Настройка веб-сервера в Docker (NGINX + PHP + MariaDB)

По данной инструкции мы настроим наш веб сервер в Docker следующим образом:

* В качестве веб-сервера будет использоваться NGINX.
* Будет поддержка приложений на PHP.
* Мы создадим 2 контейнера Docker — один для NGINX + PHP, второй для СУБД (MariaDB).
* Веб-приложение будет зашито в контейнере. Раздел для данных MariaDB будет монтироваться в качестве volume.

Данную конфигурацию можно использовать для быстрого развертывания сайтов или для локальной разработки.

Мы будем работать в системе на базе Linux. Предполагается, что Docker уже установлен.

NGINX + PHP + PHP-FPM

Рекомендуется каждый микросервис помещать в свой отдельный контейнер, но мы (для отдельного примера) веб-сервер с интерпретатором PHP поместим в один и тот же имидж, на основе которого будут создаваться контейнеры.

Создание образа

Создадим каталог, в котором будут находиться файлы для сборки образа веб-сервера:

mkdir -p /opt/docker/web-server

Переходим в созданный каталог:

cd /opt/docker/web-server/

Создаем докер-файл:

vi Dockerfile

|  |  |
| --- | --- |
| 1. | FROM centos:8 |
| 2. |  |
| 3. | MAINTAINER Docker Boss <mail@site.domain> |
| 4. |  |
| 5. | ENV TZ=Europe/Moscow |
| 6. |  |
| 7. | RUN dnf update -y |
| 8. | RUN dnf install -y nginx php php-fpm php-mysqli |
| 9. | RUN dnf clean all |
| 10. | RUN echo "daemon off;" >> /etc/nginx/nginx.conf |
| 11. | RUN mkdir /run/php-fpm |
| 12. |  |
| 13. | COPY ./html/ /usr/share/nginx/html/ |
| 14. |  |
| 15. | CMD php-fpm -D ; nginx |
| 16. |  |
| 17. | EXPOSE 80\* где: |

*1) указываем, какой берем базовый образ. В нашем случае, CentOS 8.  
3) инструкция* MAINTAINER*, сообщает Docker автора образа и его email. Это полезно, чтобы пользователи образа могли связаться с автором при необходимости.  
3) задаем для информации того, кто создал образ. Указываем свое имя и адрес электронной почты.  
5) создаем переменную окружения****TZ****с указанием временной зоны (в нашем примере, московское время).  
7) запускаем обновление системы.  
8) устанавливаем пакеты: веб-сервер****nginx****, интерпретатор****php****, сервис****php-fpm****для обработки скриптов, модуль****php-mysqli****для работы php с СУБД MySQL/MariaDB.  
9) удаляем скачанные пакеты и временные файлы, образовавшиеся во время установки.  
10) добавляем в конфигурационный файл nginx строку****daemon off****, которая запретит веб-серверу автоматически запуститься в качестве демона.  
11) создаем каталог****/run/php-fpm****— без него не сможет запуститься php-fpm.  
13) копируем содержимое каталога****html****, который находится в том же каталоге, что и dockerfile, в каталог****/usr/share/nginx/html/****внутри контейнера. В данной папке должен быть наше веб-приложение.  
15) запускаем****php-fpm****и****nginx****. Команда CMD в dockerfile может быть только одна.  
17) открываем порт****80****для работы веб-сервера.*

В рабочем каталоге создаем папку html:

mkdir html

... а в ней — файл index.php:

vi html/index.php

<?php  
  
phpinfo();  
  
?>

*\* мы создали скрипт, который будет выводить информацию о php в браузере для примера. По идее, в данную папку мы должны положить сайт (веб-приложение).*

Создаем первый билд для нашего образа:

docker build -t dmosk/webapp:v1 .

Новый образ должен появиться в системе:

docker images

При желании, его можно отправить на Docker Hub следующими командами:

docker login --username username

docker tag username/webapp:v1 dmosk/web:nginx\_php7

docker push username/web:nginx\_php7

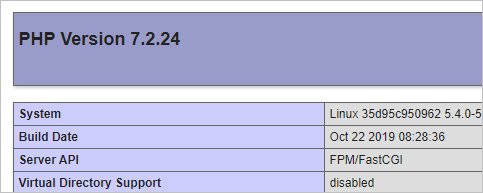
*\* первой командой мы прошли аутентификацию на портале докер-хаба (в качестве id/login мы используем****username****— это учетная запись, которую мы зарегистрировали в Docker Hub). Вторая команда создает тег для нашего образа, где****username****— учетная запись на dockerhub;****web****— имя репозитория;****nginx\_php7****— сам тег. Последняя команда заливает образ в репозиторий.*

Запуск контейнера и проверка работы

Запускаем веб-сервер из созданного образа:

docker run --name web\_server -d -p 80:80 dmosk/webapp:v1

Открываем браузер и переходим по адресу http://<IP-адрес сервера с docker> — откроется страница phpinfo:



Наш веб-сервер из Docker работает.

