Laradock - Introduction

Laradock is a full PHP development environment for Docker.

It supports a variety of common services, all pre-configured to provide a ready PHP development environment.

Use Docker First - Learn About It Later!

Features

- Easy switch between PHP versions: 7.4, 7.3, 7.2, 7.1, 5.6…
- Choose your favorite database engine: MySQL, Postgres, MariaDB…
- Run your own stack: Memcached, HHVM, RabbitMQ…
- Each software runs on its own container: PHP-FPM, NGINX, PHP-CLI…
- Easy to customize any container, with simple edits to the `Dockerfile`.
- All Images extends from an official base Image. (Trusted base Images).
- Pre-configured NGINX to host any code at your root directory.
- Can use Laradock per project, or single Laradock for all projects.
- Easy to install/remove software's in Containers using environment variables.
- Clean and well structured Dockerfiles (`Dockerfile`).
- Latest version of the Docker Compose file (`docker-compose`).
- Everything is visible and editable.
- Fast Images Builds.

Quick Overview

Let's see how easy it is to setup our demo stack `PHP`, `NGINX`, `MySQL`, `Redis` and `Composer`:

1 - Clone Laradock inside your PHP project:

```
git clone https://github.com/Laradock/laradock.git
```

2 - Enter the laradock folder and rename `env-example` to `.env`.

```
cp env-example .env
```

3 - Run your containers:

```
docker-compose up -d nginx mysql phpmyadmin redis workspace
```

4 - Open your project's `.env` file and set the following:

```
DB_HOST=mysql
REDIS_HOST=redis
QUEUE_HOST=beanstalkd
```

5 - Open your browser and visit localhost: `http://localhost`.

`That's it! enjoy :)`

Supported Services

Laradock, adheres to the ‘separation of concerns’ principle, thus it runs each software on its own Docker Container. You can turn On/Off as many instances as you want without worrying about the configurations.

To run a chosen container from the list below, run `docker-compose up -d {container-name}`. The container name `{container-name}` is the same as its folder name. Example to run the “PHP FPM” container, use the name “php-fpm”.

- **Web Servers:**
    - NGINX
    - Apache2
    - Caddy
- **Load Balancers:**
    - HAProxy
    - Traefik
- **PHP Compilers:**
    - PHP FPM
    - HHVM
- **Database Management Systems:**
    - MySQL
    - PostgreSQL
        - PostGIS
    - MariaDB
    - Percona
    - MSSQL
    - MongoDB
        - MongoDB Web UI
    - Neo4j
    - CouchDB
    - RethinkDB
    - Cassandra
- **Database Management Apps:**
    - PhpMyAdmin
    - Adminer
    - PgAdmin
- **Cache Engines:**
    - Redis
        - Redis Web UI

        - Redis Cluster
    - Memcached
    - Aerospike
    - Varnish
- **Message Brokers:**
    - RabbitMQ
        - RabbitMQ Admin Console
    - Beanstalkd
        - Beanstalkd Admin Console
    - Eclipse Mosquitto
    - PHP Worker
    - Laravel Horizon
    - Gearman
    - Amazon Simple Queue Service
- **Mail Servers:**
    - Mailu
    - MailCatcher
    - Mailhog
    - MailDev
- **Log Management:**
    - GrayLog
- **Testing:**
    - Selenium
- **Monitoring:**
    - Grafana
    - NetData
- **Search Engines:**
    - ElasticSearch
    - Apache Solr
    - Manticore Search
- **IDE’s**
    - ICE Coder
    - Theia
    - Web IDE
- **Miscellaneous:**
    - Workspace *(Laradock container that includes a rich set of pre-configured useful tools)*
        - `PHP CLI`
        - `Composer`
        - `Git`
        - `Vim`
        - `xDebug`

        - `Linuxbrew`
        - `Node`
        - `V8JS`
        - `Gulp`
        - `SQLite`
        - `Laravel Envoy`
        - `Deployer`
        - `Yarn`
        - `SOAP`
        - `Drush`
        - `Wordpress CLI`
    - Apache ZooKeeper *(Centralized service for distributed systems to a hierarchical key-value store)*
    - Kibana *(Visualize your Elasticsearch data and navigate the Elastic Stack)*
    - Dejavu *(Edit your Elasticsearch data)*
    - LogStash *(Server-side data processing pipeline that ingests data from a multitude of sources simultaneously)*
    - Jenkins *(automation server, that provides plugins to support building, deploying and automating any project)*
    - Certbot *(Automatically enable HTTPS on your website)*
    - Swoole *(Production-Grade Async programming Framework for PHP)*
    - SonarQube *(continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs and more)*
    - Gitlab *(A single application for the entire software development lifecycle)*
    - PostGIS *(Database extender for PostgreSQL. It adds support for geographic objects allowing location queries to be run in SQL)*
    - Blackfire *(Empowers all PHP developers and IT/Ops to continuously verify and improve their app's performance)*
    - Laravel Echo *(Bring the power of WebSockets to your Laravel applications)*
    - Phalcon *(A PHP web framework based on the model–view–controller pattern)*
    - Minio *(Cloud storage server released under Apache License v2, compatible with Amazon S3)*
    - AWS EB CLI *(CLI that helps you deploy and manage your AWS Elastic Beanstalk applications and environments)*
    - Thumbor *(Photo thumbnail service)*
    - IPython *(Provides a rich architecture for interactive computing)*
    - Jupyter Hub *(Jupyter notebook for multiple users)*
    - Portainer *(Build and manage your Docker environments with ease)*
    - Docker Registry *(The Docker Registry implementation for storing and distributing Docker images)*
    - Docker Web UI *(A browser-based solution for browsing and modifying a private Docker registry)*

You can choose, which tools to install in your workspace container and other containers, from the `.env` file.

> If you modify `docker-compose.yml`, `.env` or any `dockerfile` file, you must re-build your containers, to see those effects in the running instance.

If you can't find your Software in the list, build it yourself and submit it. Contributions are welcomed :)

Join Us

[chat on gitter]

[Gitpod ready-to-code]

Awesome People

Laradock is an MIT-licensed open source project with its ongoing development made possible entirely by the support of all these smart and generous people, from code contributors to financial contributors.

Project Maintainers

Mahmoud Zalt
@mahmoudz

Bo-Yi Wu
@appleboy

Philippe Trépanier
@philtrep

Mike Erickson
@mikeerickson

Dwi Fahni Denni
@zeroc0d3

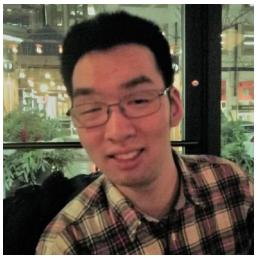
Thor Erik
@thorerik

Winfried van Loon
@winfried-van-loon

TJ Miller
@sixlive

Yu-Lung Shao (Allen)
@bestlong

Milan Urukalo
@urukalo



Vince Chu
@vwchu

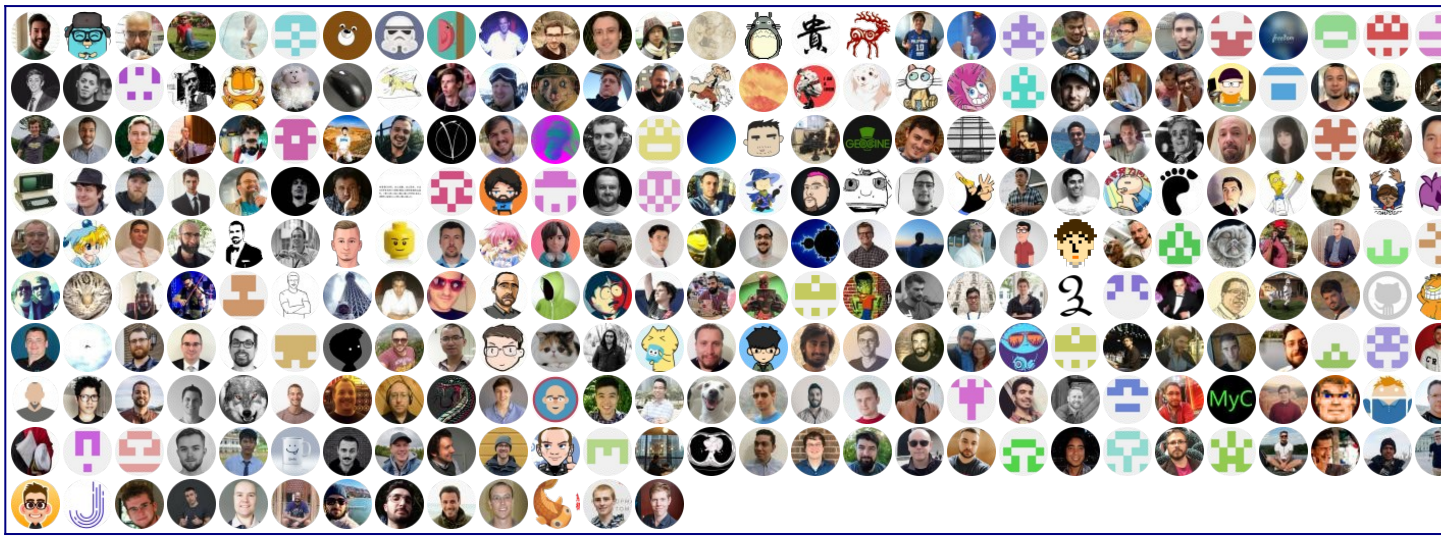
Huadong Zuo
@zuohuadong

Lan Phan
@lanphan

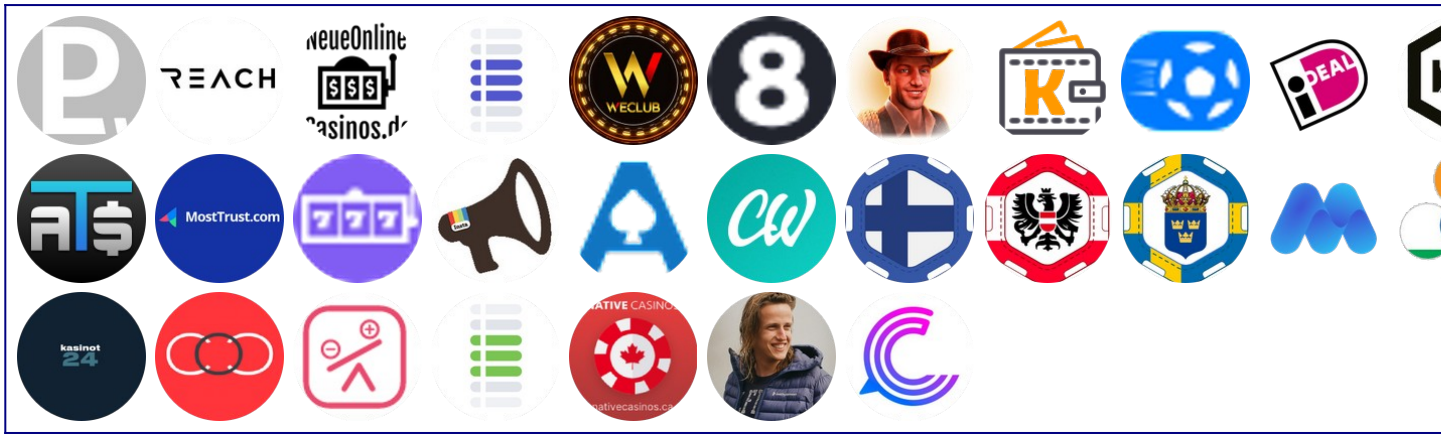
Ahkui
@ahkui

< Join Us >
@laradock

Code Contributors



Financial Contributors



You can support us using any of the methods below:

