docker-sync Documentation

Release 0.5.11

EugenMayer

Feb 17, 2020

Introduction

Run your application at full speed while syncing your code for development, finally empowering you to utilize docker for development under OSX/Windows/Linux*

CHAPTER 1

Introduction

Developing with docker under OSX/ Windows is a huge pain, since sharing your code into containers will slow down the code-execution about 60 times (depends on the solution). Testing and working with a lot of the alternatives made us pick the best of those for each platform, and combine this in one single tool: `docker-sync`.

- For OSX, see *OSX*.
- For Windows, see *Windows*.
- For Linux, see *Linux*.
- See the list of alternatives at *Alternatives*

1.1 Features

- Support for OSX, Windows, Linux and FreeBSD
- Runs on Docker for Mac, Docker for Windows and Docker Toolbox
- Uses either native_osx, unison or rsync as possible strategies. The container performance is not influenced at all, see *Performance*
- Very efficient due to the *native_osx in depth*
- Without any dependencies on OSX when using (native_osx)
- Backgrounds as a daemon
- Supports multiple sync-end points and multiple projects at the same time
- Supports user-remapping on sync to avoid permission problems on the container
- Can be used with your docker-compose way or the integrated docker-compose way to start the stack and sync at the same time with one command
- Using overlays to keep your production docker-compose.yml untouched and portable. See *docker-compose.yml*.
- Supports Linux* to use the same toolchain across all platforms, but maps on a native mount in linux (no sync)

- Besides performance being the first priority for docker-sync, the second is, not forcing you into using a specific docker solution. Use docker-for-mac, docker toolbox, VirtualBox, VMware Fusion or Parallels, xhyve or whatever!

1.1.1 Installation

No matter if its OSX/Linux/Windows

```
gem install docker-sync
```

Depending on the OS, you might need more steps to setup. Continue reading below.

Installation as non-root

You may elect to install `docker-sync` as a non-root user. To install as the current user:

```
gem install --user-install docker-sync
```

Then set-up your shell to add the `docker-sync` command to the `PATH`. Edit `.bashrc`:

```
if which ruby >/dev/null && which gem >/dev/null; then
  PATH="$(ruby -r rubygems -e 'puts Gem.user_dir')/bin:$PATH"
fi
```

... then start a new shell and run `docker-sync`.

OSX

Dependencies

With native_osx we no longer have any host dependencies.

Advanced / optional

Optionally, if you do not want to use unison or want a better rsync or use unison (than the built-in OS X one)

if you use unison

```
brew install unison
brew install eugenmayer/dockersync/unox
```

if you use rsync

```
brew install rsync
```

Homebrew aka brew is a tool you need under OSX to install / easy compile other tools. You can use other tools/ways to install or compile fswatch, but those are out of scope for this docs. All we need is the binary in PATH.

Linux

> **Caution:** Linux support is still to be considered BETA - do not get too crazy if we have bugs!

Dependencies

Default sync strategy for linux (native) do not need any host dependencies.

Advanced / optional

Optionally, if you want to use unison, then you would need to install some more stuff.

The following instructions are written for Ubuntu; if you're using another linux flavor, you might need to adjust some stuff like using a different package manager.

Using Unison Strategy

The Ubuntu package for unison doesn't come with unison-fsmonitor, as such, we would need to build from source.

```
sudo apt-get install build-essential ocaml
wget https://github.com/bcpierce00/unison/archive/v2.51.2.tar.gz
tar xvf v2.51.2.tar.gz
cd unison-2.51.2
make UISTYLE=text
sudo cp src/unison /usr/local/bin/unison
sudo cp src/unison-fsmonitor /usr/local/bin/unison-fsmonitor
```

and that should be enough to get you up and running using unison.

Using rsync strategy

rsync strategy is not currently supported under linux, but it can be done. If you need this, please see #386, and send us some help.

Windows

> **Caution:** Windows support is still to be considered BETA, - do not get too crazy if there are some bugs!

This guide provides detailed instructions on getting docker-sync running on Windows Subsystem for Linux.

As the time goes by these instructions may not be updated, so please also check out the repo's issues if you have any 'unknown' problem that is not treated in this guide.

Still the procedure is pretty straightforward and should help set you up and running without too much hassle.

Benefits of Docker-sync on Windows

- Inotify works on containers that support it. No more polling!
- Performance might be a bit better or right on par with native Windows volumes. This needs more testing.

