

Установка WordPress с помощью Docker Compose в Centos 7.

WordPress — бесплатная система управления контентом (CMS) с открытым исходным кодом, которая опирается на базу данных MySQL и обрабатывает запросы с помощью PHP. Благодаря огромному количеству плагинов и системе шаблонов, а также тому факту, что большая часть административных функций может производиться через веб-интерфейс, WordPress завоевала популярность среди создателей самых разных сайтов, от блогов и страниц с описанием продукта и до сайтов электронной торговли.

Для запуска WordPress, как правило, требуется установка стека LAMP Linux, Apache, MySQL и PHP или LEMP Linux, Nginx, MySQL и PHP, что может занять много времени. С помощью таких инструментов, как Docker и Docker Compose, вы можете упростить процесс настройки предпочитаемого стека и установки WordPress.

Установка Docker:

1. Установите необходимые пакеты:

```
sudo yum update
sudo yum install yum-utils device-mapper-persistent-data lvm2
```

2. Добавьте репозиторий Docker:

```
sudo yum-config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo
```

3. Установите Docker CE:

```
sudo yum install docker-ce
```

4. Запустите службу Docker:

```
sudo systemctl start docker
sudo systemctl enable docker
```

5. Добавьте пользователя в группу Docker:

```
sudo usermod -aG docker $USER
```

6. Проверка установки:

```
docker run hello-world
```

```
[root@localhost ~]# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:53641cd209a4fecfc68e21a99871ce8c6920b2e7502df0a20671c6fccc73a
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Установка Docker Compose.

Скачайте последнюю версию Docker Compose.

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Сделайте файл исполняемым:

```
sudo chmod +x /usr/local/bin/docker-compose
```

Проверьте, что Docker Compose установлен успешно

```
docker-compose -version
```

```
[root@wp-server ~]# docker-compose --version
docker-compose version 1.29.2, build 5becea4c
[root@wp-server ~]#
```

3.* Зарегистрированное доменное имя и настроены записи DNS.

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 наряду с блоками расположения, которые будут направлять передаваемые Let's Encrypt запросы верификации клиенту Certbot для автоматизированного обновления сертификатов.

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 расположения для направления запросов сертификатов от клиента Certbot, обработки PHP и запросов статичных активов.

Добавьте в файл следующий код. Обязательно замените example.com на ваше доменное имя.

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

    server_name example.com www.example.com;

    index index.php index.html index.htm;

    root /var/www/html;

    location ~ /\.well-known/acme-challenge {
        allow all;
        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, что позволит нам использовать плагин webroot Certbot для наших запросов сертификатов. Обратите внимание, что мы пока *не* будем включать порт 443, мы обновим нашу конфигурацию и добавим SSL после успешного получения наших сертификатов.
- **server_name:** этот элемент определяет имя вашего сервера и серверный блок, которые должны использоваться для запросов к вашему серверу. Обязательно замените example.com в этой строке на ваше собственное доменное имя.
- **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 ~ /\.well-known/acme-challenge:** этот блок расположения будет обрабатывать запросы в директории .well-known, где Certbot будет размещать временный файл для подтверждения того, что DNS для нашего домена будет работать **с нашим сервером**. **Настроив данную конфигурацию**, мы сможем использовать плагин webroot Certbot для получения сертификатов для нашего домена.
- **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.

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`:

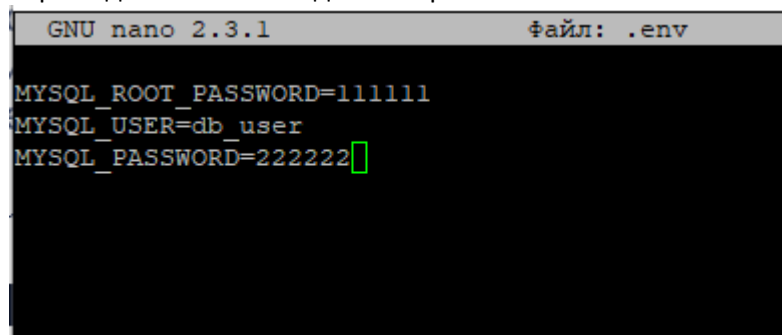
```
nano .env
```

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: Обязательно предоставьте здесь **ваши собственные значения** для каждой переменной:

```
MYSQL_ROOT_PASSWORD=your_root_password
MYSQL_USER=your_wordpress_database_user
MYSQL_PASSWORD=your_wordpress_database_password
```

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

A screenshot of the GNU nano 2.3.1 text editor. The title bar shows "GNU nano 2.3.1" and "файл: .env". The editor content shows three lines of text: "MYSQL_ROOT_PASSWORD=111111", "MYSQL_USER=db_user", and "MYSQL_PASSWORD=222222". A green cursor is positioned at the end of the third line.

