Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра Автоматизированных систем управления

ЛАБОРАТОРНАЯ РАБОТА №6

По дисциплине «ОС Linux»

Контейнеризация

Студент Печенкин Д.В.

Группа ПИ-18

Руководитель

Доцент Кургасов В.В.

Липецк 2021г

Цель работы

Изучение современных методов разработки ПО в динамических и распределенных средах на примере контейнеров Docker.

Задание кафедры

1. С помощью Docker Compose на своем компьютере поднять сборку nginx+php-fpm+postgres, продемонстрировать ее работоспособность, запустив внутри контейнера демо-проект на symfony. По умолчанию проект работает с sqlite-базой. Нужно заменить ее на postgres.

2. Заменить DATABASE_URL в .env на строку подключения к postgres.

3. Создать схему БД и заполнить ее данными из фикстур.

4. Создание образа с Wordpress

Выполнение работы

1. Выполнение работы с проектом demo

1) С помощью команды git clone https://github.com/symfony/demo клонируем проект в папку demo

```
danibrogue@daniel-vb:~$ git clone https://github.com/symfony/demo
Cloning into 'demo'...
remote: Enumerating objects: 9831, done.
remote: Total 9831 (delta 0), reused 0 (delta 0), pack-reused 9831
Receiving objects: 100% (9831/9831), 16.42 MiB | 1.84 MiB/s, done.
Resolving deltas: 100% (5917/5917), done.
danibrogue@daniel-vb:~$ ls
demo      Documents  Music     Public     Videos
Desktop   Downloads  Pictures  Templates
danibrogue@daniel-vb:~$
```

Рисунок 1 – Скачивание проекта demo

2) В проекте demo создаем файл docker-compose.yml и Dockerfile

```
version: "3"
services:
 app:
  container_name: docker-node-mongo
  restart: always
  build: .
  ports:
   - "80:8000"
  links:
   -postgres
 postgres:
  container_name: postgres
  image: postgres
  ports:
   - "5432:5432"
  environment:
   POSTGRES_USER: jacob
   POSTGRES_PASSWORD: 12345
   POSTGRES_DB: jacob_db
  volumes:
   - ./pg-data:/var/lib/postgresql/12/main/base
```

Рисунок 2 – Содержимое файла docker-compose.yml

```
FROM richarvey/nginx-php-fpm
WORKDIR /var>>/www/html/demo
COPY composer.json ./
RUN COMPOSER_MEMORY_LIMIT=-1 composer install
COPY . .
EXPOSE 8000
CMD ["php", "-S", "0.0.0.0:8000", "-t", "public/"]
```

Рисунок 3 – Содержимое файла Dockerfile

3) Изменяем database_url в файле .env для БД, которая будет расположена в будущем контейнере postgres.

```
###> doctrine/doctrine-bundle ###
# Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html#connecting-using-a-url
# For a MySQL database, use: "mysql://db_user:db_password@127.0.0.1:3306/db_name"
# For a PostgreSQL database, use: "postgresql://db_user:db_password@127.0.0.1:5432/db_name?serverVersion=11&charset=utf8"
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
DATABASE_URL="postgresql://jacob:12345@postgres:5432/jacob_db?serverVersion=11&charset=utf8"
```