Possible Future Supported Environments

- Cygwin
- Native Windows (no posix)

My Setup (for reference)

Windows 10 Pro 1709

Pro version required for using Docker for Windows (Hyper-V), also update your system to the latest available version from MS

Docker for Windows CE 18.03.0-ce-rc3-win56 (16433) edge

(stable version should also work fine)

Let's go!

1. Enable WSL Open the Windows Control Panel, Programs and Features, click on the left on Turn Windows features on or off and check Windows Subsystem for Linux near the bottom.

2. Install a distro Open the Microsoft Store and search for 'linux'.

You will be then able to choose and install Debian, SUSE, openSUSE, Ubuntu, etc..

In this guide I am using Debian GNU/Linux. Direct link for Debian GNU/Linux

3. Launch and update The distro you choose is now an 'app' on your system.

Open the start menu and launch it, then follow the on screen instructions in order to complete the installation.

When you have a fully working shell, update the system.

```
sudo apt update

sudo apt upgrade
```

4. Install Docker Follow the official documentation for installing Docker on Linux: (the following is for Debian)

https://docs.docker.com/install/linux/docker-ce/debian/#install-docker-ce

Note that the Docker Server doesn't work on the subsystem - we will then expose Docker for Windows to WSL later

with Windows 10 >= 1803 you can place a symlink to the Windows binary

```
sudo ln -s "/mnt/c/Program Files/Docker/Docker/resources/bin/docker.exe" /usr/local/
↪bin/docker
```

5. Install Docker Compose

```
sudo apt install docker-compose
```

Or if that does not work, follow the official documentation: https://docs.docker.com/compose/install/

with Windows 10 >= 1803 you can place a symlink to the Windows binary

```
sudo ln -s "/mnt/c/Program Files/Docker/Docker/resources/bin/docker-compose.exe" /usr/
↪local/bin/docker-compose
```

6. Install Ruby and Ruby-dev

```
sudo apt-get install ruby ruby-dev
```

7. Install docker-sync

Install the gem

```
sudo gem install docker-sync
```

8. Set your Docker for Windows host as an ENV variable

Open the Docker for Windows settings and check Expose daemon on tcp://localhost:2375 without TLS

Then type the following command in your WSL shell.

```
echo "export DOCKER_HOST=tcp://127.0.0.1:2375" >> ~/.bashrc
```

9. Compile and install OCaml

Before doing this please check out first the eugenmayer/unison dockerfile and ensure that the OCaml version that you are going to install is the same. To find the required OCaml version, do a search for "ocaml" within the eugenmayer/unison's dockerfile (https://github.com/EugenMayer/docker-image-unison/blob/master/Dockerfile)

Install build script

```
sudo apt-get install build-essential
```

As for now the procedure is as follows:

```
sudo apt-get install make
wget http://caml.inria.fr/pub/distrib/ocaml-4.08/ocaml-4.08.1.tar.gz
tar xvf ocaml-4.08.1.tar.gz
cd ocaml-4.08.1
./configure
make world
make opt
umask 022
sudo make install
sudo make clean
```

10. Compile and install Unison

Look up the latest Unison release (https://github.com/bcpierce00/unison/releases), download the source code, compile and install.

As for now the procedure is as follows:

```
wget https://github.com/bcpierce00/unison/archive/v2.51.2.tar.gz
tar xvf v2.51.2.tar.gz
cd unison-2.51.2
# The implementation src/system.ml does not match the interface system.cmi:curl and␣
↪needs to be patched
curl https://github.com/bcpierce00/unison/commit/23fa1292.diff?full_index=1 -o patch.
↪diff
git apply patch.diff
make UISTYLE=text
sudo cp src/unison /usr/local/bin/unison
sudo cp src/unison-fsmonitor /usr/local/bin/unison-fsmonitor
```

11. Set Timezone if not done already

Check if /etc/localtime is a symlink. If not run dpkg-reconfigure tzdata and set your correct timezone.

12. (bonus!) Bind custom mount points to fix Docker for Windows and WSL differences (thanks to @nickjanetakis)

You might encounter various strange problems with volumes while starting up Docker containers from WSL.