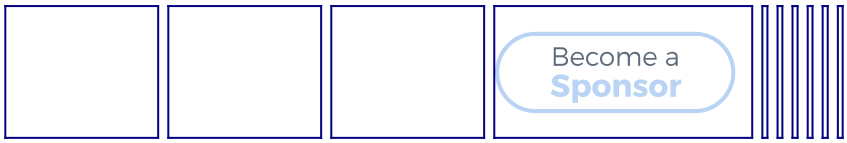
1: Open Collective
2: Paypal
3: Github Sponsors
4: Patreon

Sponsors

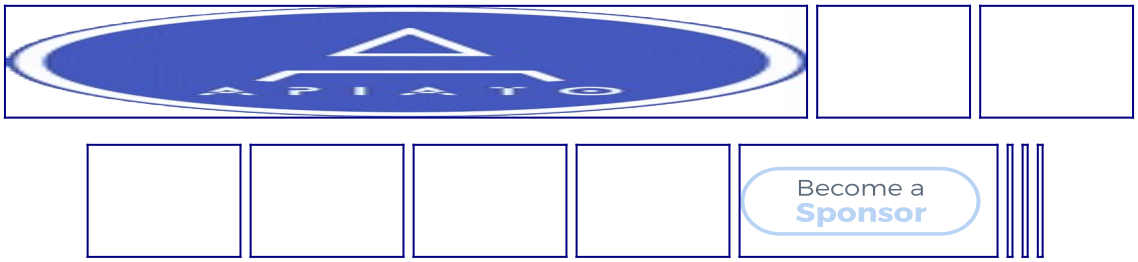
Sponsoring is an act of giving in a different fashion.

Gold Sponsors

Silver Sponsors



Bronze Sponsors



You can sponsor us using any of the methods below:

1: Sponsor via Open Collective.
2: Email us at support@laradock.io.

Sponsors logos are displayed on the github repository page and the documentation website home page.

License

MIT © Mahmoud Zalt

Getting Started

Requirements

- Git
- Docker [>= 17.12]

Installation

Choose the setup the best suits your needs.

A) Setup for Single Project

(Follow these steps if you want a separate Docker environment for each project)

A.1) Already have a PHP project:

1 - Clone laradock on your project root directory:

```
git submodule add https://github.com/Laradock/laradock.git
```

Note: If you are not using Git yet for your project, you can use `git clone` instead of `git submodule`.

To keep track of your Laradock changes, between your projects and also keep Laradock updated check these docs

2 - Make sure your folder structure should look like this:

```
* project-a
*   laradock-a
* project-b
*   laradock-b
```

(It’s important to rename the laradock folders to unique name in each project, if you want to run laradock per project).

3 - Go to the Usage section.

A.2) Don’t have a PHP project yet:

1 - Clone this repository anywhere on your machine:

```
git clone https://github.com/laradock/laradock.git
```

Your folder structure should look like this:

```
* laradock
* project-z
```

2 - Edit your web server sites configuration.

We'll need to do step 1 of the Usage section now to make this happen.

```
cp env-example .env
```

At the top, change the `APP_CODE_PATH_HOST` variable to your project path.

```
APP_CODE_PATH_HOST=../project-z/
```

Make sure to replace `project-z` with your project folder name.

3 - Go to the Usage section.

B) Setup for Multiple Projects:

(Follow these steps if you want a single Docker environment for all your projects)

1 - Clone this repository anywhere on your machine (similar to Steps A.2. from above):

```
git clone https://github.com/laradock/laradock.git
```

Your folder structure should look like this:

```
* laradock
* project-1
* project-2
```

2 - Go to your web server and create config files to point to different project directory when visiting different domains:

For **Nginx** go to `nginx/sites`, for **Apache2** `apache2/sites`.

Laradock by default includes some sample files for you to copy `app.conf.example`, `laravel.conf.example` and `symfony.conf.example`.

3 - change the default names `*.conf`:

You can rename the config files, project folders and domains as you like, just make sure the `root` in the config files, is pointing to the correct project folder name.

4 - Add the domains to the **hosts** files.

```
127.0.0.1  project-1.test
127.0.0.1  project-2.test
...
```

If you use Chrome 63 or above for development, don't use `.dev`. Why?. Instead use `.localhost`, `.invalid`, `.test`, or `.example`.

4 - Go to the Usage section.

Usage

Read Before starting:

If you are using **Docker Toolbox** (VM), do one of the following:

- Upgrade to Docker Native for Mac/Windows (Recommended). Check out Upgrading Laradock
- Use Laradock v3.*. Visit the Laradock-ToolBox branch. *(outdated)*

We recommend using a Docker version which is newer than 1.13.

> **Warning:** If you used an older version of Laradock it's highly recommended to rebuild the containers you need to use see how you rebuild a container in order to prevent as much errors as possible.

1 - Enter the laradock folder and copy `env-example` to `.env`

```
cp env-example .env
```

You can edit the `.env` file to choose which software's you want to be installed in your environment. You can always refer to the `docker-compose.yml` file to see how those variables are being used.

Depending on the host's operating system you may need to change the value given to `COMPOSE_FILE`. When you are running Laradock on Mac OS the correct file separator to use is `:`. When running Laradock from a Windows environment multiple files must be separated with `;`.

By default the containers that will be created have the current directory name as suffix (e.g. `laradock_workspace_1`). This can cause mixture of data inside the container volumes if you use laradock in multiple projects. In this case, either read the guide for multiple projects or change the variable `COMPOSE_PROJECT_NAME` to something unique like your project name.

2 - Build the environment and run it using `docker-compose`

In this example we'll see how to run NGINX (web server) and MySQL (database engine) to host a PHP Web Scripts:

```
docker-compose up -d nginx mysql
```

Note: All the web server containers `nginx`, `apache` ..etc depends on `php-fpm`, which means if you run any of them, they will automatically launch the `php-fpm` container for you, so no need to explicitly specify it in the `up` command. If you have to do so, you may need to run them as follows: `docker-compose up -d nginx php-fpm mysql`.

You can select your own combination of containers from this list.

(Please note that sometimes we forget to update the docs, so check the `docker-compose.yml` file to see an updated list of all available containers).

3 - Enter the Workspace container, to execute commands like (Artisan, Composer, PHPUnit, Gulp, …)

```
docker-compose exec workspace bash
```

Alternatively, for Windows PowerShell users: execute the following command to enter any running container:

```
docker exec -it {workspace-container-id} bash
```

Note: You can add `--user=laradock` to have files created as your host's user. Example:

```
docker-compose exec --user=laradock workspace bash
```

You can change the PUID (User id) and PGID (group id) variables from the `.env` file)

4 - Update your project configuration to use the database host

Open your PHP project's `.env` file or whichever configuration file you are reading from, and set the database host `DB_HOST` to `mysql`:

```
DB_HOST=mysql
```

You need to use the Laradock's default DB credentials which can be found in the `.env` file (ex: `MYSQL_USER=`). Or you can change them and rebuild the container.

If you want to install Laravel as PHP project, see How to Install Laravel in a Docker Container.

5 - Open your browser and visit your localhost address.

Make sure you add use the right port number as provided by your running server.

http://localhost

If you followed the multiple projects setup, you can visit `http://project-1.test/` and `http://project-2.test/`.

Documentation

List current running Containers

```
docker ps
```

You can also use the following command if you want to see only this project containers:

```
docker-compose ps
```

Close all running Containers

```
docker-compose stop
```

To stop single container do:

```
docker-compose stop {container-name}
```

Delete all existing Containers

```
docker-compose down
```

Enter a Container

> Run commands in a running Container.

1 - First list the currently running containers with `docker ps`

2 - Enter any container using:

```
docker-compose exec {container-name} bash
```

Example: enter MySQL container

```
docker-compose exec mysql bash
```

Example: enter to MySQL prompt within MySQL container

```
docker-compose exec mysql mysql -udefault -psecret
```

3 - To exit a container, type `exit`.

Edit default Container config

Open the `docker-compose.yml` and change anything you want.

Examples:

Change MySQL Database Name:

```
    environment:
        MYSQL_DATABASE: laradock
    ...
```

Change Redis default port to 1111:

```
    ports:
        - "1111:6379"
    ...
```

Edit a Docker Image

1 - Find the `Dockerfile` of the image you want to edit,
example for `mysql` it will be `mysql/Dockerfile`.

2 - Edit the file the way you want.

3 - Re-build the container:

```
docker-compose build mysql
```

More info on Containers rebuilding here.

Build/Re-build Containers

If you do any change to any `Dockerfile` make sure you run this command, for the changes to take effect:

```
docker-compose build
```

Optionally you can specify which container to rebuild (instead of rebuilding all the containers):

```
docker-compose build {container-name}
```

You might use the `--no-cache` option if you want full rebuilding (`docker-compose build --no-cache {container-name}`).

Add more Docker Images

To add an image (software), just edit the `docker-compose.yml` and add your container details, to do so you need to be familiar with the docker compose file syntax.

View the Log files

The NGINX Log file is stored in the `logs/nginx` directory.

However to view the logs of all the other containers (MySQL, PHP-FPM,…) you can run this:

```
docker-compose logs {container-name}
```

```
docker-compose logs -f {container-name}
```

More options

Install PHP Extensions

You can set extensions to install in the .env file's corresponding section (`PHP_FPM`, `WORKSPACE`, `PHP_WORKER`), just change the `false` to `true` at the desired extension's line. After this you have to rebuild the container with the `--no-cache` option.

```
docker build --no-cache {container-name}
```

Change the (PHP-FPM) Version

By default the latest stable PHP version is configured to run.

> The PHP-FPM is responsible for serving your application code, you don't have to change the PHP-CLI version if you are planning to run your application on different PHP-FPM version.

A) Switch from PHP `7.2` to PHP `5.6`

1 - Open the `.env`.

2 - Search for `PHP_VERSION`.

3 - Set the desired version number:

```
PHP_VERSION=5.6
```

4 - Finally rebuild the image

```
docker-compose build php-fpm
```

> For more details about the PHP base image, visit the official PHP docker images.

Change the PHP-CLI Version

> Note: it's not very essential to edit the PHP-CLI version. The PHP-CLI is only used for the Artisan Commands & Composer. It doesn't serve your Application code, this is the PHP-FPM job.

