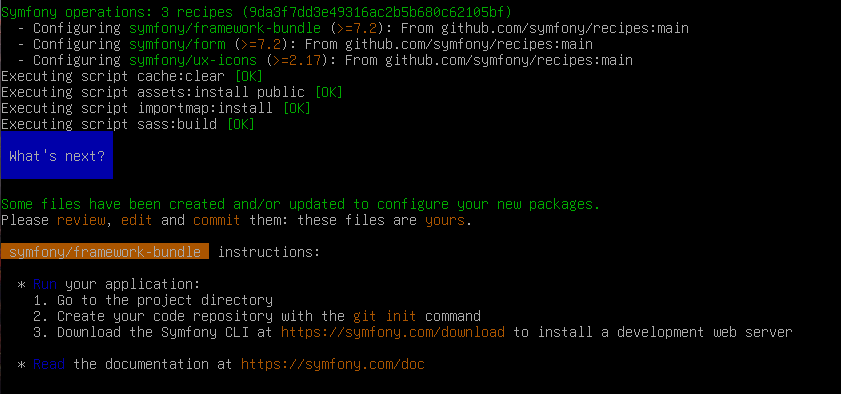


Рисунок 6 - Успешное выполнение composer install

Снова пробуем выполнить запуск сервера – php bin/console server:start, но сталкиваемся с очередной ошибкой, представленной на рисунке 7.

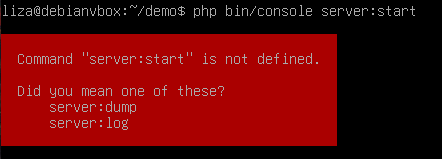


Рисунок 7 - Ошибка запуска сервера

Эта ошибка возникает из-за того, что команда server:start больше не существует в современных версиях Symfony. Начиная с Symfony 4.4, встроенный веб-сервер Symfony был удалён, и вместо этого рекомендуется использовать встроенный сервер PHP или внешний веб-сервер (например, Apache или Nginx).

Вместо команды server:start, запустим проект с помощью встроенного веб-сервера PHP: php -S 127.0.0.0:8000 -t public. Сайт был успешно запущен, но CSS, JavaScript, и другие ресурсы не были загружены, рисунок 8.



Рисунок 8 - Запуск проекта только с html

Если приложение использует маршруты для отдачи статических файлов (например, через Asset в Symfony), сервер может неправильно их обрабатывать. Для решения этой проблемы необходимо произвести установку Symfony CLI.

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 -1sLf '<https://dl.cloudsmith.io/public/symfony/stable/s>etup.deb.sh' |  
 sudo -E bash

После произведем установку пакета из только что добавленного репозитория: sudo apt install symfony-cli

Снова воспользуемся командой php -S 127.0.0.0:8000 -t public и теперь наш сервер будет доступен по адресу <http://127.0.0.1:8000>, рисунок 9.