If so, as a workaround you have to set up a special mountpoint inside /etc/fstab and start your container from there.

```
sudo mkdir /c
sudo mount --bind /mnt/c /c
echo "sudo mount --bind /mnt/c /c" >> ~/.bashrc && source ~/.bashrc
```

In order to automatically mount the volume without asking any password you can add a rule into your sudoers file.

```
sudo visudo
```

Add the following at the bottom of the file, replacing "username" with your WSL username.

```
username ALL=(root) NOPASSWD: /bin/mount
```

with Windows 10 >= 1803 you can place a new file to /etc/wsl.conf instead

```
[automount]
root = /
options = "metadata"
```

12. Laradock? No problem!

If, as an example, you are using Laradock, you just need to follow the official documentation changing the sync strategy to 'unison' and adding the docker-compose.sync.yml in your .env file.

```
...
COMPOSE_PATH_SEPARATOR=;
COMPOSE_FILE=docker-compose.yml:docker-compose.dev.yml:docker-compose.sync.yml
...
DOCKER_SYNC_STRATEGY=unison
```

Then you need to add the following 'sync_args' line in the laradock/docker-sync.yml file, as follows:

```
...
sync_strategy: '${DOCKER_SYNC_STRATEGY}' # for osx use 'native_osx', for windows use
↪'unison'

sync_args: ['-perms=0'] #required for two way sync ie generators, etc
...
```

This will allow proper synchronization between the Linux containers and your Windows host that manages permissions in a different way.

Now you can start syncing using sync.sh provided with Laradock.

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

Done!

You should now have a working version of docker-sync via the Unison strategy.

In your home directory in WSL you can link your projects from Windows and run docker-sync or docker-sync-stack.

The rest of your workflow should be the same as before in either Command Prompt, PowerShell, or some other Windows terminal.

FYI - An example of a docker-sync.yml file

```
version: "2"
options:
    verbose: true
syncs:
    app-unison-sync: # tip: add -sync and you keep consistent names als a convention
        sync_args: ['-perms=0'] #required for two way sync ie generators, etc
        sync_strategy: 'unison'
        sync_host_ip: '127.0.0.1' #host ip isn't properly inferred
        sync_excludes: ['.gitignore', '.idea/*','.git/*', '*.coffee', '*.scss', '*.
↪sass','*.log']
        src: './'
```

FreeBSD

Dependencies

Default sync strategy for FreeBSD is `rsync`, you need to install it first:

```
# pkg install rsync
```

Using `rsync`

To setup an rsync resource you need a `docker-sync.yml` similar to:

```
version: "2"

syncs:
  code-sync:
    sync_strategy: "rsync"
    src: "path/to/src"
    sync_host_port: 10871
    # sync_host_allow: "..."
```

`sync_host_port` is mandatory and it must be unique for this shared resource.

You might need to specify `sync_host_allow`, this will let the rsync daemon know from which IP to expect connections from, network format (`10.0.0.0/8`) or an specific IP (`10.2.2.2`) is supported. The value depends on your virtualization solution and network stack defined (`NAT` vs `host-only`). A quick way to determine the value is to run `docker-sync start` and let it fail, the error will show you the needed IP value.

Using `unison`

`unison` could be supported on FreeBSD, but it wasn't tested yet.

Using `native_osx`

This strategy is not supported, its OSX only.

1.1.2 Configuration

Caution: When you change anything in your docker-sync.yml be sure to run docker-sync clean and docker-sync start right after it. Just running docker-sync start or stop will not recreate the container and your changes will have no effect!

docker-sync.yml

The file docker-sync.yml should be placed in the top-level folder of your project, so docker-sync can find it. The configuration will be searched from the point you run docker-sync from, traversing up the path tree

In there, you usually configure one (or more) sync points. Be sure to decide which sync-strategy you want to chose, see *Sync strategies*.

Below are all the available options, simple examples can be found in the docker-sync-boilerplate.

Important: Be sure to use a sync-name which is unique, since it will be a container name. Do not use your app name, but rather app-sync.