The PHP-CLI is installed in the Workspace container. To change the PHP-CLI version you need to simply change the `PHP_VERSION` in the .env file as follow:

1 - Open the `.env`.

2 - Search for `PHP_VERSION`.

3 - Set the desired version number:

```
PHP_VERSION=7.2
```

4 - Finally rebuild the image

```
docker-compose build workspace
```

Install xDebug

1 - First install `xDebug` in the Workspace and the PHP-FPM Containers:
a) open the `.env` file
b) search for the `WORKSPACE_INSTALL_XDEBUG` argument under the Workspace Container
c) set it to `true`
d) search for the `PHP_FPM_INSTALL_XDEBUG` argument under the PHP-FPM Container
e) set it to `true`

2 - Re-build the containers `docker-compose build workspace php-fpm`

For information on how to configure xDebug with your IDE and work it out, check this Repository or follow up on the next section if you use linux and PhpStorm.

Start/Stop xDebug:

By installing xDebug, you are enabling it to run on startup by default.

To control the behavior of xDebug (in the `php-fpm` Container), you can run the following commands from the Laradock root folder, (at the same prompt where you run docker-compose):

- Stop xDebug from running by default: `.php-fpm/xdebug stop`.
- Start xDebug by default: `.php-fpm/xdebug start`.
- See the status: `.php-fpm/xdebug status`.

Note: If `.php-fpm/xdebug` doesn't execute and gives `Permission Denied` error the problem can be that file `xdebug` doesn't have execution access. This can be fixed by running `chmod` command with desired access permissions.

Install pcov

1 - First install `pcov` in the Workspace and the PHP-FPM Containers:
a) open the `.env` file
b) search for the `WORKSPACE_INSTALL_PCOV` argument under the Workspace Container
c) set it to `true`
d) search for the `PHP_FPM_INSTALL_PCOV` argument under the PHP-FPM Container
e) set it to `true`

2 - Re-build the containers `docker-compose build workspace php-fpm`

Note that pcov is only supported on PHP 7.1 or newer. For more information on setting up pcov optimally, check the recommended section of the README

Install phpdbg

Install `phpdbg` in the Workspace and the PHP-FPM Containers:

1 - Open the `.env`.

2 - Search for `WORKSPACE_INSTALL_PHPDBG`.

3 - Set value to `true`

4 - Do the same for `PHP_FPM_INSTALL_PHPDBG`

```
WORKSPACE_INSTALL_PHPDBG=true

PHP_FPM_INSTALL_PHPDBG=true
```

Install ionCube Loader

1 - First install `ionCube Loader` in the Workspace and the PHP-FPM Containers:
a) open the `.env` file
b) search for the `WORKSPACE_INSTALL_IONCUBE` argument under the Workspace Container
c) set it to `true`
d) search for the `PHP_FPM_INSTALL_IONCUBE` argument under the PHP-FPM Container
e) set it to `true`

2 - Re-build the containers `docker-compose build workspace php-fpm`

Always download the latest version of Loaders for ionCube .

Install Deployer

> A deployment tool for PHP.

1 - Open the `.env` file
2 - Search for the `WORKSPACE_INSTALL_DEPLOYER` argument under the Workspace Container
3 - Set it to `true`

4 - Re-build the containers `docker-compose build workspace`

Deployer Documentation Here

Install SonarQube

> An automatic code review tool.

SonarQube® is an automatic code review tool to detect bugs, vulnerabilities and code smells in your code. It can integrate with your existing workflow to enable continuous code inspection across your project branches and pull requests.
1 - Open the `.env` file
2 - Search for the `SONARQUBE_HOSTNAME=sonar.example.com` argument
3 - Set it to your-domain `sonar.example.com`
4 - `docker-compose up -d sonarqube`
5 - Open your browser: http://localhost:9000/

Troubleshooting:

if you encounter a database error:

```
docker-compose exec --user=root postgres
source docker-entrypoint-initdb.d/init_sonarqube_db.sh
```

If you encounter logs error:

```
docker-compose run --user=root --rm sonarqube chown sonarqube:sonarqube
/opt/sonarqube/logs
```

SonarQube Documentation Here

Prepare Laradock for Production

It's recommended for production to create a custom `docker-compose.yml` file, for example `production-docker-compose.yml`

In your new production `docker-compose.yml` file you should contain only the containers you are planning to run in production (usage example: `docker-compose -f production-docker-compose.yml up -d nginx mysql redis ...`).

Note: The Database (MySQL/MariaDB/…) ports should not be forwarded on production, because Docker will automatically publish the port on the host, which is quite insecure, unless specifically told not to. So make sure to remove these lines:

```
ports:
    - "3306:3306"
```

To learn more about how Docker publishes ports, please read this excellent post on the subject.

Install Laravel from Container

1 - First you need to enter the Workspace Container.

2 - Install Laravel.

Example using Composer

```
composer create-project laravel/laravel my-cool-app "5.2.*"
```

> We recommend using `composer create-project` instead of the Laravel installer, to install Laravel.

For more about the Laravel installation click here.

3 - Edit `.env` to Map the new application path:

By default, Laradock assumes the Laravel application is living in the parent directory of the laradock folder.

Since the new Laravel application is in the `my-cool-app` folder, we need to replace `../:/var/www` with `../my-cool-app/:/var/www`, as follow:

```
APP_CODE_PATH_HOST=../my-cool-app/
```

4 - Go to that folder and start working.

```
cd my-cool-app
```

5 - Go back to the Laradock installation steps to see how to edit the `.env` file.

Run Artisan Commands

You can run artisan commands and many other Terminal commands from the Workspace container.

1 - Make sure you have the workspace container running.

```
docker-compose up -d workspace // ..and all your other containers
```

2 - Find the Workspace container name:

```
docker-compose ps
```

3 - Enter the Workspace container:

```
docker-compose exec workspace bash
```

Note: Should add `--user=laradock` (example `docker-compose exec --user=laradock workspace bash`) to have files created as your host's user to prevent issue owner of log file will be changed to root then laravel website cannot write on log file if using rotated log and new log file not existed

4 - Run anything you want :)

```
php artisan
```

```
composer update
```

```
phpunit
```

```
vue serve
```

(browse the results at `http://localhost:[WORKSPACE_VUE_CLI_SERVE_HOST_PORT]`)

```
vue ui
```

(browse the results at `http://localhost:[WORKSPACE_VUE_CLI_UI_HOST_PORT]`)

Run Laravel Queue Worker

1 - Create supervisor configuration file (for ex., named `laravel-worker.conf`) for Laravel Queue Worker in `php-worker/supervisord.d/` by simply copy from `laravel-worker.conf.example`

2 - Start everything up

```
docker-compose up -d php-worker
```

Run Laravel Scheduler

Laradock provides 2 ways to run Laravel Scheduler 1 - Using cron in workspace container. Most of the time, when you start Laradock, it'll automatically start workspace container with cron inside, along with setting to run `schedule:run` command every minute.

2 - Using Supervisord in php-worker to run `schedule:run`. This way is suggested when you don't want to start workspace in production environment.
a) Comment out cron setting in workspace container, file `workspace/crontab/laradock`

```
# * * * * * laradock /usr/bin/php /var/www/artisan schedule:run >> /dev/null
2>&1
```

b) Create supervisor configuration file (for ex., named `laravel-scheduler.conf`) for Laravel Scheduler in `php-worker/supervisord.d/` by simply copy from `laravel-scheduler.conf.example`
c) Start php-worker container

```
docker-compose up -d php-worker
```

Use Browsersync

Using Use Browsersync with Laravel Mix.

1. Add the following settings to your `webpack.mix.js` file. Please refer to Browsersync Options page for more options.

```
const mix = require('laravel-mix')

(...)

mix.browserSync({
  open: false,
  proxy: 'nginx' // replace with your web server container
})
```

1. Run `npm run watch` within your `workspace` container.
2. Open your browser and visit address `http://localhost:[WORKSPACE_BROWSERSYNC_HOST_PORT]`. It will refresh the page automatically whenever you edit any source file in your project.
3. If you wish to access Browsersync UI for your project, visit address `http://localhost:[WORKSPACE_BROWSERSYNC_UI_HOST_PORT]`.

Use Mailu

1 - You need register a domain.

2 - Required RECAPTCHA for signup email HERE

2 - modify following environment variable in `.env` file

```
MAILU_RECAPTCHA_PUBLIC_KEY=<YOUR_RECAPTCHA_PUBLIC_KEY>
MAILU_RECAPTCHA_PRIVATE_KEY=<YOUR_RECAPTCHA_PRIVATE_KEY>
MAILU_DOMAIN=laradock.io
MAILU_HOSTNAMES=mail.laradock.io
```

2 - Open your browser and visit `http://YOUR_DOMAIN`.

Use NetData

1 - Run the NetData Container (`netdata`) with the `docker-compose up` command. Example:

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

2 - Open your browser and visit the localhost on port **19999**: `http://localhost:19999`

Use Metabase

1 - Run the Metabase Container (`metbase`) with the `docker-compose up` command. Example:

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

2 - Open your browser and visit the localhost on port **3030**: `http://localhost:3030`

3 - You can use environment to configure Metabase container. See docs in: Running Metabase on Docker

Use Jenkins

1) Boot the container `docker-compose up -d jenkins`. To enter the container type `docker-compose exec jenkins bash`.

2) Go to `http://localhost:8090/` (if you didn't change your default port mapping)

3) Authenticate from the web app.

- Default username is `admin`.
- Default password is `docker-compose exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword`.

(To enter container as root type `docker-compose exec --user root jenkins bash`).

4) Install some plugins.

5) Create your first Admin user, or continue as Admin.

Note: to add user go to `http://localhost:8090/securityRealm/addUser` and to restart it from the web app visit `http://localhost:8090/restart`.

You may wanna change the default security configuration, so go to `http://localhost:8090/configureSecurity/` under Authorization and choosing "Anyone can do anything" or "Project-based Matrix Authorization Strategy" or anything else.

Use Redis

1 - First make sure you run the Redis Container (`redis`) with the `docker-compose up` command.

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

> To execute redis commands, enter the redis container first `docker-compose exec redis bash` then enter the `redis-cli`.