Рисунок 4 – Изменение файла .env

4) Введем команду “composer install” для скачивания и установки пакетов.(При первом использовании команды, на консоли будет выводится информация о скачиваемых файлах)

```
danibrogue@daniel-vb:~/demo$ composer install
Loading composer repositories with package information
Installing dependencies (including require-dev) from lock file
Package operations: 105 installs, 0 updates, 0 removals
  - Installing composer/package-versions-deprecated (1.11.99.1): Downloading (c
Downloading (100%)
  - Installing symfony/flex (v1.11.0): Downloading (100%)
Enable the "cURL" PHP extension for faster downloads

Prefetching 103 packages 🎶
  - Downloading (100%)

  - Installing doctrine/cache (1.10.2): Loading from cache
  - Installing doctrine/collections (1.6.7): Loading from cache
  - Installing doctrine/event-manager (1.1.1): Loading from cache
  - Installing doctrine/lexer (1.2.1): Loading from cache
  - Installing doctrine/annotations (1.11.1): Loading from cache
  - Installing doctrine/persistence (2.1.0): Loading from cache
  - Installing doctrine/common (3.1.1): Loading from cache
  - Installing symfony/polyfill-php80 (v1.22.0): Loading from cache
  - Installing symfony/deprecation-contracts (v2.2.0): Loading from cache
  - Installing symfony/routing (v5.2.2): Loading from cache
  - Installing symfony/polyfill-mbstring (v1.22.0): Loading from cache
  - Installing symfony/polyfill-php73 (v1.22.0): Loading from cache
  - Installing symfony/polyfill-ctype (v1.22.0): Loading from cache
  - Installing symfony/http-foundation (v5.2.2): Loading from cache
  - Installing symfony/http-client-contracts (v2.3.1): Loading from cache
  - Installing psr/event-dispatcher (1.0.0): Loading from cache
  - Installing symfony/event-dispatcher-contracts (v2.2.0): Loading from cache
```

Рисунок 5 – Установка пакетов

5) Собираем контейнеры командой docker-compose build

```
    1. Install the messenger component by running composer require messenger;
    2. Add 'Symfony\Component\Mailer\Messenger\SendEmailMessage': amqp to the
       config/packages/messenger.yaml file under framework.messenger.routing
       and replace amqp with your transport name of choice.

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


 How to test?


  * Write test cases in the tests/ folder
  * Run php bin/phpunit

Removing intermediate container f20a540a3a20
 ---> 0bb36a373d51
Step 5/7 : COPY . .
 ---> 4b657cde3784
Step 6/7 : EXPOSE 8000
 ---> Running in 2829d571bfd3
Removing intermediate container 2829d571bfd3
 ---> eecd29391fb9
Step 7/7 : CMD ["php", "-S", "0.0.0.0:8000", "-t", "public/"]
 ---> Running in 9582598536f0
Removing intermediate container 9582598536f0
 ---> a02a0fcc46c1
Successfully built a02a0fcc46c1
Successfully tagged demo_app:latest
danibrogue@daniel-vb:~/demo$
```

Рисунок 6 – Сборка контейнеров

6) Запускаем контейнеры командой docker-compose up -d

```
danibrogue@daniel-vb:~/demo$ docker-compose up -d
postgres is up-to-date
Starting docker-node-mongo ... done
danibrogue@daniel-vb:~/demo$
```

Рисунок 7 – Запуск контейнеров

7) Проверим наличие пользователя и базы данных в контейнере postgres

```
danibrogue@daniel-vb:~/demo$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS
         PORTS                                              NAMES
627d5942cf28   demo_app   "docker-php-entrypoi…"   15 minutes ago   Up About a
minute   80/tcp, 443/tcp, 9000/tcp, 0.0.0.0:80->8000/tcp   docker-node-mongo
7a6ea49c26a6   postgres   "docker-entrypoint.s…"   16 minutes ago   Up 15 minut
es       0.0.0.0:5432->5432/tcp                             postgres
danibrogue@daniel-vb:~/demo$ docker exec -it postgres bash
root@7a6ea49c26a6:/# psql -h localhost -p 5432 -U jacob -d jacob_db -W
Password:
psql (13.1 (Debian 13.1-1.pgdg100+1))
Type "help" for help.

jacob_db=#
```

Рисунок 8 – Проверка наличия базы данных в контейнере

8) В каталоге demo создадим схему БД и заполним её командами:

php bin/console doctrine:schema:create

php bin/console doctrine:fixtures:load

После чего зайдем на сервер по адресу localhost:80 и проверим работоспособность

Рисунок 9 – Работоспособность сервера и базы данных.

После закрытия сервера и его повторного включения, загрузилась прежняя база данных с двумя личными записями.

2. Работа с WordPress

WordPress – это система управления контентом. Она позволяет создавать веб приложения для управления сайтами и публиковать контент без знаний программирования. WordPress использует PHP и базу данных MySQL.

