TechDoc: Docker

Qore.Works - 2018

Created at:	**2018-10-12**
Updated at:	**2018-10-12**
Client:	**Dolphiq Internal**
Document version:	**v1.0**

Index

Purpose

This document is prepared for developers to setup docker in their computers to up and run a PHP project.

Why do we need docker?

In a PHP developer's perspective, when you need to have multiple versions of PHP,MySQL installed, you can run on to problems with WAMP/LAMP/MAMP/XAMPP. As a solution we can use VirtualBox but it costs more memory and disk space and unnecessary complications when transferring data between VirtualBox and actual OS.

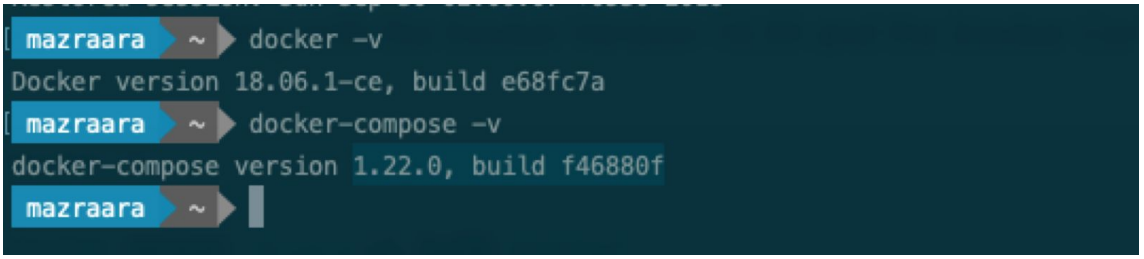
So we have docker, which is a container virtualization software. It isolates individual services in a container and runs them independently on the operating system. Virtualization with Docker is very lightweight and saves resources on the host system.

Installing Docker

You can download and install Docker Community Edition for Mac from:

- https://store.docker.com/search?type=edition&offering=community

After the successful installation and start, you should be able to check the docker version by typing “docker -v” on the terminal.



Setting Up Docker Environment

1. Create a new directory called “docker-project”

2. Create a new file called “docker-compose.yml” inside the directory and copy paste the below code block.

```
version: '2'
services:

  # The Application
  app:
        build:
        context: ./
        dockerfile: docker/app.docker
        working_dir: /var/www
        volumes:
        - ~/.composer-docker/cache:/root/.composer/cache:delegated
        - ./:/var/www

  # The Web Server
  web:
        build:
        context: ./
        dockerfile: docker/web.docker
        working_dir: /var/www
        volumes_from:
        - app
        links:
        - db
        ports:
        - 8080:80

  # The DB Server
  db:
        image: mysql:5.7
        volumes:
        - "./docker/data/db:/var/lib/mysql"
        restart: always
        ports:
        - "3306:3306"
        expose:
        - "3306"
        environment:
        MYSQL_ROOT_PASSWORD: root@123
```

```
      MYSQL_DATABASE: Homestead
      MYSQL_USER: db
      MYSQL_PASSWORD: db@123

# The phpMyAdmin
phpmyadmin:
      depends_on:
      - db
      image: phpmyadmin/phpmyadmin:latest
      ports:
      - "8088:80"
      links:
      - db
      environment:
      PMA_HOST: db
```

You can also download it from:
https://bitbucket.org/dq-private/dev-docker/src/development/docker-compose.yml

We are going to setup 4 containers respectively for the Application, Web Server, MySQL & phpMyAdmin. No need to go through each line and memorize this as we will only be working on these ones when a project is initiating.

Application and Web Server containers are referring to external dockerfile files because we are building a customized container.

Example:
dockerfile: docker/web.docker

MySQL container will create a readymade database with details provided under environment block.

Image:
You need to mention the name of the image and version here.
For example nginx:<version>.
You can browse all the available readymade images at https://hub.docker.com/explore/

Volumes:
This block is used to synch-up the host computer with the container. If this is not setup, every time we stop the containers or turn off docker we will lose all changes for example data stored in the database.