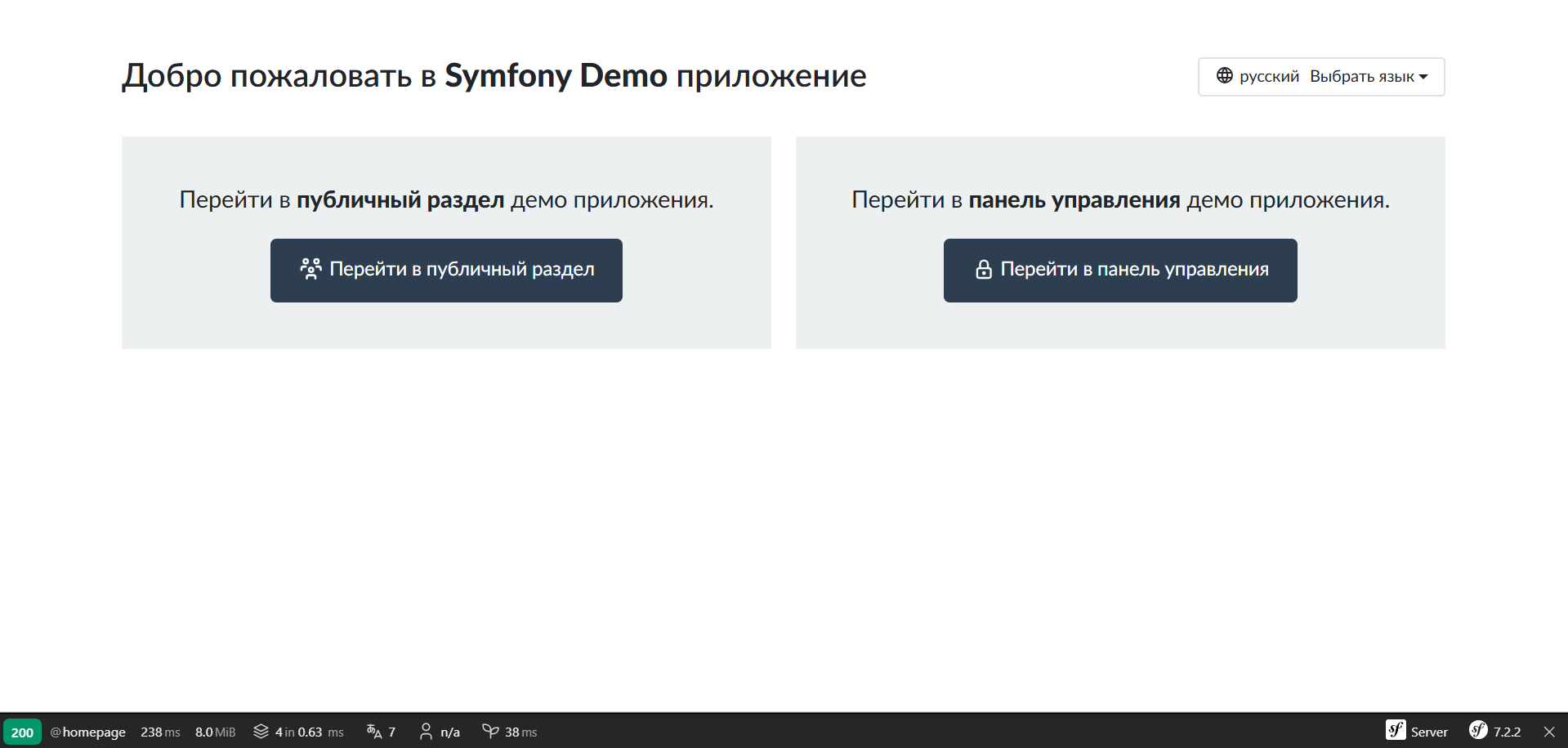


Рисунок 9 - Страница проекта

**2. Часть I – Установка Docker**

Для установки Docker воспользуемся официальной инструкцией с сайте <https://docs.docker.com/engine/install/debian/>. Последовательно выполним команды с сайта:

- 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

- echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/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

Затем 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 10.

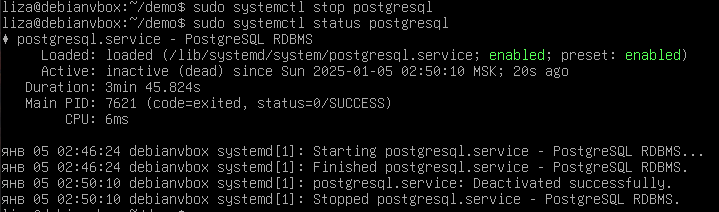


Рисунок 10 - Остановка PostgreSQL

В корневой папке проекта создаем Dockerfile, который будет содержать инструкции из задания для создания среды, в которой будет работать сервис. Его содержимое представлено на рисунке 11.



Рисунок 11 - Содержимое Dockerfile

Также необходимо создать файл docker-compose.yml для управления контейнерами и описания взаимодействия сервисов. Его содержимое представлено на рисунке 12.

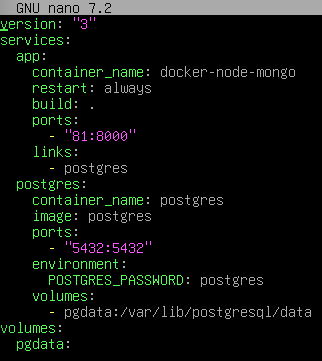


Рисунок 12 - Содержимое docker-compose.yml

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 переходим в новый терминал, создаем базу данных и заполняем созданную БД данными:

- 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>. Главная страница представлена на рисунке 13. Работа контрольной панели представлена на рисунке 14. Зайдем в учетную запись администратора, рисунок 15. Попробуем авторизироваться так же под учетной записью пользователя, рисунок 16. Из чего можно сделать вывод, что сервис и база данных в контейнере работают корректно.

