ДЗ по " Операционные системы и виртуализация (Linux) (семинары)"

Запуск веб-приложения из контейнеров

Оглавление

Домашнее задание

• Установить в виртуальную машину или VDS Docker, настроить набор контейнеров через docker compose по инструкции. Часть с настройкой certbot и HTTPS опустить, если у вас нет настоящего домена и белого IP.

• (не обязательно) Запустить два контейнера, связанные одной сетью (используя документацию). Первый контейнер БД (например, образ mariadb:10.8), второй контейнер — phpmyadmin. Получить доступ к БД в первом контейнере через второй контейнер (веб-интерфейс phpmyadmin).

Результат

Текст команд, которые применялись при выполнении задания. При наличии: часть конфигурационных файлов, которые решают задачу. Скриншоты результата запуска контейнеров (веб-интерфейс). Присылаем в формате текстового документа: задание и команды для решения (без вывода). Формат — PDF (один файл на все задания).

1 Установка необходимых пакетов

1.1 Установка Docker

Docker — это программная платформа для разработки, доставки и запуска контейнерных приложений. Он позволяет создавать контейнеры, автоматизировать их запуск и развертывание, управляет жизненным циклом. С помощью Docker можно запускать множество контейнеров на одной хост-машине.

```
sudo su

apt install docker.io
```

1.2 Установка docker-compose

Docker Compose — это средство для определения и запуска приложений Docker с несколькими контейнерами. При работе в Compose вы используете файл YAML для настройки служб приложения. Затем вы создаете и запускаете все службы из конфигурации путем выполнения одной команды.

```
apt install docker-compose
```

1.3 Установка yamllint

Валидатор YAML-файлов помогает избежать ошибок. Для Linux можно воспользоваться yamllint. Он наглядно выводит отчет об ошибках.

```
apt install yamllint
```

2 Настройка конфигурации веб-сервера

Перед запуском контейнеров прежде всего необходимо настроить конфигурацию нашего веб-сервера Nginx. Наш файл конфигурации будет включать несколько специфических для Wordpress блоков расположения.

Во-первых, создайте директорию проекта для настройки WordPress с именем `wordpress` и перейдите в эту директорию:

```
mkdir wordpress && cd wordpress
```

Затем создайте директорию для файла конфигурации:

```
mkdir nginx-conf
```

Откройте файл с помощью `nano` или своего любимого редактора:

```
nano nginx-conf/nginx.conf
```

В этом файле мы добавим серверный блок с директивами для имени нашего сервера и корневой директории документов, а также блок обработки PHP и запросов статичных активов.

Добавьте в файл следующий код.

~/wordpress/nginx-conf/nginx.conf

```
server {
    listen 80;
    listen [::]:80;

    server_name glonass-gps.net www.glonass-gps.net;

    index index.php index.html index.htm;

    root /var/www/html;

    location / {
        try_files $uri $uri/ /index.php$is_args$args;
    }

    location ~ \.php$ {
        try_files $uri =404;
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        fastcgi_pass wordpress: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;
    }

    location = /favicon.ico {
        log_not_found off; access_log off;
    }
    location = /robots.txt {
        log_not_found off; access_log off; allow all;
    }
    location ~* \.(css|gif|ico|jpeg|jpg|js|png)$ {
        expires max;
        log_not_found off;
    }
}
```

Наш серверный блок содержит следующую информацию:

Директивы:

`listen`: данный элемент просит Nginx прослушивать порт `80`.

`server_name`: этот элемент определяет имя вашего сервера и серверный блок, которые должны использоваться для запросов к вашему серверу.

`index`: директива `index` определяет файлы, которые будут использоваться в качестве индексов при обработке запросов к вашему серверу. Здесь мы изменили порядок приоритета по умолчанию, поставив `index.php` перед `index.html`, в результате чего Nginx будет давать приоритет файлам с именем `index.php` при наличии возможности.

`root`: наша директива `root` назначает имя корневой директории для запросов к нашему серверу. Эта директория, `/var/www/html`, создается в качестве точки монтирования в момент сборки с помощью инструкций в Dockerfile WordPress. Эти инструкции Dockerfile также гарантируют, что файлы релиза WordPress монтируются в этот том.

Блоки расположения:

`location /`: в этом блоке расположения мы будем использовать директиву `try_files` для проверки файлов, соответствующих отдельным запросам URI. Вместо того, чтобы возвращать по умолчанию статус 404 `не найдено`, мы будем передавать контроль файлу `index.php` Wordpress с аргументами запроса.