1) Создание нового каталога и файла docker-compose.yml

```
 GNU nano 4.8
version: '3.3'
services:
  wordpress:
    image: wordpress:latest
    restart: always
    links:
      - db:mysql
    ports:
      - "80:80"
    working_dir: /var/www/html
    volumes:
      - "/opt/wp-content:/var/www/html/wp-content"
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
  db:
    image: mysql:5.7
    restart: always
    volumes:
      - "/opt/mysql:/var/lib/mysql"
    environment:
      MYSQL_ROOT_PASSWORD: secret
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
```

Рисунок 1 – Содержимое файла docker-compose.yml

2) Создадим каталоги для локального хранения файлов плагинов, контента и базы данных MySQL:

sudo mkdir /opt/mysql
sudo mkdir /opt/wp-content
sudo chmod 777 /opt/wp-content

3) Запустим контейнеры командой docker-compose up -d

```
danibrogue@daniel-vb:~/lab/wordpress$ docker-compose up -d
Creating wordpress_db_1 ... done
Creating wordpress_wordpress_1 ... done
```

Рисунок 2 – Запуск контейнеров

4) Переходим в браузере по адресу localhost:80 и выполняем настройку WordPress.

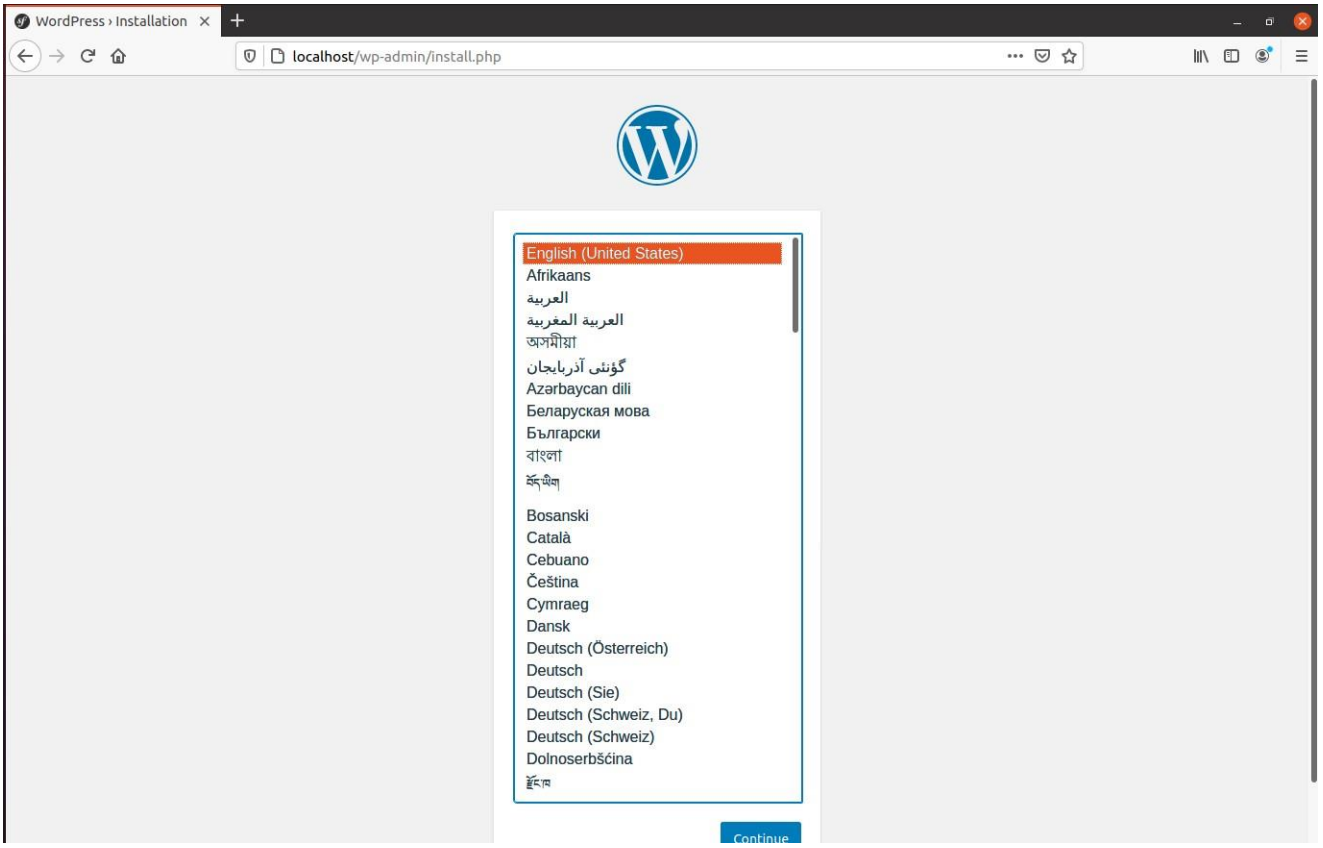

Рисунок 3 – Настройка языка

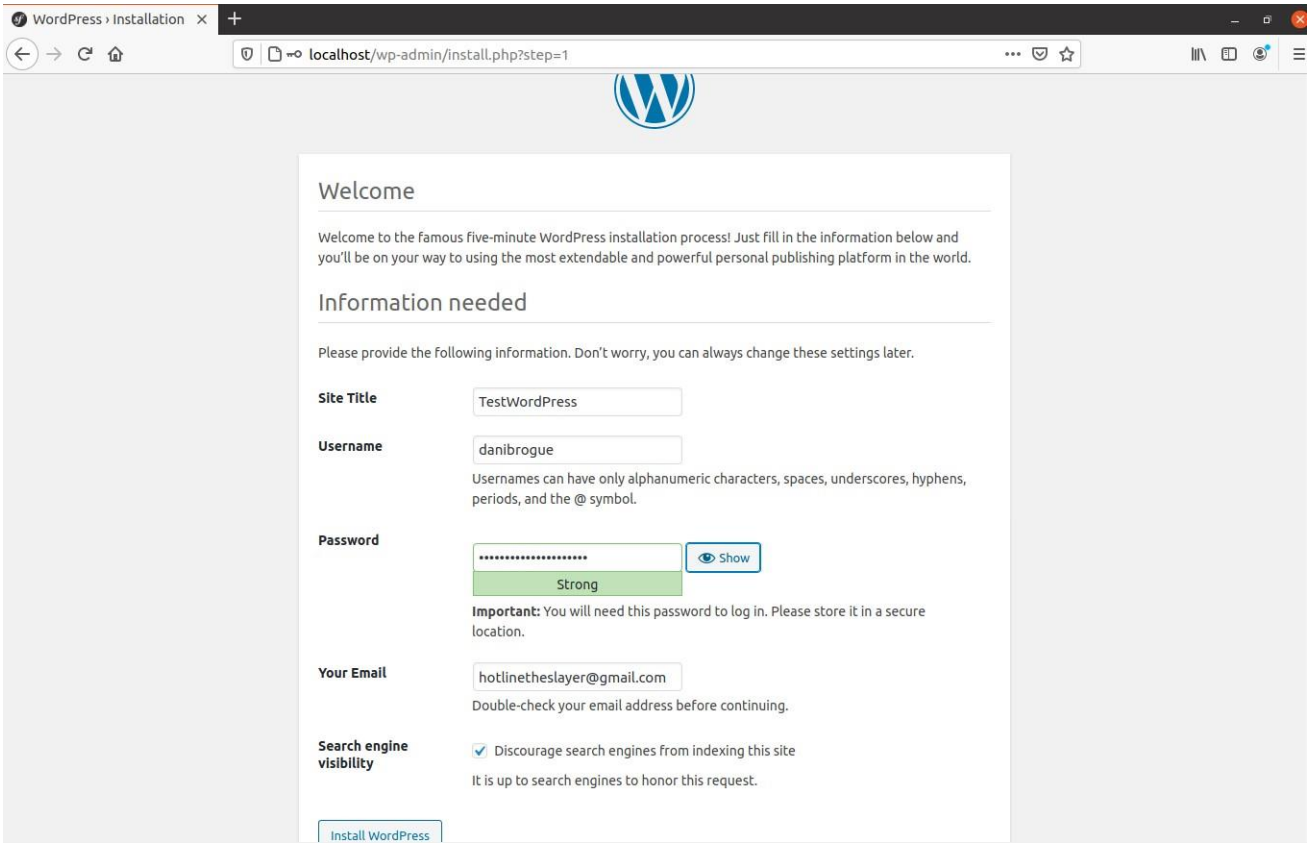

Рисунок 4 – Ввод необходимой информации и регистрация пользователя

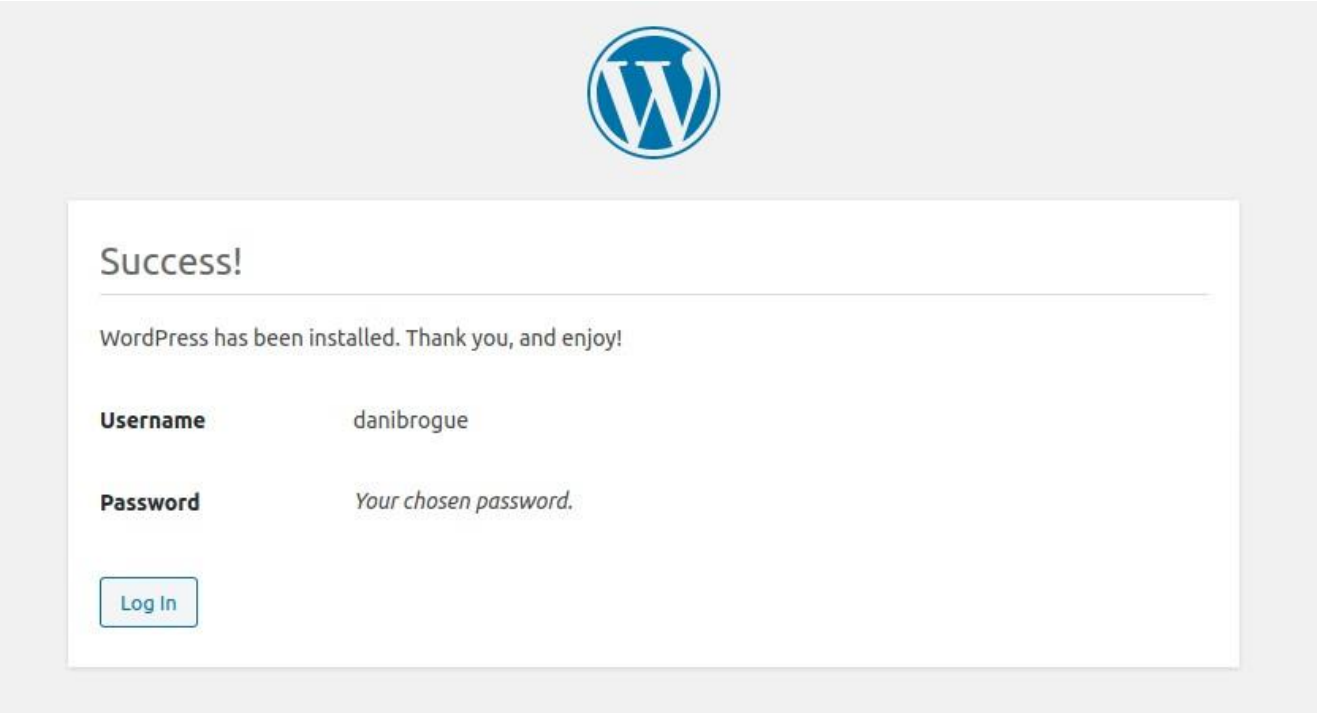

Рисунок 5 – Успешная установка WordPress

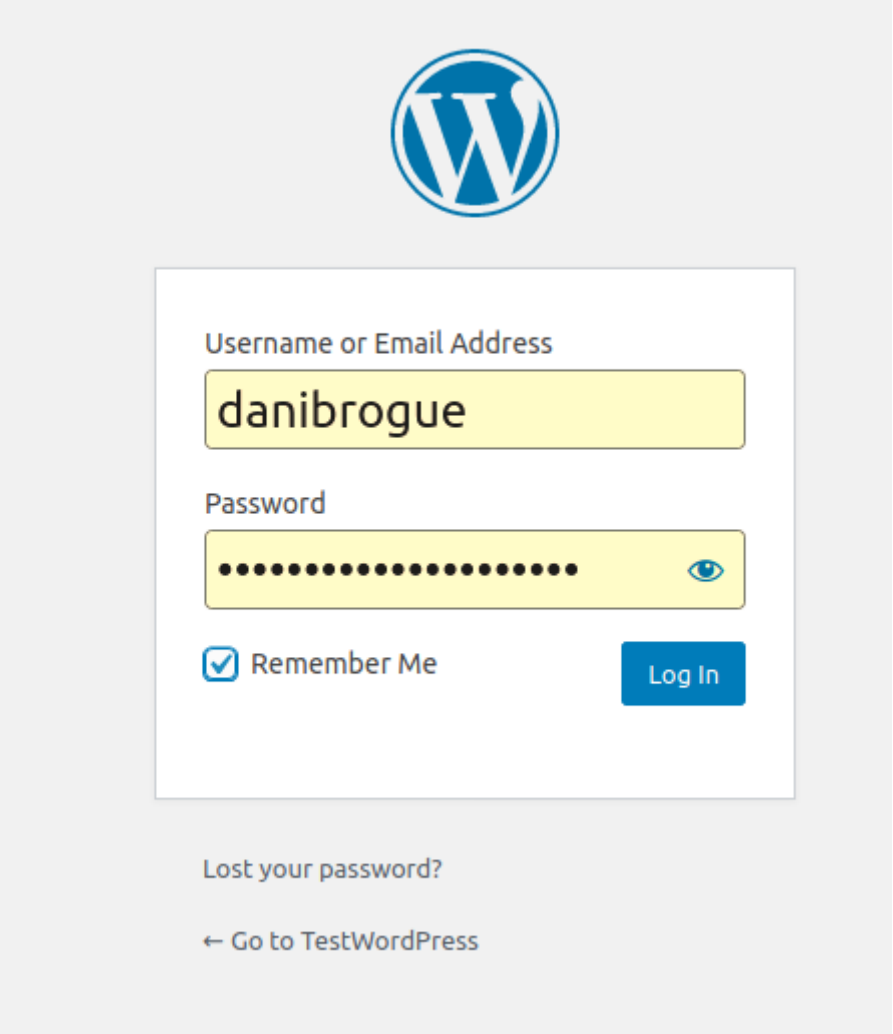

Рисунок 6 - Авторизация

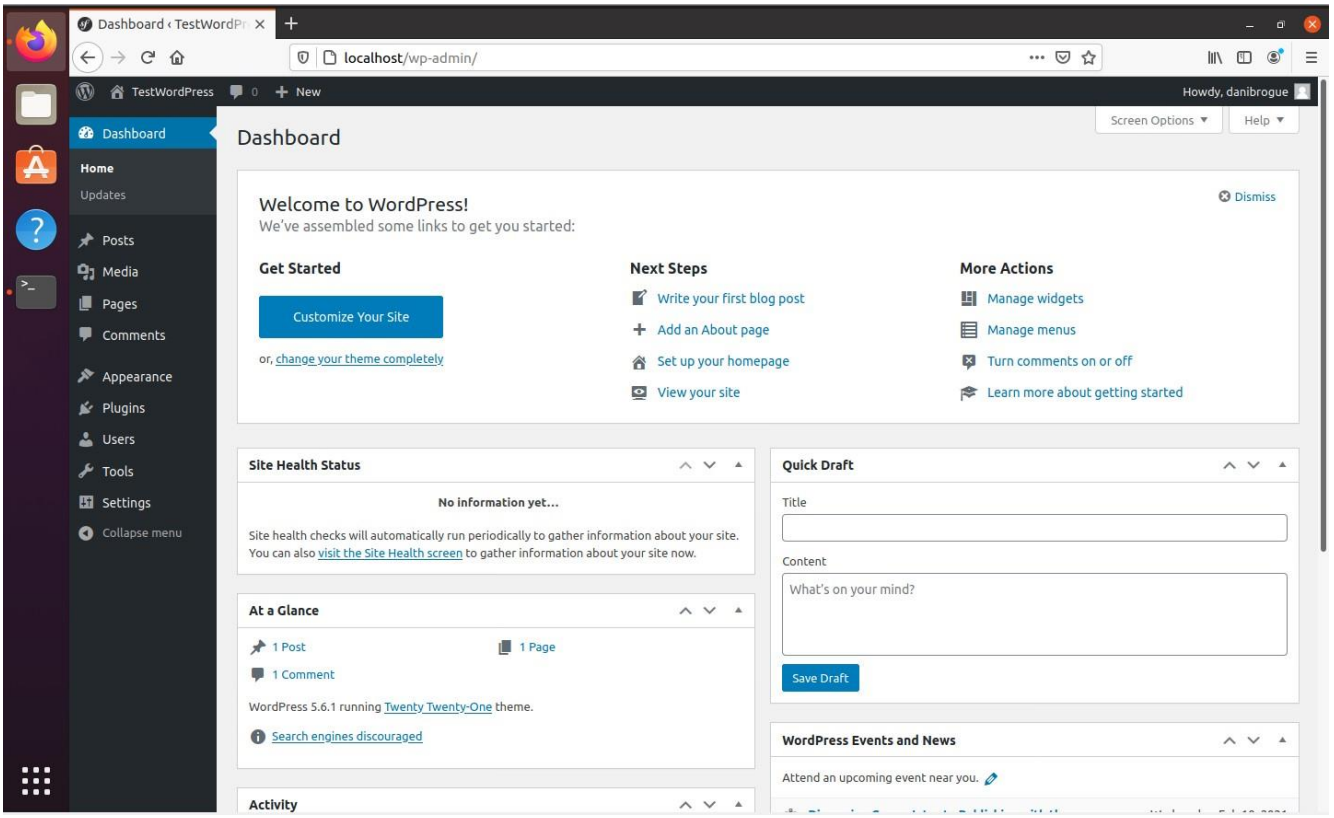

Рисунок 7 – Вход окно настроек сайта

Вывод