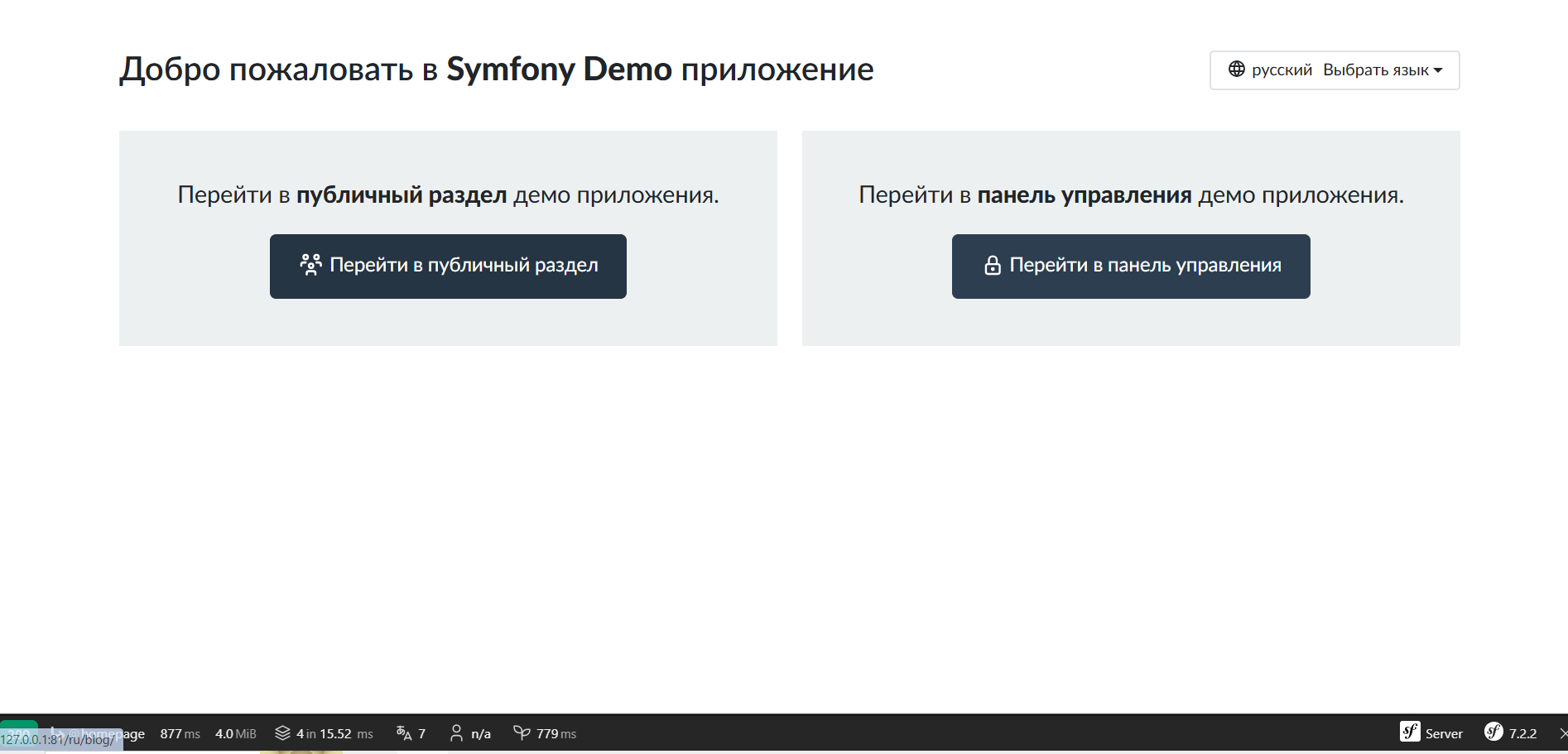


Рисунок 13 - Главная страница

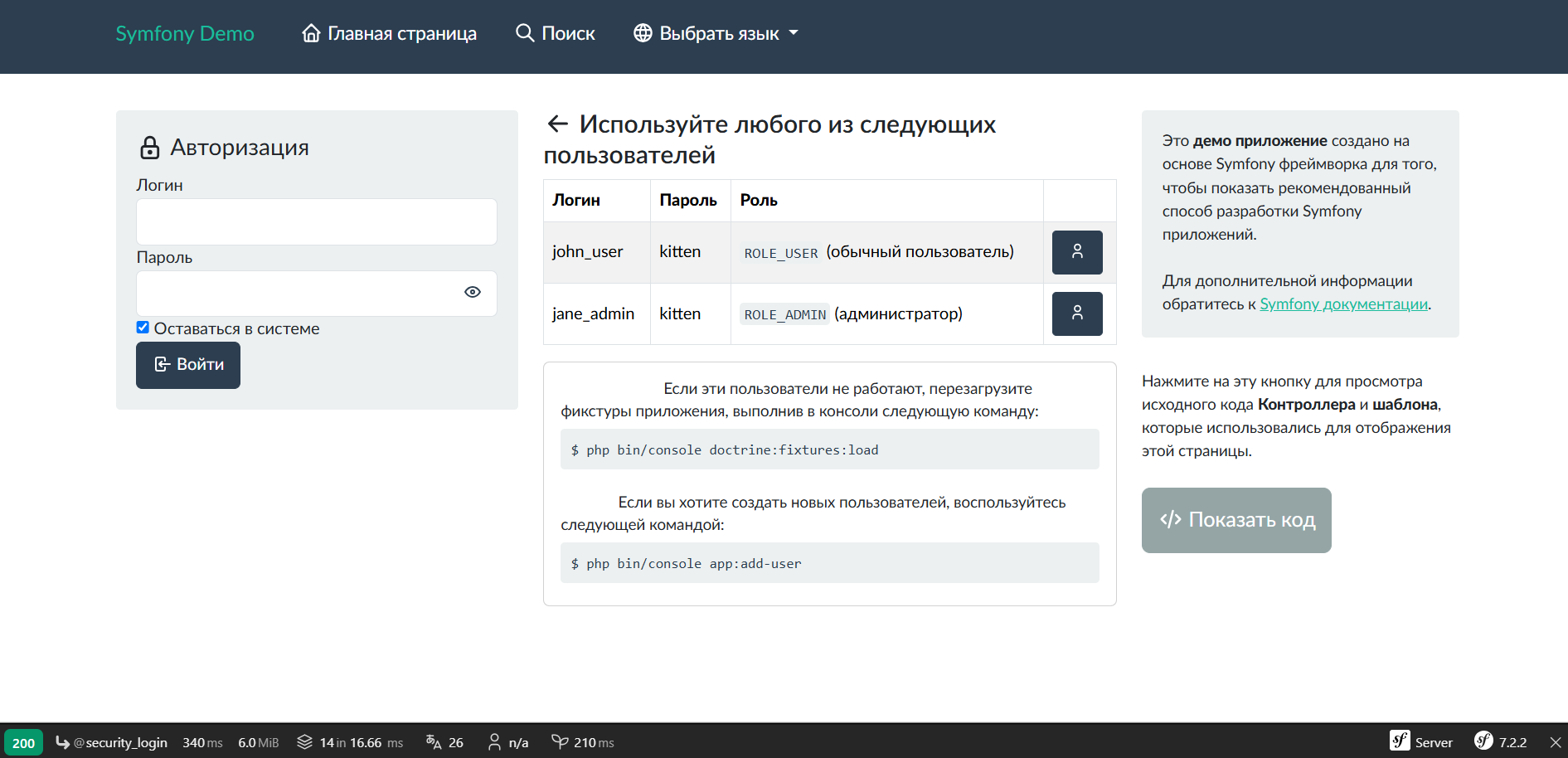


Рисунок 14 - Контрольная панель

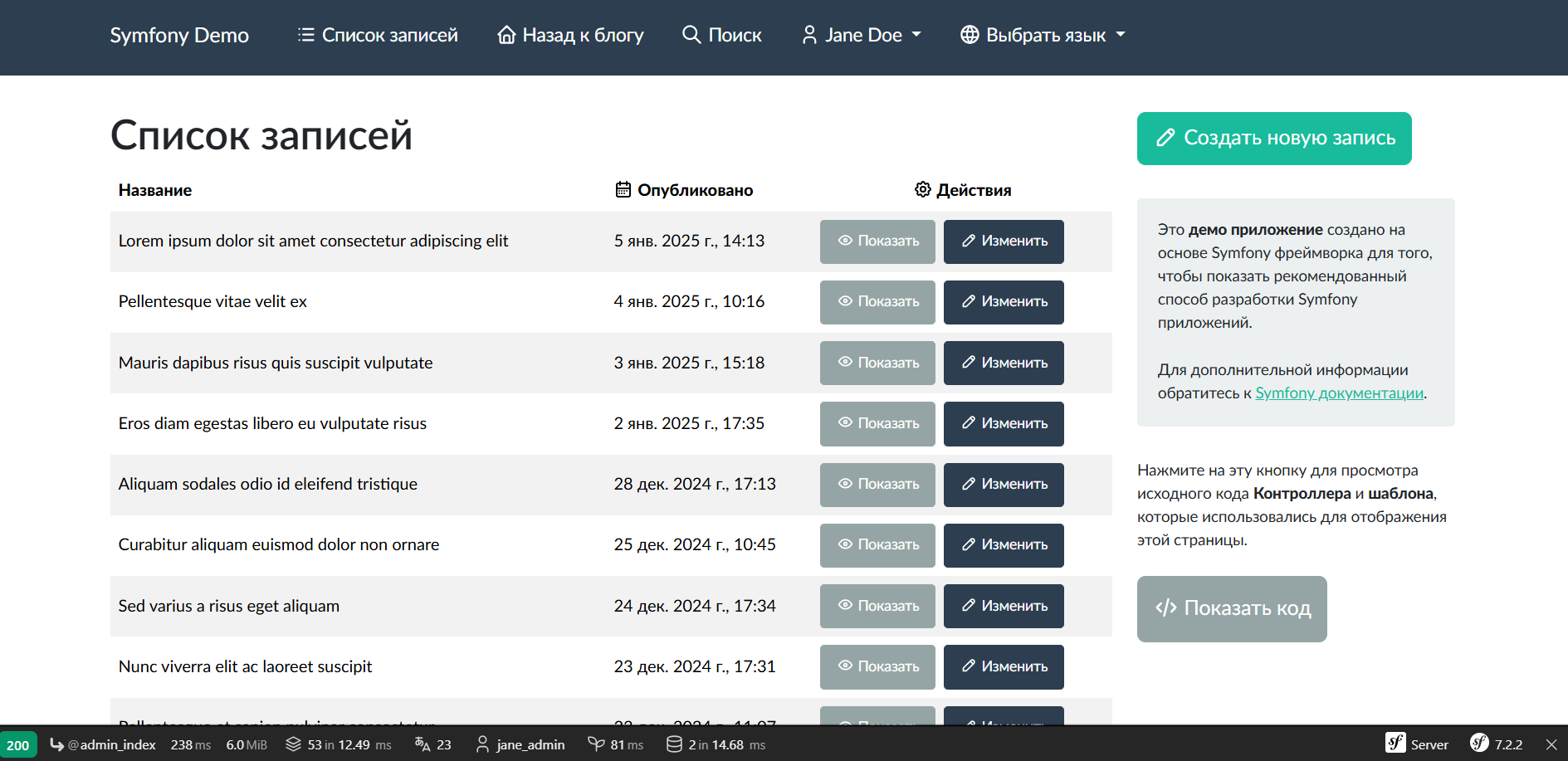


Рисунок 15 - Учетная запись администратора

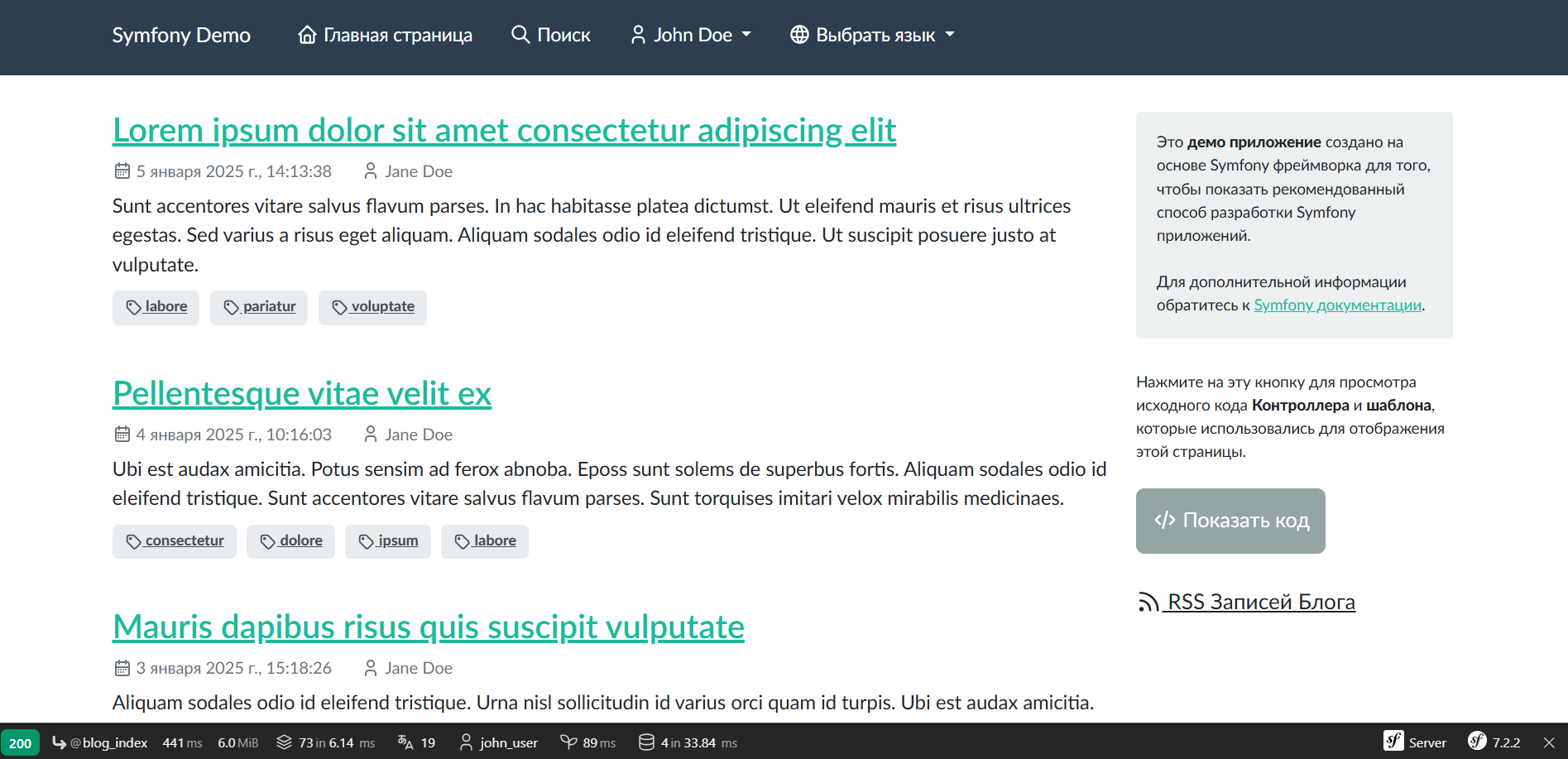


Рисунок 16 - Учетная запись пользователя

**Часть II**

**1. Установка Nginx**

Создаём новую директорию для проекта "project". В кореньевой папке этого проекта создаем файл docker-compose.yml, его содержимое представлено на рисунке 17

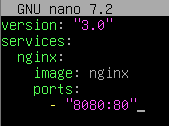


Рисунок 17 – Содержимое docker-compose.yml

Выполним команды для сборки и перезапуска контейнера:

* sudo docker compose down
* sudo docker compose up -d

Переходим по адресу <http://localhost:8080>. Приветственная страница Nginx представлена на рисунке 18.



Рисунок 18 – Главная страница Nginx