2 - Open your Laravel's `.env` file and set the `REDIS_HOST` to `redis`

```
REDIS_HOST=redis
```

If you're using Laravel, and you don't find the `REDIS_HOST` variable in your `.env` file. Go to the database configuration file `config/database.php` and replace the default `127.0.0.1` IP with `redis` for Redis like this:

```
'redis' => [
    'cluster' => false,
    'default' => [
```

```
        'host'     => 'redis',
        'port'     => 6379,
        'database' => 0,
    ],
],
```

3 - To enable Redis Caching and/or for Sessions Management. Also from the `.env` file set `CACHE_DRIVER` and `SESSION_DRIVER` to `redis` instead of the default `file`.

```
CACHE_DRIVER=redis
SESSION_DRIVER=redis
```

4 - Finally make sure you have the `predis/predis` package `(~1.0)` installed via Composer:

```
composer require predis/predis:^1.0
```

5 - You can manually test it from Laravel with this code:

```
\Cache::store('redis')->put('Laradock', 'Awesome', 10);
```

Use Redis Cluster

1 - First make sure you run the Redis-Cluster Container (`redis-cluster`) with the `docker-compose up` command.

```
docker-compose up -d redis-cluster
```

2 - Open your Laravel's `config/database.php` and set the redis cluster configuration. Below is example configuration with phpredis.

Read the Laravel official documentation for more details.

```
'redis' => [
    'client' => 'phpredis',
    'options' => [
        'cluster' => 'redis',
    ],
    'clusters' => [
        'default' => [
            [
                'host' => 'redis-cluster',
                'password' => null,
                'port' => 7000,
                'database' => 0,
            ],
        ],
    ],
],
```

Use Varnish

The goal was to proxy the request to varnish server using nginx. So only nginx has been configured for Varnish proxy. Nginx is on port 80 or 443. Nginx sends request through varnish server and

varnish server sends request back to nginx on port 81 (external port is defined in `VARNISH_BACKEND_PORT`).

The idea was taken from this post

The Varnish configuration was developed and tested for Wordpress only. Probably it works with other systems.

Steps to configure varnish proxy server:

1. You have to set domain name for VARNISH_PROXY1_BACKEND_HOST variable.
2. If you want to use varnish for different domains, you have to add new configuration section in your env file. `VARNISH_PROXY1_CACHE_SIZE=128m VARNISH_PROXY1_BACKEND_HOST=replace_with_your_domain.name VARNISH_PROXY1_SERVER=SERVER1`
3. Then you have to add new config section into docker-compose.yml with related variables: `custom_proxy_name: container_name: custom_proxy_name build: ./varnish expose: - ${VARNISH_PORT} environment: - VARNISH_CONFIG=${VARNISH_CONFIG} - CACHE_SIZE=${VARNISH_PROXY2_CACHE_SIZE} - VARNISHD_PARAMS=${VARNISHD_PARAMS} - VARNISH_PORT=${VARNISH_PORT} - BACKEND_HOST=${VARNISH_PROXY2_BACKEND_HOST} - BACKEND_PORT=${VARNISH_BACKEND_PORT} - VARNISH_SERVER=${VARNISH_PROXY2_SERVER} ports: - "${VARNISH_PORT}:${VARNISH_PORT}" links: - workspace networks: - frontend`
4. change your varnish config and add nginx configuration. Example Nginx configuration is here: `nginx/sites/laravel_varnish.conf.example`.
5. `varnish/default.vcl` is old varnish configuration, which was used in the previous version. Use `default_wordpress.vcl` instead.

How to run:

1. Rename `default_wordpress.vcl` to `default.vcl`
2. `docker-compose up -d nginx`
3. `docker-compose up -d proxy`

Keep in mind that varnish server must be built after Nginx cause varnish checks domain affordability.

FAQ:

1. How to purge cache?
   run from any cli:
   `curl -X PURGE https://yourwebsite.com/.`
2. How to reload varnish?
   `docker container exec proxy varnishreload`
3. Which varnish commands are allowed?
   - varnishadm
   - varnishd

- varnishhist
- varnishlog
- varnishncsa
- varnishreload
- varnishstat
- varnishtest
- varnishtop

4. How to reload Nginx?

```
docker exec Nginx nginx -t
docker exec Nginx nginx -s reload
```

Use Mongo

1 - First install `mongo` in the Workspace and the PHP-FPM Containers:

a) open the `.env` file

b) search for the `WORKSPACE_INSTALL_MONGO` argument under the Workspace Container

c) set it to `true`

d) search for the `PHP_FPM_INSTALL_MONGO` argument under the PHP-FPM Container

e) set it to `true`

2 - Re-build the containers `docker-compose build workspace php-fpm`

3 - Run the MongoDB Container (`mongo`) with the `docker-compose up` command.

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

4 - Add the MongoDB configurations to the `config/database.php` configuration file:

```
'connections' => [

    'mongodb' => [
        'driver'   => 'mongodb',
        'host'     => env('DB_HOST', 'localhost'),
        'port'     => env('DB_PORT', 27017),
        'database' => env('DB_DATABASE', 'database'),
        'username' => '',
        'password' => '',
        'options'  => [
            'database' => '',
        ]
    ],

        // ...

],
```

5 - Open your Laravel's `.env` file and update the following variables:

- set the `DB_HOST` to your `mongo`.
- set the `DB_PORT` to `27017`.
- set the `DB_DATABASE` to `database`.

6 - Finally make sure you have the `jenssegers/mongodb` package installed via Composer and its Service Provider is added.

```
composer require jenssegers/mongodb
```

More details about this here.

7 - Test it:

- First, let your Models extend from the Mongo Eloquent Model. Check the documentation.
- Enter the Workspace Container.
- Migrate the Database `php artisan migrate`.

Use PhpMyAdmin

1 - Run the phpMyAdmin Container (`phpmyadmin`) with the `docker-compose up` command. Example:

```
# use with mysql
docker-compose up -d mysql phpmyadmin

# use with mariadb
docker-compose up -d mariadb phpmyadmin
```

Note: To use with MariaDB, open `.env` and set `PMA_DB_ENGINE=mysql` to `PMA_DB_ENGINE=mariadb`.

2 - Open your browser and visit the localhost on port **8081**: `http://localhost:8081`

Use Gitlab

1 - Run the Gitlab Container (`gitlab`) with the `docker-compose up` command. Example:

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

2 - Open your browser and visit the localhost on port **8989**: `http://localhost:8989`
Note: You may change GITLAB_DOMAIN_NAME to your own domain name like `http://gitlab.example.com` default is `http://localhost`

Use Gitlab Runner

1 - Retrieve the registration token in your gitlab project (Settings > CI / CD > Runners > Set up a specific Runner manually)

2 - Open the `.env` file and set the following changes:

```
# so that gitlab container will pass the correct domain to gitlab-runner
container
GITLAB_DOMAIN_NAME=http://gitlab
```

```
GITLAB_RUNNER_REGISTRATION_TOKEN=<value-in-step-1>

# so that gitlab-runner container will send POST request for registration to
correct domain
GITLAB_CI_SERVER_URL=http://gitlab
```

3 - Open the `docker-compose.yml` file and add the following changes:

```
    gitlab-runner:
      environment: # these values will be used during `gitlab-runner register`
        - RUNNER_EXECUTOR=docker # change from shell (default)
        - DOCKER_IMAGE=alpine
        - DOCKER_NETWORK_MODE=laradock_backend
      networks:
        - backend # connect to network where gitlab service is connected
```

4 - Run the Gitlab-Runner Container (`gitlab-runner`) with the `docker-compose up` command. Example:

```
docker-compose up -d gitlab-runner
```

5 - Register the gitlab-runner to the gitlab container

```
docker-compose exec gitlab-runner bash
gitlab-runner register
```

6 - Create a `.gitlab-ci.yml` file for your pipeline

```
before_script:
  - echo Hello!

job1:
  scripts:
    - echo job1
```

7 - Push changes to gitlab

8 - Verify that pipeline is running successfully

Use Adminer

1 - Run the Adminer Container (`adminer`) with the `docker-compose up` command. Example:

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

2 - Open your browser and visit the localhost on port **8081**: `http://localhost:8081`

Additional Notes

- You can load plugins in the `ADM_PLUGINS` variable in the `.env` file. If a plugin requires parameters to work correctly you will need to add a custom file to the container. Find more info in section ‘Loading plugins’.
- You can choose a design in the `ADM_DESIGN` variable in the `.env` file. Find more info in section ‘Choosing a design’.

- You can specify the default host with the `ADM_DEFAULT_SERVER` variable in the `.env` file. This is useful if you are connecting to an external server or a docker container named something other than the default `mysql`.

Use Portainer

1 - Run the Portainer Container (`portainer`) with the `docker-compose up` command. Example:

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

2 - Open your browser and visit the localhost on port **9010**: `http://localhost:9010`

Use PgAdmin

1 - Run the pgAdmin Container (`pgadmin`) with the `docker-compose up` command. Example:

```
docker-compose up -d postgres pgadmin
```

2 - Open your browser and visit the localhost on port **5050**: `http://localhost:5050`

3 - At login page use default credentials:

Username : pgadmin4@pgadmin.org

Password : admin

Use Beanstalkd

1 - Run the Beanstalkd Container:

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

2 - Configure Laravel to connect to that container by editing the `config/queue.php` config file.

a. first set `beanstalkd` as default queue driver b. set the queue host to beanstalkd : `QUEUE_HOST=beanstalkd`

beanstalkd is now available on default port `11300`.

3 - Require the dependency package pda/pheanstalk using composer.

Optionally you can use the Beanstalkd Console Container to manage your Queues from a web interface.

1 - Run the Beanstalkd Console Container:

```
docker-compose up -d beanstalkd-console
```

2 - Open your browser and visit `http://localhost:2080/`

Note: You can customize the port on which beanstalkd console is listening by changing `BEANSTALKD_CONSOLE_HOST_PORT` in `.env`. The default value is 2080.

3 - Add the server

- Host: beanstalkd
- Port: 11300

4 - Done.

Use Confluence

1 - Run the Confluence Container (`confluence`) with the `docker-compose up` command. Example:

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

2 - Open your browser and visit the localhost on port **8090**: `http://localhost:8090`

Note: Confluence is a licensed application - an evaluation licence can be obtained from Atlassian.

You can set custom confluence version in `CONFLUENCE_VERSION`. Find more info in section 'Versioning'

Confluence usage with Nginx and SSL.

1. Find an instance configuration file in `nginx/sites/confluence.conf.example` and replace sample domain with yours.
2. Configure ssl keys to your domain.

Keep in mind that Confluence is still accessible on 8090 anyway.

Use ElasticSearch

1 - Run the ElasticSearch Container (`elasticsearch`) with the `docker-compose up` command:

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

2 - Open your browser and visit the localhost on port **9200**: `http://localhost:9200`

> The default username is `elastic` and the default password is `changeme`.

Install ElasticSearch Plugin

1 - Install an ElasticSearch plugin.

```
docker-compose exec elasticsearch /usr/share/elasticsearch/bin/plugin install {plugin-name}
```

For ElasticSearch 5.0 and above, the previous “plugin” command has been renamed to “elasticsearch-plguin”. Use the following instead:

```
docker-compose exec elasticsearch /usr/share/elasticsearch/bin/elasticsearch-plugin install {plugin-name}
```

2 - Restart elasticsearch container

```
docker-compose restart elasticsearch
```

Use MeiliSearch

1 - Run the MeiliSearch Container (`meilisearch`) with the `docker-compose up` command. Example:

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

2 - Open your browser and visit the localhost on port **7700** at the following URL:
`http://localhost:7700`

The private API key is `masterkey`

Use Selenium

1 - Run the Selenium Container (`selenium`) with the `docker-compose up` command. Example:

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

2 - Open your browser and visit the localhost on port **4444** at the following URL:
`http://localhost:4444/wd/hub`

Use RethinkDB

The RethinkDB is an open-source Database for Real-time Web (RethinkDB). A package (Laravel RethinkDB) is being developed and was released a version for Laravel 5.2 (experimental).

