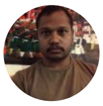
WordPress Deployment with NGINX, PHP-FPM and MariaDB using Docker Compose


Barani Murthy Follow
Oct 11, 2019 · 6 min read ★

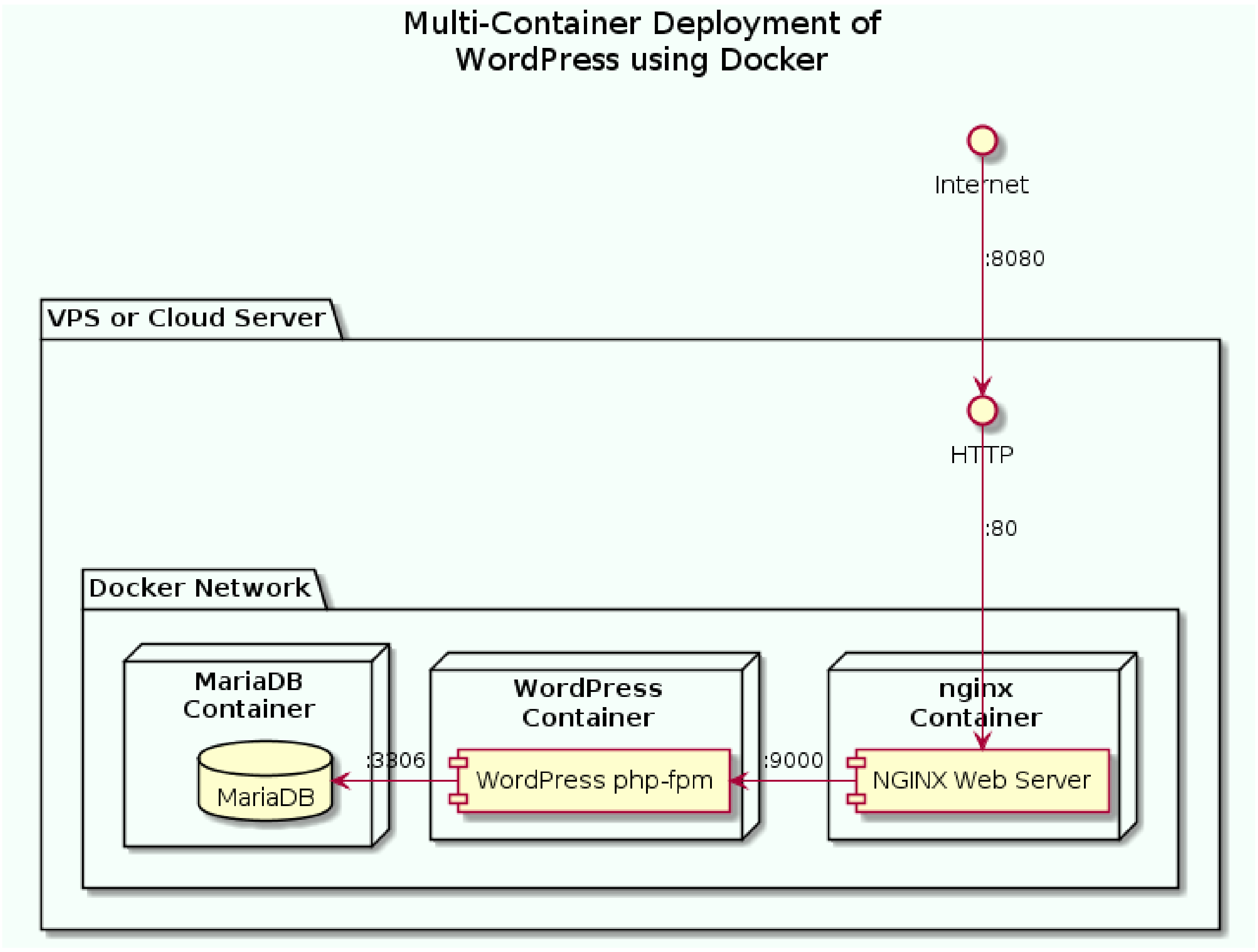
WordPress

Today, nearly half of the top 1 million sites on the internet use the WordPress CMS (Content Management System). WordPress has a plethora of features and is very easy to use even for non-technical users which makes it a go-to CMS solution for many. However, setting up a website with WordPress in a VPS server or a Cloud server can be a pretty daunting task for both non-technical as well as technical users. We'll learn how to do that in this article.