**2. Передача в контейнер html-файлов**

Была создана директория html в корневой папке проекта, в ней размещен файл index.html. В docker-compose.yml файл добавляем инструкции по подключению внешнего каталога. Содержимого docker-compose.yml представлено на рисунке 19.

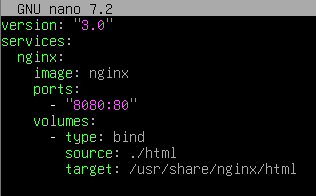


Рисунок 19 – docker-compose.yml

Пересоздадим контейнер с помощью sudo docker compose up -d и запустим его проверим по <http://localhost:8080>. Результат представлен на рисунке 20.

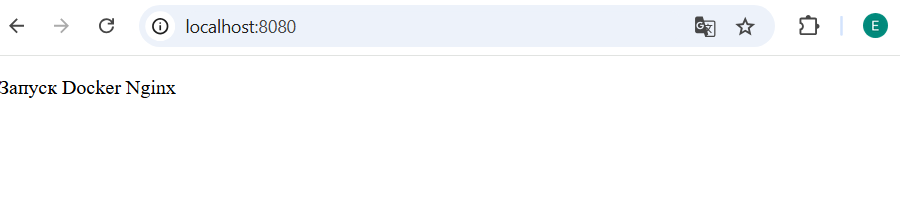


Рисунок 20 – Вывод html файла

Не останавливая работу контейнера, перейдем в другой терминал и изменим содержимое файла index.html, после чего обновим сайт <http://localhost:8080>. Мы можем увидеть, что изменения сразу появились, соответственно изменения применяются в реальном времени. Результат представлен на рисунке 21.

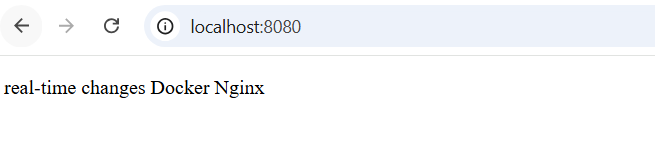


Рисунок 21 –Обновление html файла в реальном времени