MariaDB

Для запуска СУБД мы будем использовать готовый образ mariadb. Так как после остановки контейнера, все данные внутри него удаляются, мы должны подключить внешний том, на котором будут храниться наши базы.

Сначала создаем том для докера:

docker volume create --name mariadb

*\* в данном примере мы создали том с именем mariadb. Будет создан каталог****/var/lib/docker/volumes/mariadb/\_data/****на хостовом сервере, куда будут размещаться наши файлы базы.*

Выполним первый запуск нашего контейнера с mariadb:

docker run --rm --name maria\_db -d -e MYSQL\_ROOT\_PASSWORD=password -v mariadb:/var/lib/mysql mariadb

*\* где:*

* ***--rm —****удалить контейнер после остановки. Это первый запуск для инициализации базы, после параметры запуска контейнера будут другими.*
* ***--name maria\_db —****задаем имя контейнеру, по которому будем к нему обращаться.*
* ***-e MYSQL\_ROOT\_PASSWORD=password —****создаем системную переменную, содержащую пароль для пользователя root базы данных. Оставляем его таким, как в данной инструкции, так как следующим шагом мы его будем менять.*
* ***-v mariadb:/var/lib/mysql —****говорим, что для контейнера мы хотим использовать том mariadb, который будет примонтирован внутри контейнера по пути /var/lib/mysql.*
* ***mariadb —****в самом конце мы указываем имя образа, который нужно использовать для запуска контейнера. Это образ, который при первом запуске будет скачан с DockerHub.*

В каталоге тома должны появиться файлы базы данных. В этом можно убедиться командой:

ls /var/lib/docker/volumes/mariadb/\_data/

Теперь подключаемся к командной строке внутри контейнера с сервером базы данных:

docker exec -it maria\_db /bin/bash

Подключаемся к mariadb:

:/# mysql -ppassword

Меняем пароль для учетной записи root:

> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('New\_Password');

Выходим из командной строки СУБД:

> quit

Выходим из контейнера:

:/# exit

Останавливаем сам контейнер — он нам больше не нужен с данными параметрами запуска:

docker stop maria\_db

И запускаем его по новой, но уже без системной переменной с паролем и необходимостью его удаления после остановки:

docker run --name maria\_db -d -v mariadb:/var/lib/mysql mariadb

Сервер баз данных готов к работе.

Подключение к базе из веб-сервера

По отдельности, наши серверы готовы к работе. Теперь настроим их таким образом, чтобы из веб-сервера можно было подключиться к СУБД.

Зайдем в контейнер с базой данных:

docker exec -it maria\_db /bin/bash

Подключимся к mariadb:

:/# mysql -p

Создадим базу данных, если таковой еще нет:

> CREATE DATABASE docker\_db DEFAULT CHARACTER SET utf8 DEFAULT COLLATE utf8\_general\_ci;

*\* в данном примере мы создаем базу****docker\_db****.*

Создаем пользователя для доступа к нашей базе данных:

> GRANT ALL PRIVILEGES ON docker\_db.\* TO 'docker\_db\_user'@'%' IDENTIFIED BY 'docker\_db\_password';

*\* и так, мы дали полные права на базу****docker\_db****пользователю****docker\_db\_user****, который может подключаться от любого хоста (****%****). Пароль для данного пользователя —****docker\_db\_password****.*

Отключаемся от СУБД:

> quit

Выходим из контейнера:

:/# exit

Теперь перезапустим наши контейнеры с новым параметром, который будет объединять наши контейнеры по внутренней сети.

Останавливаем работающие контейнеры и удаляем их:

docker stop maria\_db web\_server

docker rm maria\_db web\_server

Создаем docker-сеть:

docker network create net1

*\* мы создали сеть net1.*

Создаем новые контейнеры из наших образов и добавляем опцию **--net**, которая указывает, какую сеть будет использовать контейнер:

docker run --name maria\_db --net net1 -d -v mariadb:/var/lib/mysql mariadb

docker run --name web\_server --net net1 -d -p 80:80 dmosk/webapp:v1

*\* указав опцию****--net****, наши контейнеры начинают видеть друг друга по своим именам, которые мы задаем опцией****--name****.*

Готово. Для проверки соединения с базой данных в php мы можем использовать такой скрипт:

<?php  
  
ini\_set("display\_startup\_errors", 1);  
ini\_set("display\_errors", 1);  
ini\_set("html\_errors", 1);  
ini\_set("log\_errors", 1);  
error\_reporting(E\_ERROR | E\_PARSE | E\_WARNING);  
  
$con = mysqli\_connect('maria\_db', 'docker\_db\_user', 'docker\_db\_password', 'docker\_db');  
  
?>

*\* в данном примере мы подключаемся к базе****docker\_db****на сервере****maria\_db****с использованием учетной записи****docker\_db\_user****и паролем****docker\_db\_password****.*

После его запуска, мы увидим либо пустой вывод (если подключение выполнено успешно), либо ошибку.