1 - Run the RethinkDB Container (`rethinkdb`) with the `docker-compose up` command.

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

2 - Access the RethinkDB Administration Console http://localhost:8090/#tables for create a database called `database`.

3 - Add the RethinkDB configurations to the `config/database.php` configuration file:

```
'connections' => [

        'rethinkdb' => [
```

```
                'name'      => 'rethinkdb',
                'driver'    => 'rethinkdb',
                'host'      => env('DB_HOST', 'rethinkdb'),
                'port'      => env('DB_PORT', 28015),
                'database'  => env('DB_DATABASE', 'test'),
        ]

        // ...

],
```

4 - Open your Laravel's `.env` file and update the following variables:

- set the `DB_CONNECTION` to your `rethinkdb`.
- set the `DB_HOST` to `rethinkdb`.
- set the `DB_PORT` to `28015`.
- set the `DB_DATABASE` to `database`.

Additional Notes

- You may do backing up of your data using the next reference: backing up your data.

Use Minio

1 - Configure Minio: - On the workspace container, change `INSTALL_MC` to true to get the client - Set `MINIO_ACCESS_KEY` and `MINIO_ACCESS_SECRET` if you wish to set proper keys

2 - Run the Minio Container (`minio`) with the `docker-compose up` command. Example:

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

3 - Open your browser and visit the localhost on port **9000** at the following URL: `http://localhost:9000`

4 - Create a bucket either through the webui or using the mc client:

```
  mc mb minio/bucket
```

5 - When configuring your other clients use the following details:

```
  AWS_URL=http://minio:9000
  AWS_ACCESS_KEY_ID=access
  AWS_SECRET_ACCESS_KEY=secretkey
  AWS_DEFAULT_REGION=us-east-1
  AWS_BUCKET=test
  AWS_PATH_STYLE=true
```

6 - In `filesystems.php` you shoud use the following details (s3):

```
's3' => [
            'driver' => 's3',
            'key' => env('AWS_ACCESS_KEY_ID'),
            'secret' => env('AWS_SECRET_ACCESS_KEY'),
            'region' => env('AWS_DEFAULT_REGION'),
            'bucket' => env('AWS_BUCKET'),
            'endpoint' => env('AWS_URL'),
```

```
            'use_path_style_endpoint' => env('AWS_PATH_STYLE', false)
        ],
```

`'AWS_PATH_STYLE'` shout set to true only for local purpouse

Use Thumbor

Thumbor is a smart imaging service. It enables on-demand crop, resizing and flipping of images. (Thumbor)

1 - Configure Thumbor: - Checkout all the options under the thumbor settings

2 - Run the Thumbor Container (`minio`) with the `docker-compose up` command. Example:

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

3 - Navigate to an example image on
`http://localhost:8000/unsafe/300x300/i.imgur.com/bvjzPct.jpg`

For more documentation on Thumbor visit the Thumbor documenation page

Use AWS

1 - Configure AWS: - make sure to add your SSH keys in aws-eb-cli/ssh_keys folder

2 - Run the Aws Container (`aws`) with the `docker-compose up` command. Example:

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

3 - Access the aws container with `docker-compose exec aws bash`

4 - To start using eb cli inside the container, initialize your project first by doing ‘eb init’. Read the aws eb cli docs for more details.

Use Grafana

1 - Configure Grafana: Change Port using `GRAFANA_PORT` if you wish to. Default is port 3000.

2 - Run the Grafana Container (`grafana`) with the `docker-compose up`command:

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

3 - Open your browser and visit the localhost on port **3000** at the following URL:
`http://localhost:3000`

4 - Login using the credentials User = `admin`, Password = `admin`. Change the password in the web interface if you want to.

Use Graylog

1 - Boot the container `docker-compose up -d graylog`

2 - Open your Laravel's `.env` file and set the `GRAYLOG_PASSWORD` to some passsword, and `GRAYLOG_SHA256_PASSWORD` to the sha256 representation of your password (`GRAYLOG_SHA256_PASSWORD` is what matters, `GRAYLOG_PASSWORD` is just a reminder of your password).

> Your password must be at least 16 characters long You can generate sha256 of some password with the following command `echo -n somesupersecretpassword | sha256sum`

```
GRAYLOG_PASSWORD=somesupersecretpassword
GRAYLOG_SHA256_PASSWORD=b1cb6e31e172577918c9e7806c572b5ed8477d3f57aa737bee4b5b1d
b3696f09
```

3 - Go to `http://localhost:9000/` (if your port is not changed)

4 - Authenticate from the app.

> Username: admin Password: somesupersecretpassword (if you haven't changed the password)

5 - Go to the system->inputs and launch new input

Use Traefik

To use Traefik you need to do some changes in `.env` and `docker-compose.yml`.

1 - Open `.env` and change `ACME_DOMAIN` to your domain and `ACME_EMAIL` to your email.

2 - You need to change the `docker-compose.yml` file to match the Traefik needs. If you want to use Traefik, you must not expose the ports of each container to the internet, but specify some labels.

2.1 For example, let's try with NGINX. You must have:

```
nginx:
  build:
    context: ./nginx
    args:
      - PHP_UPSTREAM_CONTAINER=${NGINX_PHP_UPSTREAM_CONTAINER}
      - PHP_UPSTREAM_PORT=${NGINX_PHP_UPSTREAM_PORT}
      - CHANGE_SOURCE=${CHANGE_SOURCE}
  volumes:
    - ${APP_CODE_PATH_HOST}:${APP_CODE_PATH_CONTAINER}
    - ${NGINX_HOST_LOG_PATH}:/var/log/nginx
    - ${NGINX_SITES_PATH}:/etc/nginx/sites-available
  depends_on:
    - php-fpm
  networks:
    - frontend
    - backend
  labels:
```

```
    - "traefik.enable=true"
    - "traefik.http.services.nginx.loadbalancer.server.port=80"
    # https router
    - "traefik.http.routers.https.rule=Host(`${ACME_DOMAIN}`, `www.$
{ACME_DOMAIN}`)"
    - "traefik.http.routers.https.entrypoints=https"
    - "traefik.http.routers.https.middlewares=www-redirectregex"
    - "traefik.http.routers.https.service=nginx"
    - "traefik.http.routers.https.tls.certresolver=letsencrypt"
    # http router
    - "traefik.http.routers.http.rule=Host(`${ACME_DOMAIN}`, `www.$
{ACME_DOMAIN}`)"
    - "traefik.http.routers.http.entrypoints=http"
    - "traefik.http.routers.http.middlewares=http-redirectscheme"
    - "traefik.http.routers.http.service=nginx"
    # middlewares
    - "traefik.http.middlewares.www-redirectregex.redirectregex.permanent=true"
    - "traefik.http.middlewares.www-redirectregex.redirectregex.regex=^https://
www.(.*)"
    - "traefik.http.middlewares.www-
redirectregex.redirectregex.replacement=https://$$1"
    - "traefik.http.middlewares.http-
redirectscheme.redirectscheme.permanent=true"
    - "traefik.http.middlewares.http-redirectscheme.redirectscheme.scheme=https"
```

instead of

```
nginx:
  build:
    context: ./nginx
    args:
      - PHP_UPSTREAM_CONTAINER=${NGINX_PHP_UPSTREAM_CONTAINER}
      - PHP_UPSTREAM_PORT=${NGINX_PHP_UPSTREAM_PORT}
      - CHANGE_SOURCE=${CHANGE_SOURCE}
  volumes:
    - ${APP_CODE_PATH_HOST}:${APP_CODE_PATH_CONTAINER}
    - ${NGINX_HOST_LOG_PATH}:/var/log/nginx
    - ${NGINX_SITES_PATH}:/etc/nginx/sites-available
    - ${NGINX_SSL_PATH}:/etc/nginx/ssl
  ports:
    - "${NGINX_HOST_HTTP_PORT}:80"
    - "${NGINX_HOST_HTTPS_PORT}:443"
  depends_on:
    - php-fpm
  networks:
    - frontend
    - backend
```

Use Mosquitto (MQTT Broker)

1 - Configure Mosquitto: Change Port using `MOSQUITTO_PORT` if you wish to. Default is port 9001.

2 - Run the Mosquitto Container (`mosquitto`) with the `docker-compose up`command:

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

3 - Open your command line and use a MQTT Client (Eg. https://github.com/mqttjs/MQTT.js) to subscribe a topic and publish a message.

4 - Subscribe: `mqtt sub -t 'test' -h localhost -p 9001 -C 'ws' -v`

5 - Publish: `mqtt pub -t 'test' -h localhost -p 9001 -C 'ws' -m 'Hello!'`

Install CodeIgniter

To install CodeIgniter 3 on Laradock all you have to do is the following simple steps:

1 - Open the `docker-compose.yml` file.

2 - Change `CODEIGNITER=false` to `CODEIGNITER=true`.

3 - Re-build your PHP-FPM Container `docker-compose build php-fpm`.

Install Powerline

1 - Open the `.env` file and set `WORKSPACE_INSTALL_POWERLINE` and `WORKSPACE_INSTALL_PYTHON` to `true`.

2 - Run `docker-compose build workspace`, after the step above.

Powerline is required python

Install Symfony

1 - Open the `.env` file and set `WORKSPACE_INSTALL_SYMFONY` to `true`.

2 - Run `docker-compose build workspace`, after the step above.

3 - The NGINX sites include a default config file for your Symfony project `symfony.conf.example`, so edit it and make sure the `root` is pointing to your project `web` directory.

4 - Run `docker-compose restart` if the container was already running, before the step above.

5 - Visit `symfony.test`

Miscellaneous

Change the timezone

To change the timezone for the `workspace` container, modify the `TZ` build argument in the Docker Compose file to one in the TZ database.

For example, if I want the timezone to be `New York`:

```
    workspace:
        build:
            context: ./workspace
            args:
                - TZ=America/New_York
    ...
```

We also recommend setting the timezone in Laravel.

Add locales to PHP-FPM

To add locales to the container:

1 - Open the `.env` file and set `PHP_FPM_INSTALL_ADDITIONAL_LOCALES` to `true`.

2 - Add locale codes to `PHP_FPM_ADDITIONAL_LOCALES`.

3 - Re-build your PHP-FPM Container `docker-compose build php-fpm`.

4 - Check enabled locales with `docker-compose exec php-fpm locale -a`

Update the locale setting, default is `POSIX`

1 - Open the `.env` file and set `PHP_FPM_DEFAULT_LOCALE` to `en_US.UTF8` or other locale you want.