`location ~\.php$`: этот блок расположения будет обрабатывать PHP-запросы и проксировать эти запросы в наш контейнер `wordpress`. Поскольку наш образ WordPress Docker будет опираться на образ `php:fpm`, мы также добавим параметры конфигурации, принадлежащие протоколу FastCGI, в этот блок. Nginx требует наличия независимого процессора PHP для запросов PHP: в нашем случае эти запросы будут обрабатываться процессором `php-fpm`, который будет включать образ `php:fpm`. Кроме того, этот блок расположения содержит директивы FastCGI, переменные и опции, которые будут проксировать запросы для приложения WordPress, запущенного в нашем контейнере `wordpress`, задавать предпочитаемый индекс захваченного URI запроса, а также выполнять парсинг URI-запросов.

`location ~ /\.ht`: этот блок будет обрабатывать файлы `.htaccess`, поскольку Nginx не будет обслуживать их. Директива `deny_all` гарантирует, что файлы `.htaccess` никогда не будут отображаться для пользователей.

`location = /favicon.ico`, `location = /robots.txt`: эти блоки гарантируют, что запросы для `/favicon.ico` и `/robots.txt` не будут регистрироваться.

`location ~*\ (css|gif|ico|jpeg|jpg|js|png)$`: этот блок отключает запись в журнал для запросов статичных активов и гарантирует, что эти активы будут иметь высокую кэшируемость, поскольку обычно их трудно обслуживать.

Дополнительную информацию о проксировании FastCGI см. в статье Понимание и реализация проксирования FastCGI в Nginx. Информацию о серверных блоках и блоках расположения см. в статье Знакомство с сервером Nginx и алгоритмы выбора блоков расположения.

Сохраните и закройте файл после завершения редактирования. Если вы используете `nano`, нажмите `CTRL+X`, `Y`, затем `ENTER`.

После настройки конфигурации Nginx вы можете перейти к созданию переменных среды для передачи в контейнеры приложения и базы данных во время исполнения.

3 Настройка переменных среды

Контейнеры базы данных и приложения WordPress должны получить доступ к определенным переменным среды в момент выполнения для сохранения данных приложения и предоставления доступа к этим данным для вашего приложения. Эти переменные включают чувствительные и нечувствительные данные: к чувствительным данным относятся **root**-пароль MySQL и пароль и пользователь базы данных приложения, а к нечувствительным данным относится информация об имени и хосте базы данных приложения.

Вместо того, чтобы задавать эти значения в нашем файле Docker Compose, основном файле, который содержит информацию о том, как наши контейнеры будут работать, мы можем задать чувствительные значения в файле `.env` и ограничить его распространение. Это не позволит скопировать эти значения в репозиторий нашего проекта и открыть их для общего доступа.

В главной директории проекта `~/wordpress`, откройте файл с именем `.env`:

```
nano .env
```

Конфиденциальные значения, которые мы зададим в этом файле, включают пароль для нашего **root**-пользователя MySQL, имя пользователя и пароль, которые WordPress будет использовать для доступа к базе данных.

Добавьте в файл следующие имена и значения переменных:

~/wordpress/.env

```
MYSQL_ROOT_PASSWORD=1
MYSQL_USER=uc
MYSQL_PASSWORD=1
```

Мы включили пароль для административной учетной записи **root**, а также предпочитаемые имя пользователя и пароль для нашей базы данных приложения.

Сохраните и закройте файл после завершения редактирования.

4 Определение служб с помощью Docker Compose

Ваш файл `docker-compose.yml` будет содержать определения службы для вашей настройки. Служба в Compose — это запущенный контейнер, а определения службы содержат информацию о том, как каждый контейнер будет работать.

Используя Compose, вы можете определить различные службы для запуска приложений с несколькими контейнерами, поскольку Compose позволяет привязать эти службы к общим сетям и томам. Это будет полезно для нашей текущей настройки, поскольку мы создадим различные контейнеры для нашей базы данных, приложения WordPress и веб-сервера.