Использование docker-compose

В отличие от docker, с помощью docker-compose можно разворачивать проекты, состоящие из нескольких контейнеров, одной командой.

И так, автоматизируем запуск наших контейнеров с использованием docker-compose. Необходимо, чтобы он был установлен в системе.

Сначала удалим контейнеры, которые создали на предыдущих этапах:

docker rm -f web\_server maria\_db

Переходим в каталог для наших сборок:

cd /opt/docker/

Создаем yml-файл с инструкциями сборки контейнеров через docker-compose:

vi docker-compose.yml

---  
  
version: "3.8"  
  
services:  
  
  web\_server:  
    build:  
      context: ./web-server/  
      args:  
        buildno: 1  
    container\_name: web\_server  
    restart: always  
    environment:  
      TZ: "Europe/Moscow"  
    ports:  
      - 80:80  
  
  maria\_db:  
    image: mariadb  
    container\_name: maria\_db  
    restart: always  
    environment:  
      TZ: "Europe/Moscow"  
    volumes:  
      - /var/lib/docker/volumes/mariadb/\_data/:/var/lib/mysql

*\* в формате yml очень важное значение имеют отступы. Если сделать лишний пробел, то мы получим ошибку.  
\* где:*

* ***version****— версия файла yml. На странице*[*docs.docker.com*](https://docs.docker.com/compose/compose-file/)*представлена таблица, позволяющая понять, какую версию лучше использовать, в зависимости от версии docker (docker -v).*
* ***services****— docker-compose оперирует сервисами, где для каждого создается свой блок описания. Все эти блоки входят в раздел services.*
* ***build****— опции сборки. В нашем примере для веб-сервера мы должны собрать имидж.*
* ***context****— указываем путь до Dockerfile.*
* ***args****— позволяет задать аргументы, которые доступны только в процессе сборки. В данном примере мы используем только аргумент с указанием номера сборки.*
* ***container\_name****— задаем имя, которое будет задано контейнеру после его запуска.*
* ***restart****— режим перезапуска. В нашем случае всегда, таким образом, после перезагрузки сервера, наши контейнеры запустятся.*
* ***ports****— при необходимости, указываем порт, который будет наш сервер пробрасывать запрос внутрь контейнера.*
* ***environment****— задает системные переменные. В данном примере, временную зону.*
* ***volumes****— позволяет внутрь контейнера прокинуть каталог сервера. Таким образом, важные данные не будут являться частью контейнера и не будут удалены после его остановки.*

Запускаем сборку наших контейнеров с помощью docker-compose:

docker-compose build

Запускаем контейнеры в режиме демона:

docker-compose up -d

Проверяем, какие контейнеры запущены:

docker ps

При внесении изменений можно перезапускать контейнеры командой:

docker-compose up --force-recreate --build -d

Просто пересоздать контейнеры:

docker-compose restart

или только для одного из сервисов:

docker-compose restart web\_server

*\* где****web\_server —****название сервиса в файле docker-compose.*

Возможные проблемы

Рассмотрим некоторые проблемы, которые могут возникнуть в процессе настройки.

1. Errors during downloading metadata for repository 'AppStream'

Ошибка возникает при попытке собрать имидж на Linux CentOS 8. Полный текст ошибки может быть такой:

Errors during downloading metadata for repository 'AppStream':  
  - Curl error (6): Couldn't resolve host name for http://mirrorlist.centos.org/?release=8&arch=x86\_64&repo=AppStream&infra=container [Could not resolve host: mirrorlist.centos.org]  
Error: Failed to download metadata for repo 'AppStream': Cannot prepare internal mirrorlist: Curl error (6): Couldn't resolve host name for http://mirrorlist.centos.org/?release=8&arch=x86\_64&repo=AppStream&infra=container [Could not resolve host: mirrorlist.centos.org]

**Причина:** система внутри контейнера не может разрешить dns-имена в IP-адрес.

**Решение:** в CentOS 8 запросы DNS могут блокироваться брандмауэром, когда в качестве серверной части (backend) стоит nftables. Переключение на iptables решает проблему. Открываем файл:

vi /etc/firewalld/firewalld.conf

Находим строку:

FirewallBackend=nftables

... и меняем ее на:

FirewallBackend=iptables

Перезапускаем сервис firewalld:

systemctl restart firewalld