**3. Web-разработка**

Настроим сетевое подключение в virtual box на "Сетевой мост". После выполнить команду ip addr и узнаем какой ip был присвоен виртуальной машине. Ip адрес: 192.168.43.19. Он 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 22.

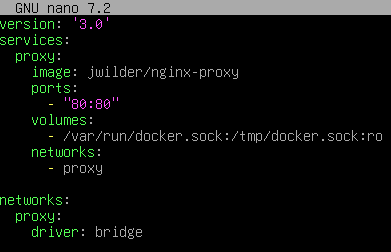


Рисунок 22 – Содержимое docker-compose.yml

Производим сборку и запуск контейнера. Результат представлен на рисунке 23. С помощью команды sudo docker network ls проверим список сетей, результат представлен на рисунке 24.

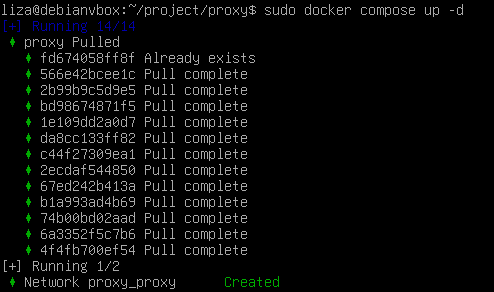


Рисунок 23 – Запуск proxy

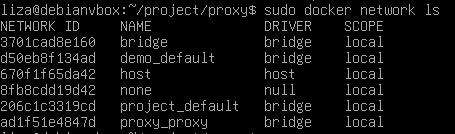


Рисунок 24 – Список созданных сетей

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

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: ./php

volumes:

- ./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 папку: docker/nginx/conf.d. Создаем файл конфигурации nginx. Содержимое данного файла представлено на рисунке 25.