Откройте файл `docker-compose.yml`:

```
nano docker-compose.yml
```

Добавьте следующий код для определения версии файла Compose и базы данных `db`:

~/wordpress/docker-compose.yml

```
version: '3'

services:
  db:
    image: mysql:8.0
    container_name: db
    restart: unless-stopped
    env_file: .env
    environment:
      - MYSQL_DATABASE=wordpress
    volumes:
      - dbdata:/var/lib/mysql
    command: '--default-authentication-plugin=mysql_native_password'
    networks:
      - app-network
```

Определение службы `db` включает следующие параметры:

`image`: данный элемент указывает Compose, какой образ будет загружаться для создания контейнера. Мы закрепим здесь образ `mysql:8.0`, чтобы избежать будущих конфликтов, так как образ `mysql:latest` продолжит обновляться. Дополнительную информацию о закреплении версии и предотвращении конфликтов зависимостей см. в документации Docker в разделе Рекомендации по работе с Dockerfile.

`container_name`: данный элемент указывает имя контейнера.

`restart`: данный параметр определяет политику перезапуска контейнера. По умолчанию установлено значение `no`, но мы задали значение, согласно которому контейнер будет перезапускаться, пока не будет остановлен вручную.

`env_file`: этот параметр указывает Compose, что мы хотим добавить переменные среды из файла с именем `.env`, расположенного в контексте сборки. В этом случае в качестве контекста сборки используется наша текущая директория.

`environment`: этот параметр позволяет добавить дополнительные переменные среды, не определенные в файле `.env`. Мы настроим переменную `MYSQL_DATABASE` со значением `wordpress`, которая будет предоставлять имя нашей базы данных приложения. Поскольку эта информация не является чувствительной, мы можем включить ее напрямую в файл `docker-compose.yml`.

`volumes`: здесь мы монтируем именованный том с названием `dbdata` в директорию `/var/lib/mysql` в контейнере. Это стандартная директория данных в большинстве дистрибутивов.

`command`: данный параметр указывает команду, которая будет переопределять используемое по умолчанию значение инструкции CMD для образа. В нашем случае мы добавим параметр для стандартной команды `mysqld` образа Docker, которая запускает сервер MySQL в контейнере. Эта опция `--default-authentication-plugin=mysql_native_password` устанавливает для системной переменной `--default-authentication-plugin` значение `mysql_native_password`, которое указывает, какой

механизм аутентификации должен управлять новыми запросами аутентификации для сервера. Поскольку PHP и наш образ WordPress не будут поддерживать новое значение аутентификации MySQL по умолчанию, мы должны внести изменения, чтобы выполнить аутентификацию пользователя базы данных приложения.

`networks`: данный параметр указывает, что служба приложения будет подключаться к сети `app-network`, которую мы определим внизу файла.

Затем под определением службы `db` добавьте определение для вашей службы приложения `wordpress`:

~/wordpress/docker-compose.yml

```
...
  wordpress:
    depends_on:
      - db
    image: wordpress:5.1.1-fpm-alpine
    container_name: wordpress
    restart: unless-stopped
    env_file: .env
    environment:
      - WORDPRESS_DB_HOST=db:3306
      - WORDPRESS_DB_USER=$MYSQL_USER
      - WORDPRESS_DB_PASSWORD=$MYSQL_PASSWORD
      - WORDPRESS_DB_NAME=wordpress
    volumes:
      - wordpress:/var/www/html
    networks:
      - app-network
```

В этом определении службы мы называем наш контейнер и определяем политику перезапуска, как уже делали это для службы `db`. Также мы добавляем в этот контейнер ряд параметров:

`depends_on`: этот параметр гарантирует, что наши контейнеры будут запускаться в порядке зависимости, и контейнер `wordpress` запускается после контейнера `db`. Наше приложение WordPress зависит от наличия базы данных приложения и пользователя, поэтому установка такого порядка зависимостей позволит выполнять запуск приложения корректно.

`image`: для этой настройки мы будем использовать образ Wordpress `5.11-fpm-alpine`. Как было показано в шаге 1, использование этого образа гарантирует, что наше приложение будет иметь процессор `php-fpm`, который требуется Nginx для обработки PHP. Это еще и образ `alpine`, полученный из проекта Alpine Linux, который поможет снизить общий размер образа. Дополнительную информацию о преимуществах и недостатках использования образов `alpine`, а также о том, имеет ли это смысл в случае вашего приложения, см. в полном описании в разделе **Варианты образа** на странице образа WordPress на Docker Hub.

`env_file`: и снова мы укажем, что хотим загрузить значения из файла `.env`, поскольку там мы определили пользователя базы данных приложения и пароль.