2 - Re-build your PHP-FPM Container `docker-compose build php-fpm`.

3 - Check the default locale with `docker-compose exec php-fpm locale`

Adding cron jobs

You can add your cron jobs to `workspace/crontab/root` after the `php artisan` line.

```
* * * * * laradock /usr/bin/php /var/www/artisan schedule:run >> /dev/null 2>&1

# Custom cron
* * * * * root echo "Every Minute" > /var/log/cron.log 2>&1
```

Make sure you change the timezone if you don’t want to use the default (UTC).

If you are on Windows, verify that the line endings for this file are LF only, otherwise the cron jobs will silently fail.

Access workspace via ssh

You can access the `workspace` container through `localhost:2222` by setting the `INSTALL_WORKSPACE_SSH` build argument to `true`.

To change the default forwarded port for ssh:

```
    workspace:
                ports:
                        - "2222:22" # Edit this line
    ...
```

Then login using:

```
ssh -o PasswordAuthentication=no    \
    -o StrictHostKeyChecking=no     \
    -o UserKnownHostsFile=/dev/null \
    -p 2222                         \
    -i workspace/insecure_id_rsa    \
    laradock@localhost
```

To login as root, replace laradock@localhost with root@localhost.

Change the (MySQL) Version

By default **MySQL 8.0** is running.

MySQL 8.0 is a development release. You may prefer to use the latest stable version, or an even older release. If you wish, you can change the MySQL image that is used.

Open up your .env file and set the `MYSQL_VERSION` variable to the version you would like to install.

```
MYSQL_VERSION=5.7
```

Available versions are: 5.5, 5.6, 5.7, 8.0, or latest. See https://store.docker.com/images/mysql for more information.

MySQL root access

The default username and password for the root MySQL user are `root` and `root`.

1 - Enter the MySQL container: `docker-compose exec mysql bash`.

2 - Enter mysql: `mysql -uroot -proot` for non root access use `mysql -udefault -psecret`.

3 - See all users: `SELECT User FROM mysql.user;`

4 - Run any commands `show databases`, `show tables`, `select * from.....`

Create Multiple Databases

With MySQL.

Create `createdb.sql` from `mysql/docker-entrypoint-initdb.d/createdb.sql.example` in `mysql/docker-entrypoint-initdb.d/*` and add your SQL syntax as follow:

```
CREATE DATABASE IF NOT EXISTS `your_db_1` COLLATE 'utf8_general_ci' ;
GRANT ALL ON `your_db_1`.* TO 'mysql_user'@'%' ;
```

Change MySQL port

Modify the `mysql/my.cnf` file to set your port number, `1234` is used as an example.

```
[mysqld]
port=1234
```

If you need MySQL access from your host, do not forget to change the internal port number (`"3306:3306"` -> `"3306:1234"`) in the docker-compose configuration file.

Use custom Domain

How to use a custom domain, instead of the Docker IP.

Assuming your custom domain is `laravel.test`

1 - Open your `/etc/hosts` file and map your localhost address `127.0.0.1` to the `laravel.test` domain, by adding the following:

```
127.0.0.1    laravel.test
```

2 - Open your browser and visit `{http://laravel.test}`

Optionally you can define the server name in the NGINX configuration file, like this:

```
server_name laravel.test;
```

Global Composer Build Install

Enabling Global Composer Install during the build for the container allows you to get your composer requirements installed and available in the container after the build is done.

1 - Open the `.env` file

2 - Search for the `WORKSPACE_COMPOSER_GLOBAL_INSTALL` argument under the Workspace Container and set it to `true`

3 - Now add your dependencies to `workspace/composer.json`

4 - Re-build the Workspace Container `docker-compose build workspace`

Add authentication for Magento

Adding authentication credentials for Magento 2.

1 - Open the `.env` file

2 - Search for the `WORKSPACE_COMPOSER_AUTH` argument under the Workspace Container and set it to `true`

3 - Now add your credentials to `workspace/auth.json`

4 - Re-build the Workspace Container `docker-compose build workspace`

Install Prestissimo

Prestissimo is a plugin for composer which enables parallel install functionality.

1 - Enable Running Global Composer Install during the Build:

Click on this Enable Global Composer Build Install and do steps 1 and 2 only then continue here.

2 - Add prestissimo as requirement in Composer:

a - Now open the `workspace/composer.json` file

b - Add `"hirak/prestissimo": "^0.3"` as requirement

c - Re-build the Workspace Container `docker-compose build workspace`

Install Node + NVM

To install NVM and NodeJS in the Workspace container

1 - Open the `.env` file

2 - Search for the `WORKSPACE_INSTALL_NODE` argument under the Workspace Container and set it to `true`

3 - Re-build the container `docker-compose build workspace`

Install PNPM

pnpm uses hard links and symlinks to save one version of a module only ever once on a disk. When using npm or Yarn for example, if you have 100 projects using the same version of lodash, you will have 100 copies of lodash on disk. With pnpm, lodash will be saved in a single place on the disk and a hard link will put it into the node_modules where it should be installed.

As a result, you save gigabytes of space on your disk and you have a lot faster installations! If you'd like more details about the unique node_modules structure that pnpm creates and why it works fine with the Node.js ecosystem. More info here: https://pnpm.js.org/en/motivation

1 - Open the `.env` file

2 - Search for the `WORKSPACE_INSTALL_NODE` and `WORKSPACE_INSTALL_PNPM` argument under the Workspace Container and set it to `true`

3 - Re-build the container `docker-compose build workspace`

Install Node + YARN

Yarn is a new package manager for JavaScript. It is so faster than npm, which you can find here.To install NodeJS and Yarn in the Workspace container:

1 - Open the `.env` file

2 - Search for the `WORKSPACE_INSTALL_NODE` and `WORKSPACE_INSTALL_YARN` argument under the Workspace Container and set it to `true`

3 - Re-build the container `docker-compose build workspace`

Install NPM GULP toolkit

To install NPM GULP toolkit in the Workspace container

1 - Open the `.env` file

2 - Search for the `WORKSPACE_INSTALL_NPM_GULP` argument under the Workspace Container and set it to `true`

3 - Re-build the container `docker-compose build workspace`

Install NPM BOWER

To install NPM BOWER package manager in the Workspace container

1 - Open the `.env` file

2 - Search for the `WORKSPACE_INSTALL_NPM_BOWER` argument under the Workspace Container and set it to `true`

3 - Re-build the container `docker-compose build workspace`

Install NPM VUE CLI

To install NPM VUE CLI in the Workspace container

1 - Open the `.env` file

2 - Search for the `WORKSPACE_INSTALL_NPM_VUE_CLI` argument under the Workspace Container and set it to `true`

3 - Change `vue serve` port using `WORKSPACE_VUE_CLI_SERVE_HOST_PORT` if you wish to (default value is 8080)

4 - Change `vue ui` port using `WORKSPACE_VUE_CLI_UI_HOST_PORT` if you wish to (default value is 8001)

5 - Re-build the container `docker-compose build workspace`

Install NPM ANGULAR CLI

To install NPM ANGULAR CLI in the Workspace container

1 - Open the `.env` file

2 - Search for the `WORKSPACE_INSTALL_NPM_ANGULAR_CLI` argument under the Workspace Container and set it to `true`

3 - Re-build the container `docker-compose build workspace`

Install Linuxbrew

Linuxbrew is a package manager for Linux. It is the Linux version of MacOS Homebrew and can be found here. To install Linuxbrew in the Workspace container:

1 - Open the `.env` file

2 - Search for the `WORKSPACE_INSTALL_LINUXBREW` argument under the Workspace Container and set it to `true`

3 - Re-build the container `docker-compose build workspace`

Install FFMPEG

To install FFMPEG in the Workspace container

1 - Open the `.env` file

2 - Search for the `WORKSPACE_INSTALL_FFMPEG` argument under the Workspace Container and set it to `true`

3 - Re-build the container `docker-compose build workspace`

4 - If you use the `php-worker` container too, please follow the same steps above especially if you have conversions that have been queued.

PS Don't forget to install the binary in the `php-fpm` container too by applying the same steps above to its container, otherwise you'll get an error when running the `php-ffmpeg` binary.

Install BBC Audio Waveform Image Generator

audiowaveform is a C++ command-line application that generates waveform data from either MP3, WAV, FLAC, or Ogg Vorbis format audio files. Waveform data can be used to produce a visual rendering of the audio, similar in appearance to audio editing applications. Waveform data files are saved in either binary format (.dat) or JSON (.json).

To install BBC Audio Waveform Image Generator in the Workspace container

1 - Open the `.env` file

2 - Search for the `WORKSPACE_INSTALL_AUDIOWAVEFORM` argument under the Workspace Container and set it to `true`

3 - Re-build the container `docker-compose build workspace`

4 - If you use the `php-worker` or `laravel-horizon` container too, please follow the same steps above especially if you have processing that have been queued.

PS Don't forget to install the binary in the `php-fpm` container too by applying the same steps above to its container, otherwise you'll get an error when running the `audiowaveform` binary.

Install wkhtmltopdf

wkhtmltopdf is a utility for outputting a PDF from HTML

To install wkhtmltopdf in the Workspace container

1 - Open the `.env` file

2 - Search for the `WORKSPACE_INSTALL_WKHTMLTOPDF` argument under the Workspace Container and set it to `true`

3 - Re-build the container `docker-compose build workspace`

PS Don't forget to install the binary in the `php-fpm` container too by applying the same steps above to its container, otherwise the you'll get an error when running the `wkhtmltopdf` binary.

Install GNU Parallel

GNU Parallel is a command line tool to run multiple processes in parallel.

(see https://www.gnu.org/software/parallel/parallel_tutorial.html)

To install GNU Parallel in the Workspace container

1 - Open the `.env` file

2 - Search for the `WORKSPACE_INSTALL_GNU_PARALLEL` argument under the Workspace Container and set it to `true`

3 - Re-build the container `docker-compose build workspace`

Install Supervisor

Supervisor is a client/server system that allows its users to monitor and control a number of processes on UNIX-like operating systems.

(see http://supervisord.org/index.html)

To install Supervisor in the Workspace container

1 - Open the `.env` file

2 - Set `WORKSPACE_INSTALL_SUPERVISOR` and `WORKSPACE_INSTALL_PYTHON` to `true`.

3 - Create supervisor configuration file (for ex., named `laravel-worker.conf`) for Laravel Queue Worker in `php-worker/supervisord.d/` by simply copy from `laravel-worker.conf.example`

4 - Re-build the container `docker-compose build workspace` Or `docker-compose up --build -d workspace`

Common Terminal Aliases

When you start your docker container, Laradock will copy the `aliases.sh` file located in the `laradock/workspace` directory and add sourcing to the container `~/.bashrc` file.

You are free to modify the `aliases.sh` as you see fit, adding your own aliases (or function macros) to suit your requirements.