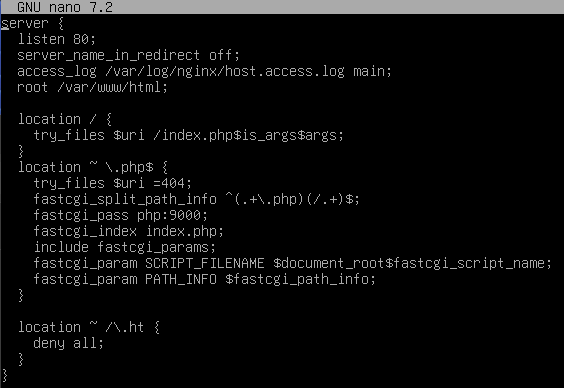


Рисунок 25 – Содержимое default.nginx

Затем создаем директорию docker/php, и создаем Dockerfile для php. Его содержимое представлено на рисунке 26.

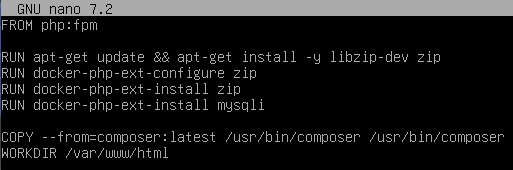


Рисунок 26 – Содержимое Dockerfile

Также в директории html необходимо создать файл index.php. Его содержимое представлено на рисунке 27.

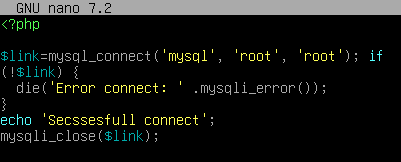


Рисунок 27 – Содержимое index.php

Производим сборку и запуск получившегося контейнера. На рисунке 28 представлена главная страница, а на рисунке 29 – контрольная панель базы данных.

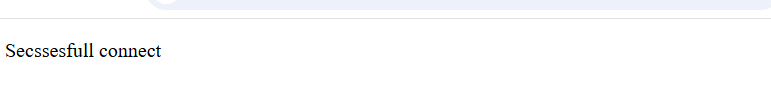


Рисунок 28 – Главная страница

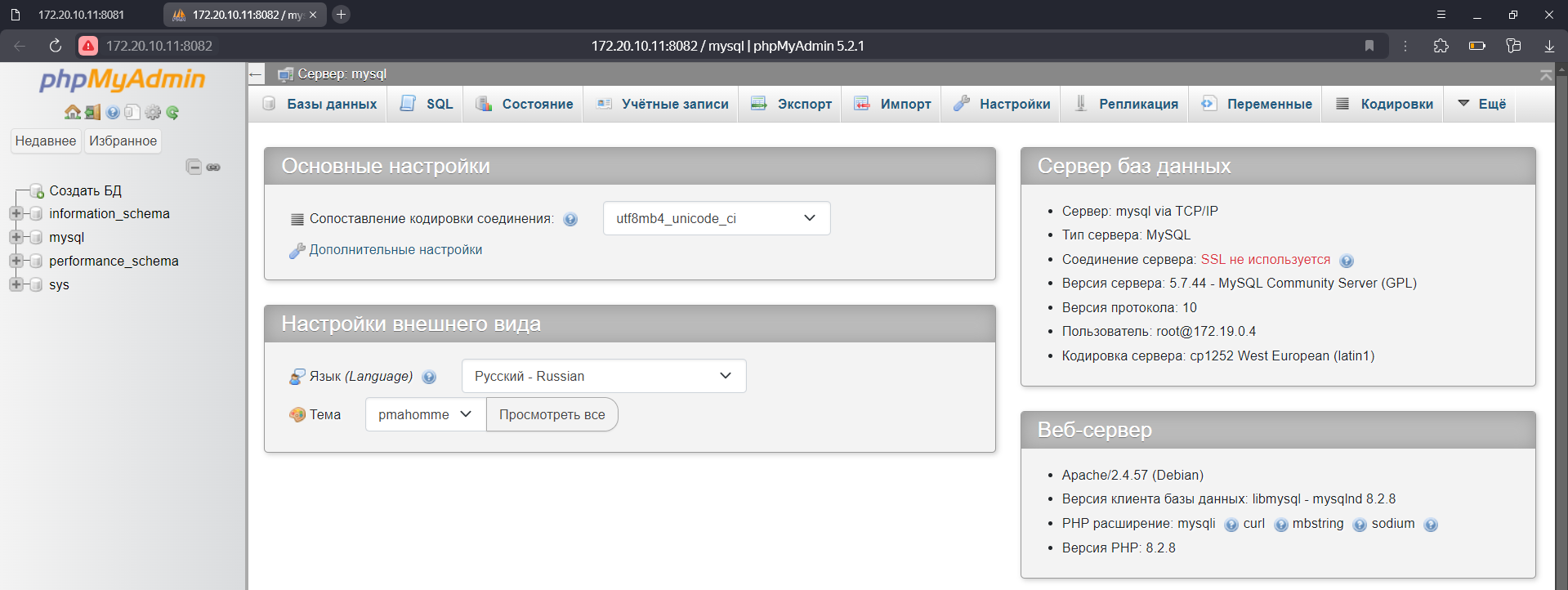


Рисунок 29 – Контрольная панель бд

**4. WordPress**

Добавляем новый сервис WordPress в docker-compose.yml, рисунок 30.

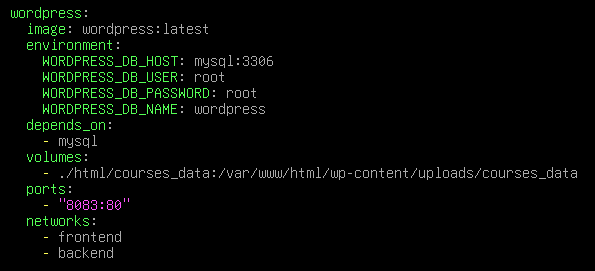


Рисунок 30 – Сервис WordPress

Производим сбор и запуск контейнеров. Переходим в контрольную панель управления бд и создаем новую базу данных wordpress. Результат представлен на рисунке 31.