`environment`: здесь мы будем использовать значения, определенные в файле `.env`, но мы привяжем их к именам переменных, которые требуются для образа WordPress:

`WORDPRESS_DB_USER` и `WORDPRESS_DB_PASSWORD`. Также мы определяем значение `WORDPRESS_DB_HOST`, которое будет указывать сервер MySQL, который будет работать в контейнере `db`, доступный на используемом по умолчанию порту MySQL `3306`. Наше значение `WORDPRESS_DB_NAME` будет тем же, которое мы указали при определении службы MySQL для `MYSQL_DATABASE`: `wordpress`.

`volumes`: мы монтируем том с именем `wordpress` на точку монтирования `/var/www/html`, созданную образом WordPress. Использование тома с именем таким образом позволит разделить наш код приложения с другими контейнерами.

`networks`: мы добавляем контейнер `wordpress` в сеть `app-network`.

Далее под определением службы приложения `wordpress` добавьте следующее определение для службы Nginx `webserver`:

~/wordpress/docker-compose.yml

```
...
webserver:
  depends_on:
   - wordpress
  image: nginx:1.15.12-alpine
  container_name: webserver
  restart: unless-stopped
  ports:
   - "8095:80"
  volumes:
   - wordpress:/var/www/html
   - ./nginx-conf:/etc/nginx/conf.d
  networks:
   - app-network
```

Мы снова присвоим имя нашему контейнеру и сделаем его зависимым от контейнера `wordpress` в отношении порядка запуска. Также мы используем образ `alpine` — образ Nginx `1.15.12-alpine`.

Это определение службы также включает следующие параметры:

`ports`: этот параметр открывает порт `80`, чтобы активировать параметры конфигурации, определенные нами в файле `nginx.conf` в шаге 1.

`volumes`: здесь мы определяем комбинацию названных томов и связанных монтируемых образов:

`wordpress:/var/www/html`: этот параметр будет монтировать код нашего приложения WordPress в директорию `/var/www/html`, директорию, которую мы задали в качестве `root`-директории в нашем серверном блоке Nginx.

`./nginx-conf:/etc/nginx/conf.d`: этот элемент будет монтировать директорию конфигурации Nginx на хост в соответствующую директорию в контейнере, гарантируя, что любые изменения, которые мы вносим в файлы на хосте, будут отражены в контейнере.

Итоговый файл `docker-compose.yml` будет выглядеть примерно так:

~/wordpress/docker-compose.yml

```
version: '3'

services:
  db:
    image: mysql:8.0
    container_name: db
    restart: unless-stopped
    env_file: .env
    environment:
      - MYSQL_DATABASE=wordpress
    volumes:
      - dbdata:/var/lib/mysql
    command: '--default-authentication-plugin=mysql_native_password'
    networks:
      - app-network

  wordpress:
    depends_on:
      - db
    image: wordpress:5.1.1-fpm-alpine
    container_name: wordpress
    restart: unless-stopped
    env_file: .env
    environment:
      - WORDPRESS_DB_HOST=db:3306
      - WORDPRESS_DB_USER=$MYSQL_USER
      - WORDPRESS_DB_PASSWORD=$MYSQL_PASSWORD
      - WORDPRESS_DB_NAME=wordpress
    volumes:
      - wordpress:/var/www/html
    networks:
      - app-network

  webserver:
    depends_on:
      - wordpress
    image: nginx:1.15.12-alpine
    container_name: webserver
    restart: unless-stopped
    ports:
      - "8095:80"
    volumes:
      - wordpress:/var/www/html
      - ./nginx-conf:/etc/nginx/conf.d
    networks:
      - app-network
```

```
volumes:
  wordpress:
  dbdata:

networks:
  app-network:
    driver: bridge
```

Сохраните и закройте файл после завершения редактирования.

После добавления определений службы вы можете запустить контейнеры.

5 Получение учетных данных

Мы можем запустить наши контейнеры с помощью команды `docker-compose up`, которая будет создавать и запускать наши контейнеры и службы в указанном нами порядке.