Install Aerospike extension

1 - First install `aerospike` in the Workspace and the PHP-FPM Containers:
a) open the `.env` file
b) search for the `WORKSPACE_INSTALL_AEROSPIKE` argument under the Workspace Container
c) set it to `true`
d) search for the `PHP_FPM_INSTALL_AEROSPIKE` argument under the PHP-FPM Container
e) set it to `true`

2 - Re-build the containers `docker-compose build workspace php-fpm`

Install Laravel Envoy

A Tasks Runner.

1 - Open the `.env` file
2 - Search for the `WORKSPACE_INSTALL_LARAVEL_ENVOY` argument under the Workspace Container
3 - Set it to `true`

4 - Re-build the containers `docker-compose build workspace`

Laravel Envoy Documentation Here

Install php calendar extension

1 - Open the `.env` file
2 - Search for the `PHP_FPM_INSTALL_CALENDAR` argument under the PHP-FPM container
3 - Set it to `true`
4 - Re-build the containers `docker-compose build php-fpm`

Install libfaketime in php-fpm

Libfaketime allows you to control the date and time that is returned from the operating system. It can be used by specifying a special string in the `PHP_FPM_FAKETIME` variable in the `.env` file. For example: `PHP_FPM_FAKETIME=-1d` will set the clock back 1 day. See (https://github.com/wolfcw/libfaketime) for more information.

1 - Open the `.env` file
2 - Search for the `PHP_FPM_INSTALL_FAKETIME` argument under the PHP-FPM container
3 - Set it to `true`
4 - Search for the `PHP_FPM_FAKETIME` argument under the PHP-FPM container
5 - Set it to the desired string
6 - Re-build the containers `docker-compose build php-fpm`

Install YAML extension in php-fpm

YAML PHP extension allows you to easily parse and create YAML structured data. I like YAML because it's well readable for humans. See http://php.net/manual/en/ref.yaml.php and http://yaml.org/ for more info.

1 - Open the `.env` file
2 - Search for the `PHP_FPM_INSTALL_YAML` argument under the PHP-FPM container
3 - Set it to `true`
4 - Re-build the container `docker-compose build php-fpm`

Install RDKAFKA extension in php-fpm

1 - Open the `.env` file
2 - Search for the `PHP_FPM_INSTALL_RDKAFKA` argument under the PHP-FPM container
3 - Set it to `true`
4 - Re-build the container `docker-compose build php-fpm`

Install RDKAFKA extension in workspace

This is needed for 'composer install' if your dependencies require Kafka.

1 - Open the `.env` file
2 - Search for the `WORKSPACE_INSTALL_RDKAFKA` argument under the WORKSPACE container
3 - Set it to `true`
4 - Re-build the container `docker-compose build workspace`

Install AST PHP extension

AST exposes the abstract syntax tree generated by PHP 7+. This extension is required by tools such as `Phan`, a static analyzer for PHP.

1 - Open the `.env` file

2 - Search for the `WORKSPACE_INSTALL_AST` argument under the Workspace Container

3 - Set it to `true`

4 - Re-build the container `docker-compose build workspace`

Note If you need a specific version of AST then search for the `WORKSPACE_AST_VERSION` argument under the Workspace Container and set it to the desired version and continue step 4.

Install Git Bash Prompt

A bash prompt that displays information about the current git repository. In particular the branch name, difference with remote branch, number of files staged, changed, etc.

1 - Open the `.env` file

2 - Search for the `WORKSPACE_INSTALL_GIT_PROMPT` argument under the Workspace Container

3 - Set it to `true`

4 - Re-build the container `docker-compose build workspace`

Note You can configure bash-git-prompt by editing the `workspace/gitprompt.sh` file and re-building the workspace container. For configuration information, visit the bash-git-prompt repository.

Install Oh My ZSH

With the Laravel autocomplete plugin.

Zsh is an extended Bourne shell with many improvements, including some features of Bash, ksh, and tcsh.

Oh My Zsh is a delightful, open source, community-driven framework for managing your Zsh configuration.

Laravel autocomplete plugin adds aliases and autocompletion for Laravel Artisan and Bob command-line interfaces.

1 - Open the `.env` file

2 - Search for the `SHELL_OH_MY_ZSH` argument under the Workspace Container

3 - Set it to `true`

4 - Re-build the container `docker-compose build workspace`

5 - Use it `docker-compose exec --user=laradock workspace zsh`

Note You can configure Oh My ZSH by editing the `/home/laradock/.zshrc` in running container.

With the ZSH autosuggestions plugin.

ZSH autosuggestions plugin suggests commands as you type based on history and completions.

1 - Enable ZSH as described previously

2 - Set `SHELL_OH_MY_ZSH_AUTOSUGESTIONS` to `true`

3 - Rebuild and use ZSH as described previously

> With bash aliases loaded.

Laradock provides aliases through the `aliases.sh` file located in the `laradock/workspace` directory. You can load it into ZSH.

1 - Enable ZSH as described previously

2 - Set `SHELL_OH_MY_ZSH_ALIASES` to `true`

3 - Rebuild and enjoy aliases

PHPStorm Debugging Guide

Remote debug Laravel web and phpunit tests.

Debugging Guide Here

Setup Google Cloud

> Setting up Google Cloud for the docker registry.

```
gcloud auth configure-docker
```

Login to gcloud for use the registry and auth the permission.

```
gcloud auth login
```

Track your Laradock changes

1. Fork the Laradock repository.
2. Use that fork as a submodule.
3. Commit all your changes to your fork.
4. Pull new stuff from the main repository from time to time.

Improve speed on MacOS

Docker on the Mac is slow, at the time of writing. Especially for larger projects, this can be a problem. The problem is older than March 2016 - as it's a such a long-running issue, we're including it in the docs here.

So since sharing code into Docker containers with osxfs have very poor performance compared to Linux. Likely there are some workarounds:

Workaround A: using dinghy

Dinghy creates its own VM using docker-machine, it will not modify your existing docker-machine VMs.

Quick Setup giude, (we recommend you check their docs)

1) `brew tap codekitchen/dinghy`

2) `brew install dinghy`

3) `dinghy create --provider virtualbox` (must have virtualbox installed, but they support other providers if you prefer)

4) after the above command is done it will display some env variables, copy them to the bash profile or zsh or.. (this will instruct docker to use the server running inside the VM)

5) `docker-compose up ...`

Workaround B: using d4m-nfs

You can use the d4m-nfs solution in 2 ways, the first is by using the built-in Laradock integration, and the second is using the tool separately. Below is show case of both methods:

B.1: using the built in d4m-nfs integration

In simple terms, docker-sync creates a docker container with a copy of all the application files that can be accessed very quickly from the other containers. On the other hand, docker-sync runs a process on the host machine that continuously tracks and updates files changes from the host to this intermediate container.

Out of the box, it comes pre-configured for OS X, but using it on Windows is very easy to set-up by modifying the `DOCKER_SYNC_STRATEGY` on the `.env`

Usage

Laradock comes with `sync.sh`, an optional bash script, that automates installing, running and stopping docker-sync. Note that to run the bash script you may need to change the permissions `chmod 755 sync.sh`

1) Configure your Laradock environment as you would normally do and test your application to make sure that your sites are running correctly.

2) Make sure to set `DOCKER_SYNC_STRATEGY` on the `.env`. Read the syncing strategies for details.

```
# osx: 'native_osx' (default)
# windows: 'unison'
# linux: docker-sync not required

DOCKER_SYNC_STRATEGY=native_osx
```

3) set `APP_CODE_CONTAINER_FLAG` to `APP_CODE_CONTAINER_FLAG=:nocopy` in the .env file

4) Install the docker-sync gem on the host-machine:

```
./sync.sh install
```

5) Start docker-sync and the Laradock environment. Specify the services you want to run, as you would normally do with `docker-compose up`

```
./sync.sh up nginx mysql
```

Please note that the first time docker-sync runs, it will copy all the files to the intermediate container and that may take a very long time (15min+). 6) To stop the environment and docker-sync do:

```
./sync.sh down
```

Setting up Aliases (optional)

You may create bash profile aliases to avoid having to remember and type these commands for everyday development. Add the following lines to your `~/.bash_profile`:

```
alias devup="cd /PATH_TO_LARADOCK/laradock; ./sync.sh up nginx mysql" #add your
services
alias devbash="cd /PATH_TO_LARADOCK/laradock; ./sync.sh bash"
alias devdown="cd /PATH_TO_LARADOCK/laradock; ./sync.sh down"
```

Now from any location on your machine, you can simply run `devup`, `devbash` and `devdown`.

Additional Commands

Opening bash on the workspace container (to run artisan for example):

```
 ./sync.sh bash
```

Manually triggering the synchronization of the files:

```
./sync.sh sync
```

Removing and cleaning up the files and the docker-sync container. Use only if you want to rebuild or remove docker-sync completely. The files on the host will be kept untouched.

```
./sync.sh clean
```

Additional Notes

- You may run laradock with or without docker-sync at any time using with the same `.env` and `docker-compose.yml`, because the configuration is overridden automatically when docker-sync is used.
- You may inspect the `sync.sh` script to learn each of the commands and even add custom ones.
- If a container cannot access the files on docker-sync, you may need to set a user on the Dockerfile of that container with an id of 1000 (this is the UID that nginx and php-fpm have configured on laradock). Alternatively, you may change the permissions to 777, but this is **not** recommended.

Visit the docker-sync documentation for more details.

B.2: using the d4m-nfs tool

D4m-nfs automatically mount NFS volume instead of osxfs one.

1) Update the Docker [File Sharing] preferences:

Click on the Docker Icon > Preferences > (remove everything form the list except `/tmp`).

2) Restart Docker.

3) Clone the d4m-nfs repository to your `home` directory.

```
git clone https://github.com/IFSight/d4m-nfs ~/d4m-nfs
```

4) Create (or edit) the file `~/d4m-nfs/etc/d4m-nfs-mounts.txt`, and write the following configuration in it:

```
/Users:/Users
```

5) Create (or edit) the file `/etc/exports`, make sure it exists and is empty. (There may be collisions if you come from Vagrant or if you already executed the `d4m-nfs.sh` script before).

6) Run the `d4m-nfs.sh` script (might need Sudo):

```
~/d4m-nfs/d4m-nfs.sh
```

That's it! Run your containers.. Example:

```
docker-compose up ...
```

Note: If you faced any errors, try restarting Docker, and make sure you have no spaces in the `d4m-nfs-mounts.txt` file, and your `/etc/exports` file is clear.

Upgrade Laradock

Moving from Docker Toolbox (VirtualBox) to Docker Native (for Mac/Windows). Requires upgrading Laradock from v3.* to v4.*:

1. Stop the docker VM `docker-machine stop {default}`
2. Install Docker for Mac or Windows.
3. Upgrade Laradock to `v4.*.*` (`git pull origin master`)
4. Use Laradock as you used to do: `docker-compose up -d nginx mysql`.