Docker

Docker is a software that can package your application, its dependencies, system tools, system libraries and settings in a single comprehensive virtual container. This combined package, which contains everything that's needed to run your application, is called a **Docker container image**. This image is a standalone, executable software package which runs reliably on any server — Linux, Windows, macOS, public cloud or private cloud. Also, these images can be used to run the application as an isolated process in user space.

In this article, we'll learn how to deploy WordPress using NGINX, PHP-FPM and MariaDB using Docker Compose in an Ubuntu machine. The below image illustrates the multi-container deployment that we intend to create.



Multi-Container Deployment of WordPress using Docker

Overview of the Installation Process

The steps in the deployment are summarized below.

1. Install **Docker** in the machine.
2. Create a **docker-compose.yml** file that enables us to create a multi-container deployment.
3. Add **MariaDB** configuration to **docker-compose.yml**.
4. Configure the host machine to handle database files.
5. Add **WordPress** configuration to **docker-compose.yml.**
6. Add a configuration section to **nginx.conf** file.
7. Add **NGINX** configuration to **docker-compose.yml.**
8. Create the containers and run the services.

1. Prepare the host machine

Install Docker

Please click the below link and follow the Docker installation instructions from the Docker website.

Get Docker Engine — Community for Ubuntu

Estimated reading time: 12 minutes To get started with Docker Engine — Community on Ubuntu, make sure you meet the...

docs.docker.com

Install Docker Compose

Please click the below link and follow the Docker Compose installation instructions from the Docker website.

Install Docker Compose

Estimated reading time: 7 minutes You can run Compose on macOS, Windows, and 64-bit Linux. Docker Compose relies on…

docs.docker.com

2. Add MariaDB configuration

Create a **docker-compose.yml** file and add the below configuration to the file. The configuration for **MariaDB** database is listed under the service name **mysql**.

```
version: '3'

services:

  mysql:

    image: mariadb

    volumes:

      - /data/mysql:/var/lib/mysql

    environment:

      MYSQL_ROOT_PASSWORD: mysql_root_pass

      MYSQL_DATABASE: db_name

      MYSQL_USER: user_name

      MYSQL_PASSWORD: user_pass

    restart: always
```

The below section mounts the host machine path `/data/mysql` in the container path `/var/lib/mysql`.

```
      volumes:
        - /data/mysql:/var/lib/mysql
```

This enables all our DB files to be stored in the host machine instead of the container. We need to prepare this directory for usage by our container.

In the **MariaDB** container, the directory `/var/lib/mysql` is owned by the user `mysql`. The uid of this user in the container is `999`. It is not the same in Ubuntu — the host machine. The below commands delete any existing user with the user name `mysql` and creates a new user with uid `999`.

```
$ sudo userdel mysql
$ sudo useradd -u 999 mysql
```

Now, set the owner of the directory `/data/mysql` in the host machine.

```
$ sudo mkdir -p /data/mysql
$ sudo chown -R mysql:mysql /data/mysql
```

3. Add WordPress Configuration

Add WordPress configuration to the **docker-compose.yml** file as a service named **wordpress**.

```
version: '3'

services:
```

```
mysql:

  image: mariadb

  volumes:

    - /data/mysql:/var/lib/mysql

  environment:

    MYSQL_ROOT_PASSWORD: mysql_root_pass

    MYSQL_DATABASE: db_name

    MYSQL_USER: user_name

    MYSQL_PASSWORD: user_pass

  restart: always

wordpress:

  image: wordpress:php7.3-fpm-alpine

  volumes:

    - /data/html:/var/www/html

  depends_on:

    - mysql

  environment:

    WORDPRESS_DB_HOST: mysql

    MYSQL_ROOT_PASSWORD: mysql_root_pass

    WORDPRESS_DB_NAME: db_name

    WORDPRESS_DB_USER: user_name

    WORDPRESS_DB_PASSWORD: user_pass

    WORDPRESS_TABLE_PREFIX: wp_

  links:
    - mysql

  restart: always
```

Make sure the **WordPress** environment variables **MYSQL_ROOT_PASSWORD, WORDPRESS_DB_NAME, WORDPRESS_DB_USER**, and **WORDPRESS_DB_PASSWORD** match exactly with the **MariaDB** environment variables **MYSQL_ROOT_PASSWORD, MYSQL_DATABASE, MYSQL_USER**, and **MYSQL_PASSWORD** respectively. In case of any mismatch, the WordPress application would not be able to connect to the database.