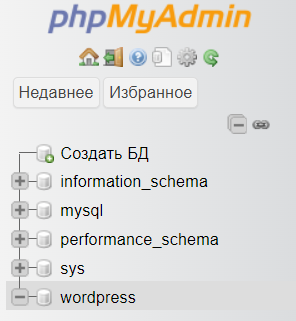


Рисунок 31 –Создание новой базы данных

Подключимся к сервису WordPress по адресу: <http://192.168.43.19:8083>. Нас встречает окно настройки (рисунок 32).

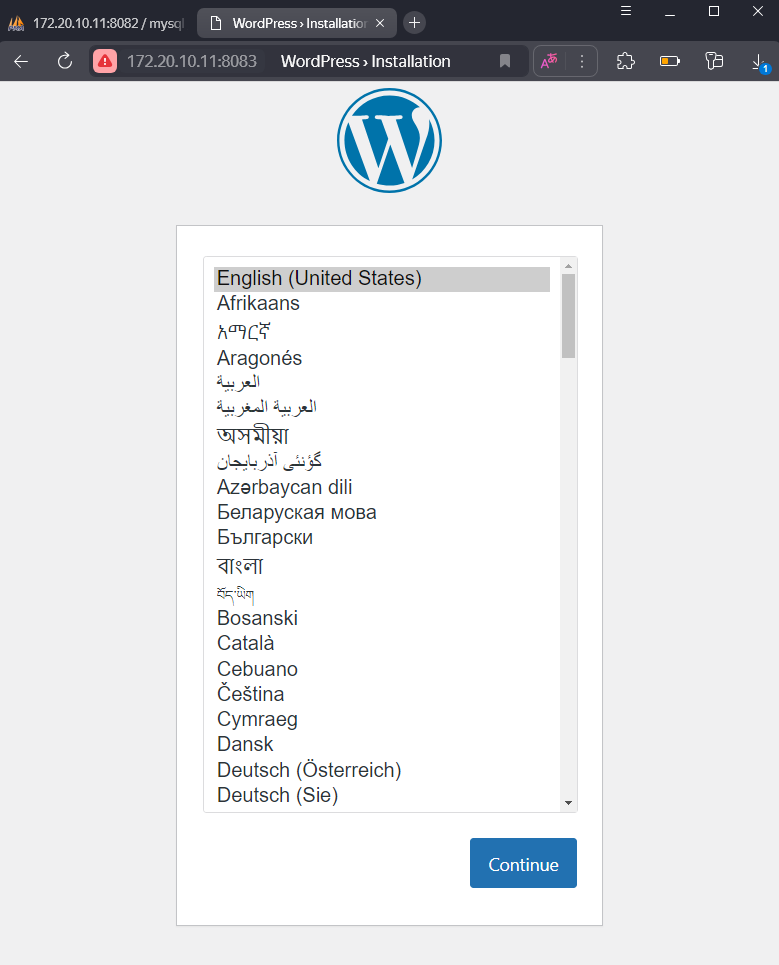


Рисунок 32 – Окно настройки

На рисунке 33 представлена главная страница панели администрации.