Создайте контейнеры с помощью команды `docker-compose up` и флага `-d`, которые будут запускать контейнеры `db`, `wordpress` и `webserver` в фоновом режиме:

```
docker-compose up -d
```

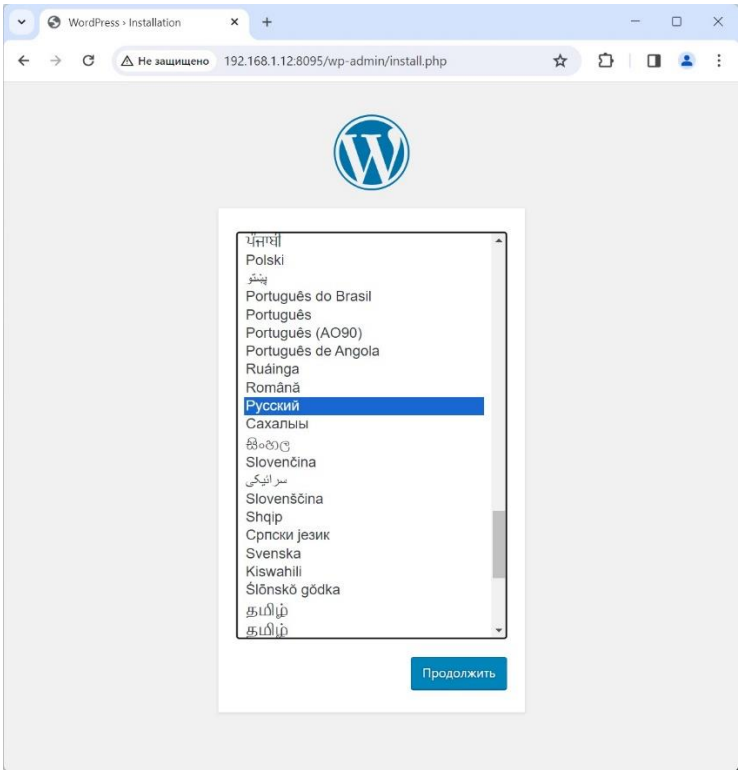
С помощью `docker-compose ps` проверьте статус ваших служб:

```
docker-compose ps
```

6 Завершение установки через веб-интерфейс

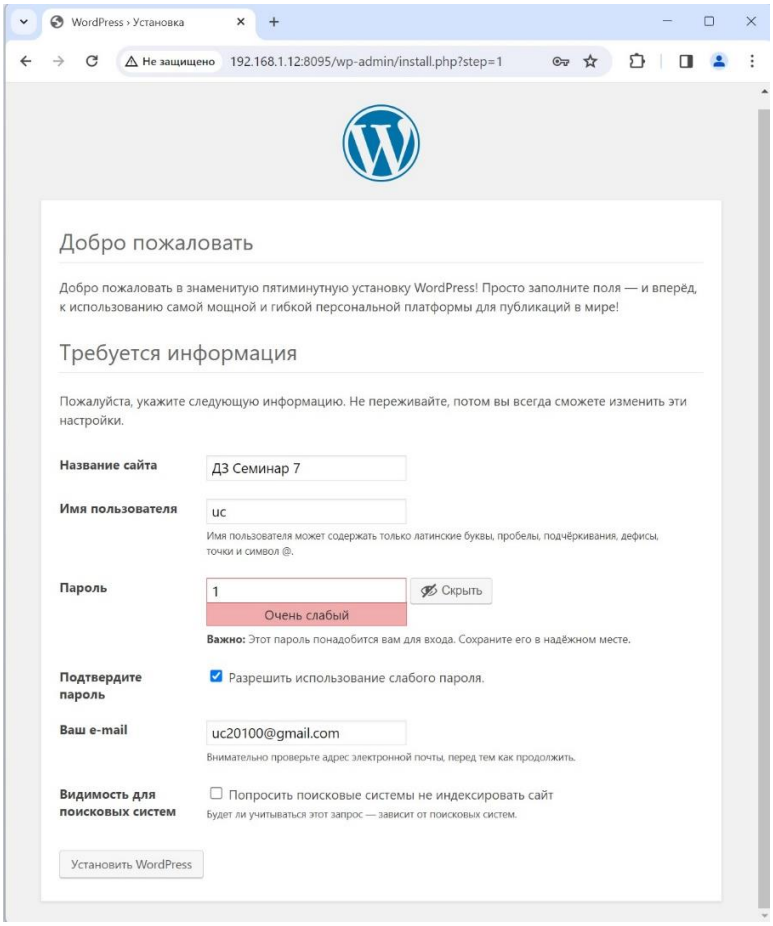
После запуска контейнеров мы можем завершить установку через веб-интерфейс WordPress.