Ports:
This block is used to map the host computer port with the container port. For example "8080:80" means we map container port 80 with host computer port 8080

3. Create the dockerfiles for Application & Web Server container inside a new directory called docker.

Create a new file called “app.docker” and copy paste the below code block.

```
FROM php:7.1-fpm
RUN apt-get update && apt-get install -y libmcrypt-dev gnupg git zip unzip \
       mysql-client libmagickwand-dev --no-install-recommends \
       && pecl install imagick \
       && docker-php-ext-enable imagick \
       && docker-php-ext-install mcrypt pdo_mysql
RUN curl -sL https://deb.nodesource.com/setup_10.x | bash -
RUN apt-get install -y nodejs build-essential
RUN php -r "readfile('http://getcomposer.org/installer');" | php --
--install-dir=/usr/bin/ --filename=composer
 #Copy MYSQL configuration into the image
#COPY ./docker/conf/my.cnf /etc/my.cnf
 ARG WORKING_DIR
# Copy the working dir to the image's web root
COPY ./ /var/www
WORKDIR /var/www
```

Create a new file called “web.docker” and copy paste the below code block.

```
FROM nginx:1.10
ADD ./docker/conf/nginx.conf /etc/nginx/conf.d/default.conf
```

You can download these files from:

https://bitbucket.org/dq-private/dev-docker/src/development/docker/app.docker
https://bitbucket.org/dq-private/dev-docker/src/development/docker/web.docker

4. Create a directory named “data” inside docker directory to store database related files.

5. Create a directory called “conf” inside the docker directory to store configurations.

Create a file named nginx.conf and add the below code block.

```
server {
    listen 80;
    server_name app.laradock.com;
    index index.php index.html;
    root /var/www/app/public;
       access_log  /var/log/nginx/nginx.access.log;
       error_log   /var/log/nginx/nginx.error.log;
    location / {
        try_files $uri /index.php?$args;
    }
    location ~ \.php$ {
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        fastcgi_pass app:9000;
        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }
}
```

We are using this configuration in the web.docker file we created.
Example: ADD ./docker/conf/nginx.conf /etc/nginx/conf.d/default.conf

Create a file named my.cnf to override the default MySQL configuration.

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
general_log_file = /var/log/mariadb/query.log
general_log = 1
[mysqld_safe]
log-error=/var/log/mariadb/mariadb.log
pid-file=/var/run/mariadb/mariadb.pid
general_log_file = /var/log/mariadb/query.log
general_log = 1

# include all files from the config directory
#
!includedir /etc/my.cnf.d
```

You can see we have included this in app.docker file
Example: COPY ./docker/conf/my.cnf /etc/my.cnf

6. Create a directory called “app/public” inside “docker-project” to host the source code. Then create a dummy index.php and echo something.

7. Add the domain “app.laradock.com” to host file.

That’s all and we are ready to launch our containers.

We need to build it for the first time and then run it.

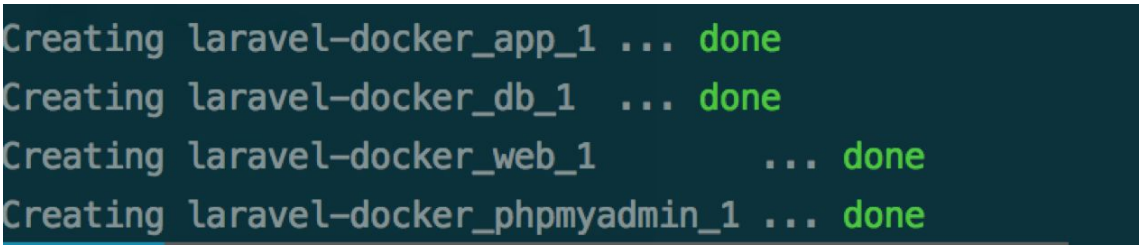
To build

```
docker-compose build
```

To run

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

Upon successful build and run, you should see something like below. First time building will take some time and then onwards It will start up in seconds.



Now you can access the phpMyAdmin from http://localhost:8088 and the Application from http://app.laradock.com or http://localhost:8080

Entire code block is available at: https://bitbucket.org/dq-private/dev-docker/src/development/

Basic docker commands to keep in mind

```
docker-compose up -d, docker-compose stop, docker-compose restart
docker exec -it ‘container-name’ bash
docker start ‘container-name’
docker stop ‘container-name’
docker rm ‘container-name’
```

Container to container connection

To connect to the database, use the link/alias you provided as a hostname. So, your php site/app can connect to MySQL using db as hostname, and port 3306. For Laravel usage, you can use the following parameters in your .env file.