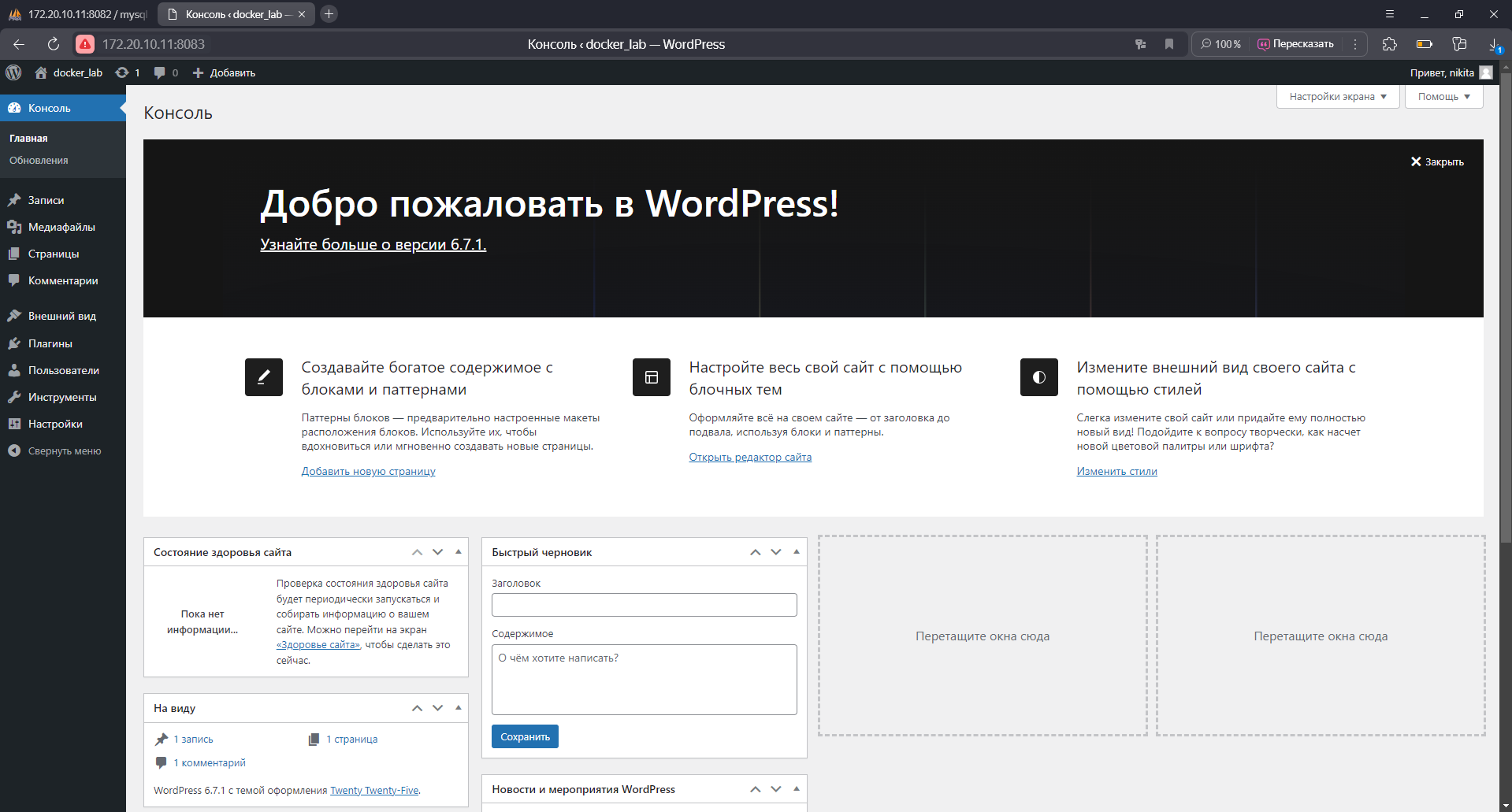


Рисунок 34 – Панель администратора WordPress

**Вывод**

В ходе выполнения данной лабораторной работы я освоила технологии контейнеризации Docker, научилась использовать контейнеры для развертки собственных сервисов. Также приобрела 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) и 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?**

Контейнеры используют ядро хоста и изолируют приложения на уровне операционной системы. Контейнеры содержат только приложение и его зависимости, что делает их легковесными и обеспечивает быстрый запуск. Они больше подходят для разработки, тестирования и масштабирования современных приложений.

Виртуализация запускает несколько полноценных операционных систем на гипервизоре. Это обеспечивает полную изоляцию, но требует значительных вычислительных ресурсов и большего времени на запуск. Виртуализация подходит для запуска различных операционных систем или когда требуется строгая изоляция.

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

docker run – создание и запуск контейнера на основе указанного образа;

docker ps - список всех запущенных контейнеров

docker build -t - строит образ из Dockerfile в текущей директории.

docker images - список всех локально доступных образов;

docker pull - скачивание образа из реестра Docker;

docker stop - остановка контейнера;

docker push - загрузка образа в реестр;

docker start – запускает остановленный контейнер.

docker compose up - запуск многоконтейнерного приложения;

docker network ls - список сетей Docker;

docker rm - удаление остановленных контейнеров;

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

Чтобы найти образы контейнеров в реестре Docker Hub, используется команда docker search <image>.

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

Прежде чем запустить контейнер, нужно скачать необходимый образ. Это можно сделать с помощью команды docker pull. Далее чтобы запустить контейнер на основе скачанного образа, используйте команду docker run.

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

Управление состоянием контейнеров означает управление жизненным циклом контейнеров: создание, запуск, остановка, перезапуск и удаление контейнеров.

**Создание контейнера** — запуск контейнера на основе образа.

Запуск и выполнение - основное состояние рабочего контейнера

Остановка контейнера - состояние после завершения главного процесса;

Удаление.

**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?**

Создаём Dockerfile - текстовый файл, содержащий набор инструкций, которые Docker использует для создания образа.

Напишем инструкций в Dockerfile:

- FROM: указывает базовый образ;

- RUN: выполняет команды, например, установка пакетов;

- COPY: копирует файлы из локальной системы в образ;

Соберём образ: docker build -t <name> <path>;

Запустим контейнер: docker run.

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

Обычно Docker Engine является движком для создания, управления и запуска контейнеров, однако существуют альтернативные способы работы с контейнерами без использования именно Docker Engine, такие как:

Podman — это инструмент для работы с контейнерами, который совместим с Docker, но не требует наличия демона.

CRI-O — это контейнерный движок, разработанный для работы с Kubernetes. Он поддерживает создание и запуск контейнеров, использует стандартные контейнерные образы, такие как Docker или OCI (Open Container Initiative).

rkt (Rocket) — еще один инструмент для работы с контейнерами, который фокусируется на безопасности и интеграции с Kubernetes.

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

Kubernetes — это система оркестрации контейнеров, предназначенная для автоматизации развертывания, масштабирования и управления конвейеризованными приложениями. Основные функции Kubernetes включают:

- Управление контейнерами: автоматизация запуска, остановки и перезапуска контейнеров;

- Оркестрация: управление множеством контейнеров, их развертывание и масштабирование по мере необходимости.

- Автоматическое масштабирование: Kubernetes автоматически масштабирует количество реплик контейнеров в зависимости от нагрузки, обеспечивая доступность и производительность.

- Сетевая связь: организация сети между контейнерами и управление доступом к приложениям через балансировщики нагрузки;

- Управление конфигурациями: ранение и управление настройками и секретами приложений (например, базы данных, ключи API).

- Автоматическое восстановление: автоматически перезапускает контейнеры в случае их сбоя, а также может перемещать их между узлами кластера для обеспечения доступности.

- Обновления без прерывания работы: поддерживает механизмы обновления приложений (например, rolling updates), которые позволяют обновлять контейнеры без остановки сервиса.

Kubernetes состоит из множества различных компонентов, каждый из которых может быть представлен как контейнер. Некоторые из основных образов Kubernetes включают:

- kube-apiserver: это основной сервер API Kubernetes, через который происходят все взаимодействия с кластером.

- kube-controller-manager: управляет контроллерами в Kubernetes. Это, например, контроллеры для автоматического масштабирования, восстановления состояния и развертывания приложений.

- kube-scheduler: распределяет поды (группы контейнеров) по узлам на основе доступных ресурсов и требований.

- kube-proxy: управляет сетевым взаимодействием внутри кластера, обеспечивая маршрутизацию трафика между подами и внешним миром.

- 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.

Все эти компоненты работают в тесной интеграции для обеспечения надежной работы кластеров, масштабируемости и устойчивости контейнеризованных приложений.