В браузере введите IP Ubuntu и порт 8095, выберите язык, который вы хотите использовать:

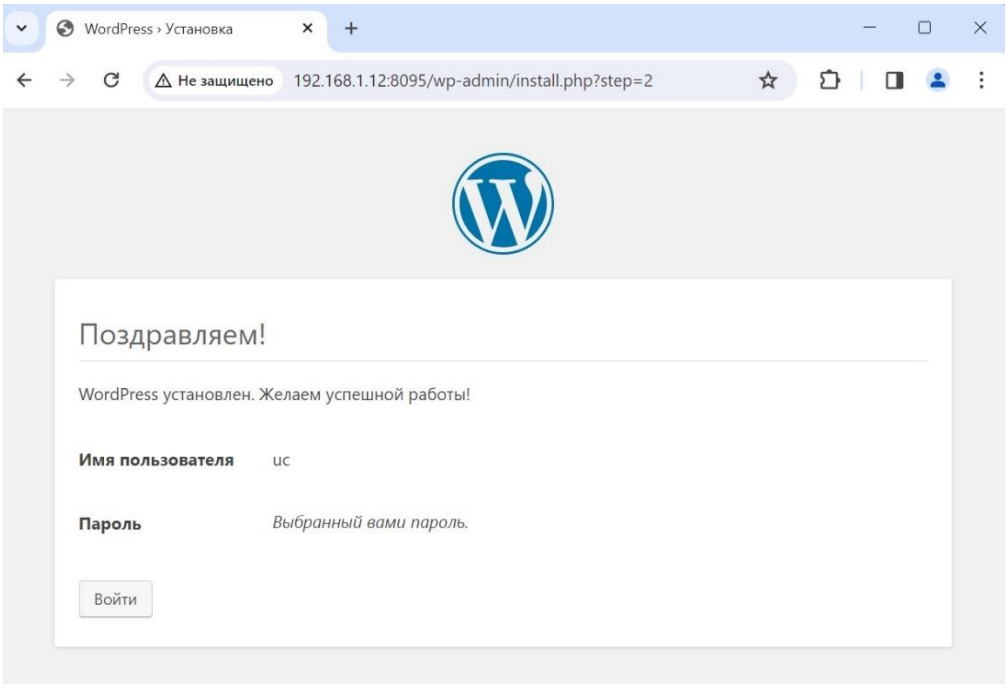


После нажатия **Continue** (Продолжить) вы перейдете на главную страницу настройки, где вам нужно будет выбрать имя вашего сайта и пользователя. Рекомендуется выбрать запоминающееся имя пользователя (не просто «admin») и надежный пароль. Вы можете использовать пароль, который генерирует WordPress автоматически, или задать собственный пароль.