References

Caution: This is a configuration reference. **Do not use all options at once, they do not make sense!** Or copy them as your starting point, rather use the docker-sync-boilerplate and then cherry pick the options you need.

```
options:
  # default: docker-compose.yml if you like, you can set a custom location (path) of␣
↪your compose file like ~/app/compose.yml
  # HINT: you can also use this as an array to define several compose files to␣
↪include. Order is important!
  compose-file-path: 'docker-compose.yml'

  # optional, default: docker-compose-dev.yml if you like, you can set a custom␣
↪location (path) of your compose file. Do not set it, if you do not want to use it␣
↪at all

  # if its there, it gets used, if you name it explicitly, it HAS to exist
  # HINT: you can also use this as an array to define several compose files to␣
↪include. Order is important!
  compose-dev-file-path: 'docker-compose-dev.yml'

  # optional, activate this if you need to debug something, default is false
  # IMPORTANT: do not run stable with this, it creates a memory leak, turn off␣
↪verbose when you are done testing
  verbose: false

  # ADVANCED: the image to use for the rsync container. Do not change this until you␣
↪exactly know, what you are doing
   # replace <sync_strategy> with either rsync, unison, native_osx to set a custom␣
↪image for all sync of this type
   # do not do that if you really do not need that!
```

(continues on next page)

(continued from previous page)

```
  <sync_strategy>_image: 'yourcustomimage'

  # optional, default auto, can be docker-sync, thor or auto and defines how the sync␣
↪will be invoked on the cli. Mostly depending if your are using docker-sync solo,␣
↪scaffolded or in development ( thor )
  cli_mode: 'auto'
  # optional, maximum number of attempts for unison waiting for the success exit␣
↪status. The default is 5 attempts (1-second sleep for each attempt). Only used in␣
↪unison.
  max_attempt: 5

  # optional, default: pwd, root directory to be used when transforming sync src into␣
↪absolute path, accepted values: pwd (current working directory), config_path (the␣
↪directory where docker-sync.yml is found)
  project_root: 'pwd'

syncs:
  default-sync:
    # os aware sync strategy, defaults to native_osx under MacOS (except with docker-
↪machine which use unison), and native docker volume under linux
    # remove this option to use the default strategy per os or set a specific one
    sync_strategy: 'native_osx'
    # which folder to watch / sync from - you can use tilde, it will get expanded.
    # the contents of this directory will be synchronized to the Docker volume with␣
↪the name of this sync entry ('default-sync' here)
    src: './default-data/'

    host_disk_mount_mode: 'cached' # see https://docs.docker.com/docker-for-mac/osxfs-
↪caching/#cached
    # other unison options can also be specified here, which will be used when run␣
↪under osx,
    # and ignored when run under linux

  # IMPORTANT: this name must be unique and should NOT match your real application␣
↪container name!
  fullexample-sync:
    # enable terminal_notifier. On every sync sends a Terminal Notification regarding␣
↪files being synced. ( Mac Only ).
    # good thing in case you are developing and want to know exactly when your␣
↪changes took effect.
    # be aware in case of unison this only gives you a notification on the initial␣
↪sync, not the syncs after changes.
    notify_terminal: true

    # which folder to watch / sync from - you can use tilde (~), it will get expanded.
↪ Be aware that the trailing slash makes a difference
    # if you add them, only the inner parts of the folder gets synced, otherwise the␣
↪parent folder will be synced as top-level folder
    src: './data1'

    # when a port of a container is exposed, on which IP does it get exposed.␣
↪Localhost for docker for mac, something else for docker-machine
    # default is 'auto', which means, your docker-machine/docker host ip will be␣
↪detected automatically. If you set this to a concrete IP, this ip will be enforced
    sync_host_ip: 'auto'

    # should be a unique port this sync instance uses on the host to offer the rsync␣
↪service on
```

(continues on next page)