В ходе выполнения лабораторной работы были изучены методы разработки ПО в динамических и распределённых средах на примере контейнеров Docker. Были изучены основы работы с системой управления контентом WordPress.

Вопросы для самопроверки:

1. Назовите отличия использования контейнеров по сравнению с виртуализацией:

A) меньшие накладные расходы на инфраструктуру;

B) время старта приложения меньше.

2. Назовите основные компоненты Docker.

C) Образы виртуальных машин

D) Реестры

3. Какие технологии используются для работы с контейнерами?

A) Пространства имен (Linux Namespaces)

C) Контрольные группы (cgroups)

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 – сетевые хранилища образов.

5. Для виртуализации требуется гипервизор, а также полная копия ОС, запускаемое приложение и все библиотеки поддержки. В случае контейнеров ядро-хоста совместно используется работающими контейнерами (это означает, что контейнеры всегда ограничиваются использованием того же ядра, которое функционирует на хосте), а так же процессы внутри контейнеров равнозначны собственными процессами ОС хоста и не влекут за собой дополнительные накладные расходы, связанные с выполнением гипервизора.

6. Основные команды утилиты docker:

docker images (можно посмотреть список образов);

docker ps (вывести список контейнеров);

docker tag (измнение имени образу);

docker rmi (удаление образа);

docker rm (удаление контейнера).

7. Командой docker search <image name> можно запустить поиск Docker образов на сервере регистра с терминала.

8. Командой docker run <image name> осуществляет запуск выбранного образа в новом конетйнере.

9. Контейнер можно быть в 3 состояниях: а) контейнер работает б) контейнер создан, но в настоящий момент не выполняется в) контейнер завершил исполнение

10. Для изоляции контейнера достаточно правильно сконфигурировать файлы Dockerfile и docker-compose.yml. По умолчанию контейнеры запускаются от root прав, поэтому стоит быть осторожным с монтирование томов на хост машину.

11. Последовательность создания нового образа:
Создается файл Dockerfile в корне проекта. Внутри описывается процесс создания образа;
Выполняется сборка образа командой docker build;
Выполняется публикация образа в Registry командой docker push.
Dockerfile — содержит инструкции по созданию образа.

12. Без docker-engine невозможно работать (запускать, изменять и т.п.) с контейнерами docker.

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