```
DB_CONNECTION=mysql
DB_HOST=db
DB_PORT=3306
DB_DATABASE=Homestead
DB_USERNAME=db
DB_PASSWORD=db@123
```

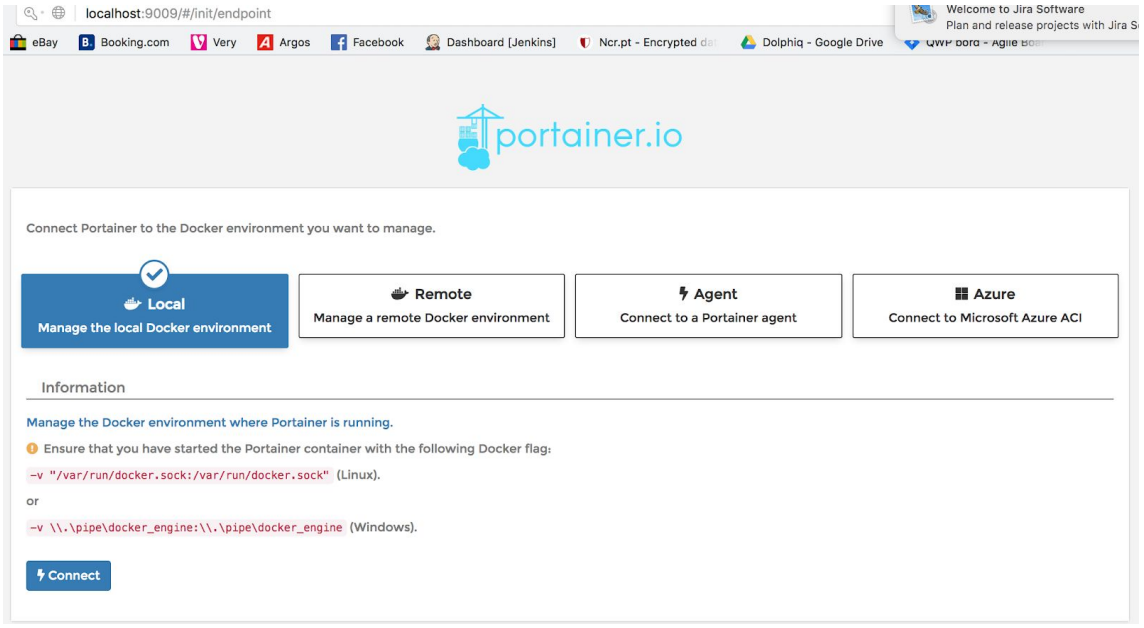
GUI tools

https://portainer.io
https://kitematic.com

How to install and use portainer

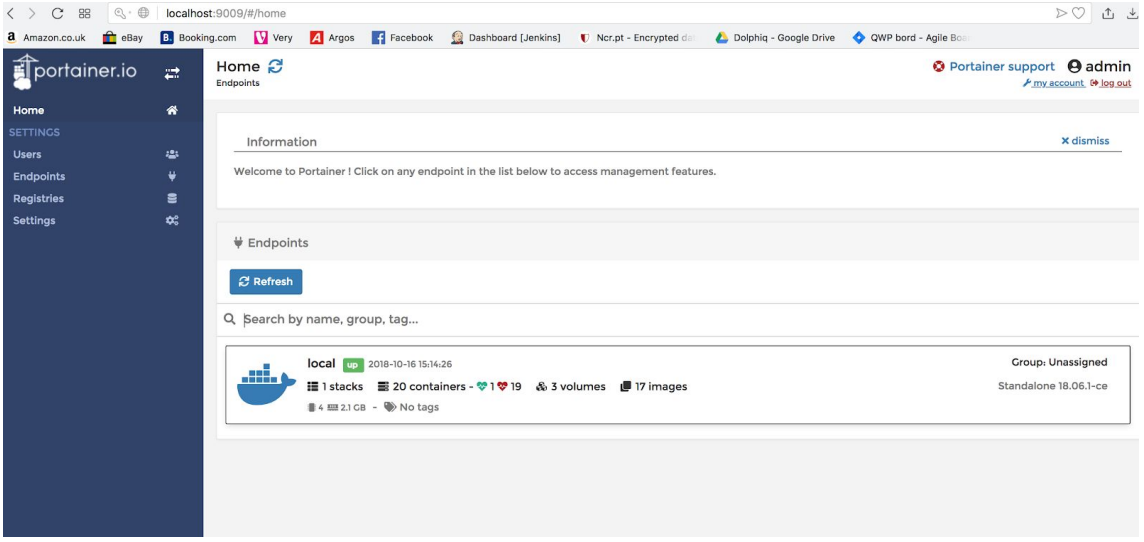
```
docker volume create portainer_data

docker run -d -p 9009:9000 -v /var/run/docker.sock:/var/run/docker.sock -v
portainer_data:/data portainer/portainer
```

After running the above commands, go to: http://localhost:9009/



Choose local and click connect.

Now you can see a cool Dashboard with all our containers and configurations listed with options to start and stop etc. you can stick to this GUI if you don't like to write commands in terminal.



Note: This document is prepared from a developer perspective. It's different when it comes for production setup with docker and it's a vast area handled by DevOps engineers.

If you have any questions on setting up, please contact me at azraar@dolphiq.nl