```
GNU nano 2.3.1                файл: .env

MYSQL_ROOT_PASSWORD=111111
MYSQL_USER=db_user
MYSQL_PASSWORD=222222
```

Сохраните и закройте файл после завершения редактирования.

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. Также мы создадим контейнер для запуска клиента Certbot, чтобы получить сертификаты для нашего веб-сервера.

Откройте файл **docker-compose.yml**:

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

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

```
version: '3'

services:
  db:
    image: mysql:8.3
    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:8.3`, 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:
  depends_on:
    - db
  image: wordpress:6.4.3-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 6.4.3-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:

```
webserver:
  depends_on:
    - wordpress
  image: nginx:1.25.4-alpine
  container_name: webserver
  restart: unless-stopped
  ports:
```

```

- "80:80"
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 1.25.4-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.

Здесь мы снова добавили этот контейнер в сеть app-network.

Добавьте определения сети и тома:

```

volumes:
  wordpress:
  dbdata:

networks:
  app-network:
    driver: bridge

```

Наш ключ верхнего уровня volumes определяет тома wordpress и dbdata. Когда Docker создает тома, содержимое тома сохраняется в директории файловой системы хоста, /var/lib/docker/volumes/, а данным процессом управляет Docker. После этого содержимое каждого тома монтируется из этой директории в любой контейнер, использующий том. Таким образом мы можем делиться кодом и данными между контейнерами. Создаваемая пользователем мостовая система app-network позволяет организовать коммуникацию между нашими контейнерами, поскольку они находятся на одном хосте демона Docker. Это позволяет организовать трафик и коммуникации внутри приложения, поскольку она открывает все порты между контейнерами в одной мостовой сети, скрывая все порты от внешнего мира. Таким образом, наши контейнеры db, wordpress и webserver могут взаимодействовать друг с другом, и нам нужно будет только открыть порт 80 для внешнего доступа к приложению.

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

```

version: '3'

services:
  db:
    image: mysql:8.3
    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:6.4.3-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:1.25.4-alpine
  container_name: webserver
  restart: unless-stopped
  ports:
    - "80:80"
  volumes:
    - wordpress:/var/www/html
    - ./nginx-conf:/etc/nginx/conf.d

  networks:
    - app-network

volumes:
  wordpress:
  dbdata:

networks:
  app-network:
    driver: bridge

```

Мы можем запустить наши контейнеры с помощью команды **docker-compose up**, которая будет создавать и запускать наши контейнеры и службы в указанном нами порядке. Если наши запросы доменов будут выполнены успешно, мы увидим корректный статус выхода в нашем выводе и нужные сертификаты, установленные в папке /etc/letsencrypt/live на контейнере webserver.

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
```

```

Digest: sha256:31bad00311cb5eeb8a6648beadcf67277a175da89989f
Status: Downloaded newer image for nginx:1.25.4-alpine
Creating db ... done
Creating wordpress ... done
Creating webserver ... done

```


С помощью docker-compose ps проверьте статус ваших служб:

```
docker-compose ps
```

```
[root@wp-server wordpress]# docker-compose ps
      Name                Command                                State      Ports
-----
db            docker-entrypoint.sh --def ...        Up         3306/tcp, 33060/tcp
webserver     /docker-entrypoint.sh nginx ...      Up         0.0.0.0:80->80/tcp, :::80->80/tcp
wordpress     docker-entrypoint.sh php-fpm          Up         9000/tcp
[root@wp-server wordpress]#
```

Если все будет выполнено успешно, ваши службы db, wordpress и webserver должны иметь статус Up
Если вы увидите любое значение, кроме Up в столбце State для служб db, wordpress и webserver, проверьте журналы службы с помощью команды docker-compose logs:

```
docker-compose logs service_name
```

Далее нужно разрешить http-трафик в файрволле:

```
sudo firewall-cmd --permanent --add-service=http --zone=public
sudo firewall-cmd -reload
sudo firewall-cmd -list-all
```

Теперь можно проверить, что wordpress работает:

```
[root@wp-server ~]# ip a s enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:40:93:14 brd ff:ff:ff:ff:ff:ff
    inet 10.10.10.102/24 brd 10.10.10.255 scope global noprefixroute dynamic enp0s3
        valid_lft 28577sec preferred_lft 28577sec
    inet6 fe80::1639:b6b1:2b15:bbc2/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
[root@wp-server ~]#
```