Note: If you face any problem with the last step above: rebuild all your containers `docker-compose build --no-cache` "Warning Containers Data might be lost!"

Help & Questions

Join the chat room on Gitter and get help and support from the community.

You can as well can open an issue on Github (will be labeled as Question) and discuss it with people on Gitter.

Common Problems

Here's a list of the common problems you might face, and the possible solutions.

I see a blank (white) page instead of the Laravel 'Welcome' page!

Run the following command from the Laravel root directory:

```
sudo chmod -R 777 storage bootstrap/cache
```

I see "Welcome to nginx" instead of the Laravel App!

Use `http://127.0.0.1` instead of `http://localhost` in your browser.

I see an error message containing (address already in use) or (port is already allocated)

Make sure the ports for the services that you are trying to run (22, 80, 443, 3306, etc.) are not being used already by other programs on the host, such as a built in `apache`/`httpd` service or other development tools you have installed.

I get NGINX error 404 Not Found on Windows.

1. Go to docker Settings on your Windows machine.
2. Click on the `Shared Drives` tab and check the drive that contains your project files.
3. Enter your windows username and password.
4. Go to the `reset` tab and click restart docker.

The time in my services does not match the current time

1. Make sure you've changed the timezone.
2. Stop and rebuild the containers (`docker-compose up -d --build <services>`)

I get MySQL connection refused

This error sometimes happens because your Laravel application isn't running on the container localhost IP (Which is 127.0.0.1). Steps to fix it:

- Option A
    1. Check your running Laravel application IP by dumping `Request::ip()` variable using `dd(Request::ip())` anywhere on your application. The result is the IP of your Laravel container.
    2. Change the `DB_HOST` variable on env with the IP that you received from previous step.
- Option B
    1. Change the `DB_HOST` value to the same name as the MySQL docker container. The Laradock docker-compose file currently has this as `mysql`

I get stuck when building nginx on (fetch mirrors.aliyun.com/alpine/v3.5/main/x86_64/APKINDEX.tar.gz)

As stated on #749, Already fixed, just set `CHANGE_SOURCE` to false.

Custom composer repo packagist url and npm registry url

In China, the origin source of composer and npm is very slow. You can add `WORKSPACE_NPM_REGISTRY` and `WORKSPACE_COMPOSER_REPO_PACKAGIST` config in `.env` to use your custom source.

Example:

```
WORKSPACE_NPM_REGISTRY=https://registry.npm.taobao.org
WORKSPACE_COMPOSER_REPO_PACKAGIST=https://packagist.phpcomposer.com
```

I got (Module build failed: Error: write EPIPE) while compiling react application

When you run `npm build` or `yarn dev` building a react application using webpack with elixir you may receive an `Error: write EPIPE` while processing .jpg images.

This is caused of an outdated library for processing **.jpg files** in ubuntu 16.04.

To fix the problem you can follow those steps

1 - Open the `.env`.

2 - Search for `WORKSPACE_INSTALL_LIBPNG` or add the key, if missing.

3 - Set the value to true:

```
WORKSPACE_INSTALL_LIBPNG=true
```

4 - Finally rebuild the workspace image

```
docker-compose build workspace
```

Related Projects

Laradock Related Projects

- Docker Stacks: A GUI for managing Laradock. (by Subhadip Naskar)
- Laradock CLI: A CLI for managing Laradock. (by Lorin Lee)
- Laradock CLI: A CLI for managing Laradock. (by Harlan Wilton)
- Ansible Laradock Kubernetes: Ansible playbook to setup docker containers for Laravel apps using Laradock. (by Sifat Rahim)
- Monitor Laradock: Laradock Monitoring Tools (using Grafana). (by Dwi Fahni Denni)
- Laradock Manager: A simple app for managing Laradock containers. Made with wails.app (go & vue.js & vuetify). (by Zámbó Levente)
- Laradock Env: A wrapper with commands for managing Laradock. (by BAG Art)
- Lara Query: Easy Laradock CLI. (by Okita kamegoro)
- Laradock CLI: Laradock CLI helper. (by Tony Messias)
- Laradock Lite: A Docker based laravel development environment with minimal dependencies. (by Liu Yang)
- Laradock Makefile: Makefile with some useful commands for Laradock. (by Dmitry Bazavluk)

- Laradock Build: Docker builder & running script for Laradock. (by Docker Framework)
- Laravel Laradock PHPStorm: Guide for configuring PHPStorm for remote debugging with Laravel & Laradock. (by Larry Eitel)
- Laradock Crudbooster: Docker compose & Kubernetes solution to build apps with crudbooster & Laradock. (by Samuele Chiocca)
- Laradock Sample: Install Laravel with Laradock. (by Tadaken3)
- Stylemix's Laradock: Alternate laradock for multiproject purpose. (by Stylemix LLC)

Inspired by Laradock

- Dockery: Laradock for Ruby. (by Taufek Johar)
- RubyDev Dock: Laradock for Ruby. (by Diogo Scudelletti)
- NoDock: Laradock for NodeJS. (by Osedea)
- Laradock Multi: Laradock for PHP & NodeJS. (by BAG Art)
- Wordpress Laradock: Laradock for Wordpress. (by Alexandr Shevchenko)
- Yii2 Laradock: Laradock for Yii2. (by Yuda Sukmana)
- MageDock: Laradock for Magento. (by Ujjwal Ojha)
- Docker Codeigniter: Laradock for Codeigniter. (by Sebastian)
- Klaradock: A customized Laradock. (by Kim Hsiao)
- Laravel Boilerplate: A boilerplate with support for JWT. (by Casiva Agustin)

> Feel free to submit a PR for listing your project here.

Contributions

Have a Question

If you have questions about how to use Laradock, please direct your questions to the discussion on Gitter. If you believe your question could help others, then consider opening an Issue (it will be labeled as `Question`) And you can still seek help on Gitter for it.

Found an Issue

If you have an issue or you found a typo in the documentation, you can help us by opening an Issue.

Steps to do before opening an Issue:

1. Before you submit your issue search the archive, maybe your question was already answered couple hours ago (search in the closed Issues as well).
2. Decide if the Issue belongs to this project or to Docker itself! or even the tool you are using such as Nginx or MongoDB…

If your issue appears to be a bug, and hasn't been reported, then open a new issue.

This helps us maximize the effort we can spend fixing issues and adding new features, by not reporting duplicate issues.

Want a Feature

You can request a new feature by submitting an Issue (it will be labeled as `Feature Suggestion`). If you would like to implement a new feature then consider submitting a Pull Request yourself.

Update the Documentation (Site)

Laradock uses Hugo as website generator tool, with the Material Docs theme. You might need to check their docs quickly.

Go the `DOCUMENTATION/content` and search for the markdown file you want to edit

Note: Every folder represents a section in the sidebar "Menu". And every page and sidebar has a `weight` number to show it's position in the site.

To update the sidebar or add a new section to it, you can edit this `DOCUMENTATION/config.toml` toml file.

> The site will be auto-generated in the `docs/` folder by Travis CI.

Host the documentation locally

Option 1: Use Hugo Docker Image:

1. Update the `DOCUMENTATION/content`.
2. Go to `DOCUMENTATION/`.
3. Run `docker run --rm -it -v $PWD:/src -p 1313:1313 -u hugo jguyomard/hugo-builder hugo server -w --bind=0.0.0.0`
4. Visit http://localhost:1313/

Option 2: Install Hugo Locally:

1. Install Hugo on your machine.
2. Update the `DOCUMENTATION/content`.
3. Delete the `/docs` folder from the root.
4. Go to `DOCUMENTATION/`.
5. Run the `hugo` command to generate the HTML docs inside a new `/docs` folder.

Support new Software (Add new Container)

- Fork the repo and clone the code.
- Create folder as the software name (example: `mysql-nginx`).
- Add your `Dockerfile` in the folder “you may add additional files as well”.
- Add the software to the `docker-compose.yml` file.
- Make sure you follow the same code/comments style.
- Add the environment variables to the `env-example` if you have any.
- **MOST IMPORTANTLY** update the `Documentation`, add as much information.
- Submit a Pull Request, to the `master` branch.

Edit supported Software (Edit a Container)

- Fork the repo and clone the code.
- Open the software (container) folder (example: `mysql-nginx`).
- Edit the files.
- Make sure to update the `Documentation` in case you made any changes.
- Submit a Pull Request, to the `master` branch.

Edit Base Image

- Open any dockerfile, copy the base image name (example: `FROM phusion/baseimage:latest`).
- Search for the image in the Docker Hub and find the source..

Most of the image in Laradock are official images, these projects live in other repositories and maintainer by other organizations.

Note: Laradock has two base images for (`Workspace` and `php-fpm`, mainly made to speed up the build time on your machine.

- Find the dockerfiles, edit them and submit a Pull Request.
- When updating a Laradock base image (`Workspace` or `php-fpm`), ask a project maintainer “Admin” to build a new image after your PR is merged.

Note: after the base image is updated, every dockerfile that uses that image, needs to update his base image tag to get the updated code.

Submit Pull Request Instructions

1. Before Submitting a Pull Request (PR)

Always Test everything and make sure its working:

- Pull the latest updates (or fork of you don't have permission)
- Before editing anything:
    - Test building the container (docker-compose build –no-cache container-name) build with no cache first.
    - Test running the container with some other containers in real app and see of everything is working fine.
- Now edit the container (edit section by section and test rebuilding the container after every edited section)
    - Testing building the container (docker-compose build container-name) with no errors.
    - Test it in a real App if possible.

2. Submitting a PR

Consider the following guidelines:

- Search GitHub for an open or closed Pull Request that relates to your submission. You don't want to duplicate efforts.
- Make your changes in a new git branch:

  ```
  git checkout -b my-fix-branch master
  ```

- Commit your changes using a descriptive commit message.
- Push your branch to GitHub:

  ```
  git push origin my-fix-branch
  ```

- In GitHub, send a pull request to `laradock:master`.
- If we suggest changes then:
    - Make the required updates.
    - Commit your changes to your branch (e.g. `my-fix-branch`).
    - Push the changes to your GitHub repository (this will update your Pull Request).

If the PR gets too outdated we may ask you to rebase and force push to update the PR:

```
git rebase master -i
git push origin my-fix-branch -f
```

WARNING. Squashing or reverting commits and forced push thereafter may remove GitHub comments on code that were previously made by you and others in your commits.

3. After your PR is merged

After your pull request is merged, you can safely delete your branch and pull the changes from the main (upstream) repository:

- Delete the remote branch on GitHub either through the GitHub web UI or your local shell as follows:

  ```
  git push origin --delete my-fix-branch
  ```

- Check out the master branch:

  ```
  git checkout master -f
  ```

- Delete the local branch:

  ```
  git branch -D my-fix-branch
  ```

- Update your master with the latest upstream version:

  ```
  git pull --ff upstream master
  ```

Happy Coding :)

https://laradock.io/