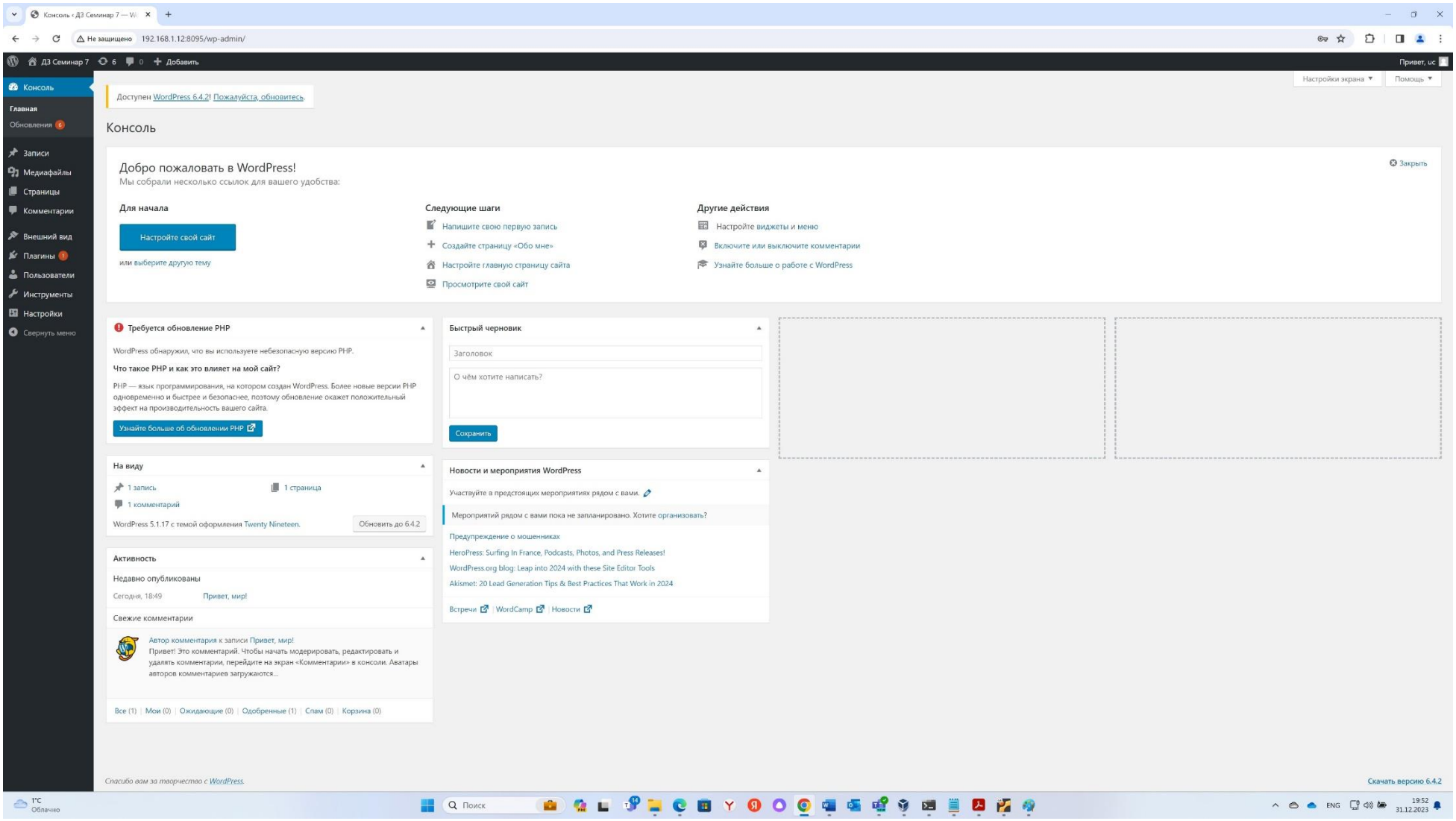
Наконец, вам нужно будет ввести ваш адрес электронной почты и указать, хотите ли вы, чтобы движки поисковых систем могли индексировать ваш сайт:



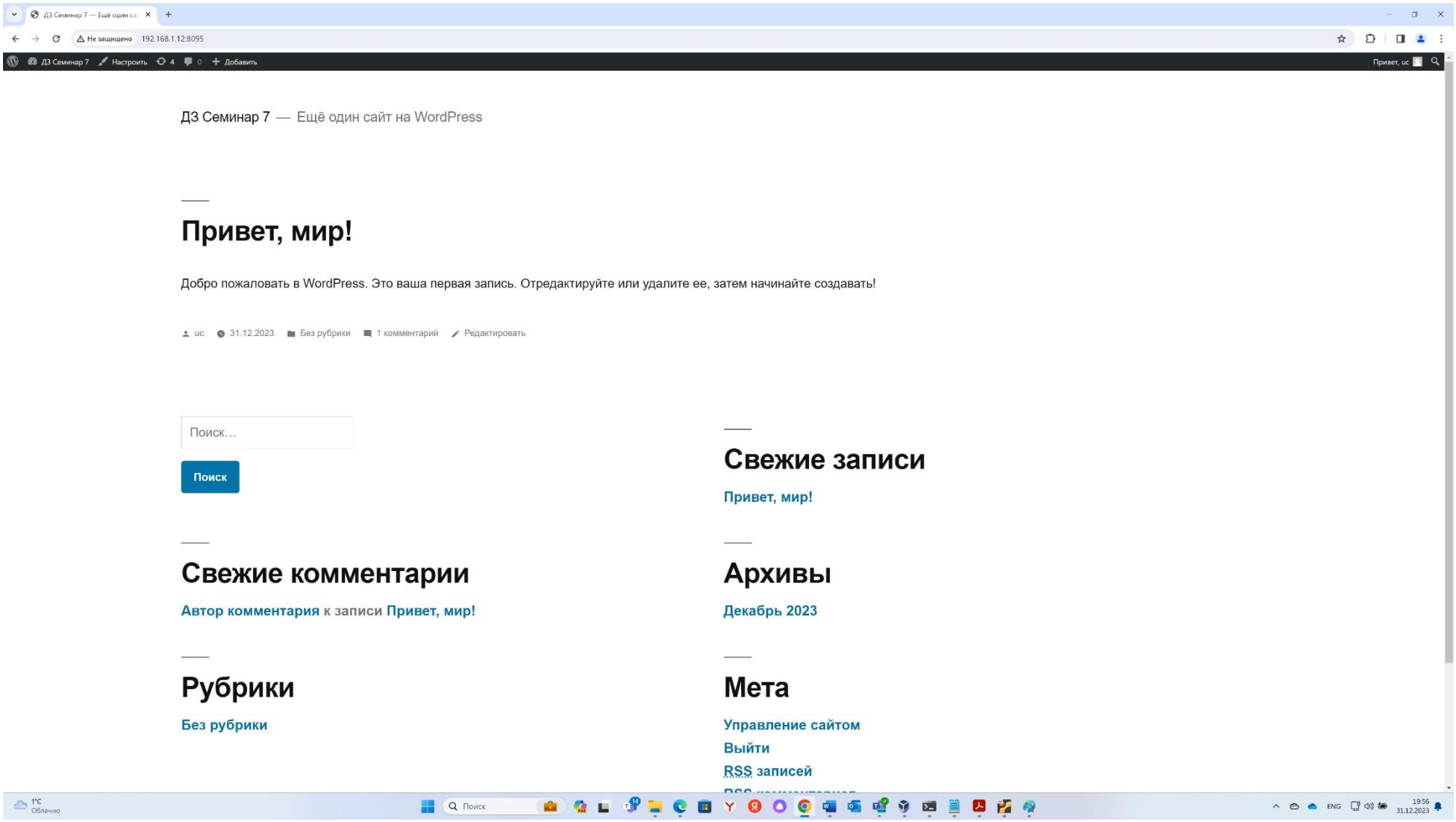
После нажатия **Install WordPress** (Установить Wordpress) внизу страницы на экране появится запрос выполнения входа:



После входа вы получите доступ к панели управления WordPress:



Установка WordPress завершена



Наша стартовая страница в браузере

7 Скриншот терминала PowerShell

Вывод

Выполнив данное ДЗ, я научился устанавливать и настраивать контейнера через docker compose. Настройки certbot для получения SSL сертификата не удалось выполнить, по сколько у меня сейчас нет свободного домена без сертификата, но идея получать бесплатный SSL сертификат мне очень понравилась, нужно будет как ни будь поэкспериментировать – приобрести домен (бесплатный сервис доменных имен сейчас не работает почему-то) и виртуальный сервер с Linux в облаке.

Вторую часть ДЗ (не обязательная), выполню как появиться свободное время, сейчас не успеваю.

Литература

DigitalOcean. (31 12 2023 г.). *Установка WordPress с помощью Docker Compose*. Получено из https://www.digitalocean.com/community/tutorials/how-to-install-wordpress-with-docker-compose-ru

GeekBrains. (31 12 2023 г.). *Запуск веб-приложения из контейнеров (Методичка семинара)*. Получено из https://gbcdn.mrgcdn.ru/uploads/asset/4986578/attachment/900d02ff5340f0d44b9c87ee8b385c9f.pdf

GeekBrains. (31 12 2023 г.). *Запуск веб-приложения из контейнеров (Семинар)*. Получено из https://gbcdn.mrgcdn.ru/uploads/record/301804/attachment/b54a13e347f644f57bbf3fe9d2de32af.mp4

GeekBrains. (31 12 2023 г.). *Основы Docker (Конспект)*. Получено из https://gbcdn.mrgcdn.ru/uploads/asset/4327231/attachment/d6dde88f89db1e0291031e16604e1d7f.pdf

GeekBrains. (31 12 2023 г.). *Основы Docker (Лекция)*. Получено из https://gbcdn.mrgcdn.ru/uploads/record/246041/attachment/62ff4db2685bdda02c653242b923e868.mp4

GeekBrains. (31 12 2023 г.). *Основы Docker (Презентация)*. Получено из https://gbcdn.mrgcdn.ru/uploads/asset/4327228/attachment/b4be88c35f243f228bb0bce50341edb7.pdf

Хабр. (31 12 2023 г.). *Let's Encrypt и nginx: настройка в Debian и Ubuntu*. Получено из https://habr.com/ru/articles/318952/