The below section states that the **wordpress** service is dependent on the **mysql** service. This implies that the services are started and stopped in the dependency order.

```
    depends_on:
      - mysql
```

The below section enables the **wordpress** service to communicate with the **mysql** service.

```
    links:
      - mysql
```

The below section mounts the host machine path `/data/html` in the container path `/var/www/html`.

```
    volumes:
      - /data/html:/var/www/html
```

This enables all our WordPress application files to be stored in the host machine instead of the container. We need to prepare this directory for usage by our container.

In the **WordPress** container, the directory `/var/www/html` is owned by the user `www-data`. The uid of this user in the container is `82`. It is not the same in Ubuntu — the host machine. The below commands delete any existing user with the user name `www-data` and creates a new user with uid `82`.

```
$ sudo userdel www-data
$ sudo useradd -u 82 www-data
```

Now, set the owner of the directory `/data/html` in the host machine.

```
$ sudo mkdir -p /data/html
$ sudo chown -R www-data:www-data /data/html
```

Add the below lines to the **server** section in **nginx.conf** to make sure all PHP requests are sent to the **PHP-FPM** service.

```
# pass the PHP scripts to FastCGI server listening on wordpress:9000

location ~ \.php$ {

  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 SCRIPT_NAME $fastcgi_script_name;
  }
```

4. Add NGINX Configuration

Add NGINX configuration to the **docker-composer.yml** file as a service named **nginx**.

```
version: '3'

services:

  mysql:

    image: mariadb

    volumes:

      - /data/mysql:/var/lib/mysql

    environment:

      MYSQL_ROOT_PASSWORD: mysql_root_pass

      MYSQL_DATABASE: db_name

      MYSQL_USER: user_name

      MYSQL_PASSWORD: user_pass

    restart: always

  wordpress:

    image: wordpress:php7.3-fpm-alpine

    volumes:

      - /data/html:/var/www/html

    depends_on:
      - mysql

    environment:

      WORDPRESS_DB_HOST: mysql

      MYSQL_ROOT_PASSWORD: mysql_root_pass

      WORDPRESS_DB_NAME: db_name

      WORDPRESS_DB_USER: user_name

      WORDPRESS_DB_PASSWORD: user_pass

      WORDPRESS_TABLE_PREFIX: wp_

    links:

      - mysql

    restart: always

  nginx:
    image: nginx:alpine

    volumes:

      - nginx:/etc/nginx/conf.d

      - /data/html:/var/www/html

    ports:

      - 8080:80

    links:

      - wordpress
```

Make sure that the host path mentioned in the **nginx** configuration — `/data/html` — matches with the host path mentioned in the **WordPress** configuration.

The below section enables the **wordpress** service to communicate with the **mysql** service.

```
links:
      - wordpress
```

The below section maps the host machine port **8080** to the container port **80**. Notice that no other container has any port exposed to the host machine. This enhances the security of our deployment.

```
ports:
      - 8080:80
```

We also need to provide a configuration file for the **Nginx** server. Create a directory named `nginx` and put the below contents in a file named `nginx.conf` in that directory.

```
server {

  listen 80;

  listen [::]:80;

  access_log off;

  root /var/www/html;

  index index.php;

  server_name example.com;

  server_tokens off;

  location / {

    # First attempt to serve request as file, then
```

```
      # as directory, then fall back to displaying a 404.

      try_files $uri $uri/ /index.php?$args;

    }

    # pass the PHP scripts to FastCGI server listening on
  wordpress:9000

    location ~ \.php$ {

      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 SCRIPT_NAME $fastcgi_script_name;

    }

  }
```

Get Set Go!!!

Now create the containers and run the services with the below command.

```
  $ sudo docker-compose up -d
```

Once the installation is over, open up the URL http://localhost:8080 in your browser. This should show the WordPress initial setup page.

Choice of the path in Volumes

Notice that the database files are stored in the directory `/data/mysql` in the host machine and the wordpress files are stored in the directory `/data/wordpress` in the host machine. In our installation, these are the only directories that contain user data. And both these directories are in the host machine. Hence, you can easily backup all user data by simply creating a tarball of the `/data` directory as given below:

```
$ sudo tar -jcvf data.tar.bz2 /data
```

How to Upgrade?

Another question is how to upgrade to the latest version of nginx or wordpress docker images in our deployment. Since all user data is outside the containers, we can easily delete the containers and re-create them to upgrade our installation.

```
$ sudo docker-compose down --rmi all
$ sudo docker-compose up -d
```

Docker WordPress Docker Compose Wordpress Installation Docker Container