(continued from previous page)

```
    # do not use this for unison - not needed there
    # sync_host_port: 10871

    # optional, a list of excludes. These patterns will not be synced
    # see http://www.cis.upenn.edu/~bcpierce/unison/download/releases/stable/unison-
↪manual.html#ignore for the possible syntax and see sync_excludes_type below
    sync_excludes: ['Gemfile.lock', 'Gemfile', 'config.rb', '.sass-cache', 'sass',
↪'sass-cache', 'composer.json' , 'bower.json', 'package.json', 'Gruntfile*', 'bower_
↪components', 'node_modules', '.gitignore', '.git', '*.coffee', '*.scss', '*.sass']

    # use this to change the exclude syntax.
    # Path: you match the exact path ( nesting problem )
    # Name: If a file or a folder does match this string ( solves nesting problem )
    # Regex: Define a regular expression
    # none: You can define a type for each sync exclude, so sync_excludes: ['Name .git
↪', 'Path Gemlock']
    #
    # for more see http://www.cis.upenn.edu/~bcpierce/unison/download/releases/stable/
↪unison-manual.html#pathspec
    sync_excludes_type: 'Name'

    # optional: use this to switch to rsync verbose mode
    sync_args: '-v'

    # optional, default can be either rsync or unison See Strategies in the wiki for␣
↪explanation
    sync_strategy: 'unison'

    # this does not user groupmap but rather configures the server to map
    # optional: usually if you map users you want to set the user id of your␣
↪application container here
    sync_userid: '5000'

    # optional: usually if you map groups you want to set the group id of your␣
↪application container here
    # this does not user groupmap but rather configures the server to map
    # this is only available for unison/rsync, not for d4m/native (default) strategies
    sync_groupid: '6000'

    # defines how sync-conflicts should be handled. With default it will prefer the␣
↪source with --copyonconflict
    # so on conflict, pick the one from the host and copy the conflicted file for␣
↪backup
    sync_prefer: 'default'

    # optional, a list of regular expressions to exclude from the fswatch - see␣
↪fswatch docs for details
    # IMPORTANT: this is not supported by native_osx
    watch_excludes: ['.*/.git', '.*/node_modules', '.*/bower_components', '.*/sass-
↪cache', '.*/.sass-cache', '.*/.sass-cache', '.coffee', '.scss', '.sass', '.gitignore
↪']

    # optional: use this to switch to fswatch verbose mode
    watch_args: '-v'

    # monit can be used to monitor the health of unison in the native_osx strategy␣
↪and can restart unison if it detects a problem
```

(continues on next page)

(continued from previous page)

```
    # optional: use this to switch monit monitoring on
    monit_enable: false

    # optional: use this to change how many seconds between each monit check (cycle)
    monit_interval: 5

    # optional: use this to change how many consecutive times high cpu usage must be␣
↪observed before unison is restarted
    monit_high_cpu_cycles: 2
```

docker-compose.yml

You should split your docker-compose configuration for production and development (as usual). The production stack (docker-compose.yml) does not need any changes and would look like this (and is portable, no docker-sync adjustments).

```
version: "2"
services:
  someapp:
    image: alpine
    container_name: 'fullexample_app'
    command: ['watch', '-n1', 'cat /var/www/somefile.txt']
  otherapp:
    image: alpine
    container_name: 'simpleexample_app'
    command: ['watch', '-n1', 'cat /app/code/somefile.txt']
```

docker-compose-dev.yml

The docker-compose-dev.yml (it needs to be called that way, look like this) will override this and looks like this.

```
version: "2"
services:
  someapp:
    volumes:
      - fullexample-sync:/var/www:nocopy # nocopy is important
  otherapp:
    # thats the important thing
    volumes:
      - simpleexample-sync:/app/code:nocopy #  nocopy is important

volumes:
  fullexample-sync:
    external: true
  simpleexample-sync:
    external: true
```

Tip: Do check that you use nocopy, see below for the explanation

So the docker-compose-dev.yml includes the volume mounts and definitions - your production docker-compose.yml will be overlaid by this when starting the stack with

```
docker-sync-stack start
```

This effectively does this in docker-compose terms

```
docker-compose -f docker-compose.yml -f docker-compose-dev.yml up
```

Portable docker-compose.yml

Most of you do not want to inject docker-sync specific things into the production `docker-compose.yml` to keep it portable. There is a good way to achieve this very cleanly based on docker-compose overrides.

1. Create a `docker-compose.yml` (you might already have that one) - that is your production file. Do not change anything here, just keep it the way you would run your production environment.
2. Create a `docker-compose-dev.yml` - this is where you put your overrides into. You will add the external volume and the mount here, also adding other development ENV variables you might need anyway

Start your compose using:

```
docker-compose -f docker-compose.yml -f docker-compose-dev.yml up
```

If you only have macOS- and Linux-based development environments, create `docker-compose-Linux.yml` and `docker-compose-Darwin.yml` to put your OS-specific overrides into. Then you may start up your dev environment as:

```
docker-compose -f docker-compose.yml -f docker-compose-$(uname -s).yml up
```

You can simplify this command by creating an appropriate shell alias or a Makefile. There is also a feature undergo to let `docker-sync-stack` support this out of the box, by simply calling:

```
docker-sync-stack start
```

A good example for this is a part of the boilerplate project.

Why :nocopy is important?

In case the folder we mount to has been declared as a VOLUME during image build, its content will be merged with the name volume we mount from the host - and thats not what we want. So with nocopy we ignore the contents which have been on the initial volume / image and do enforce the content from our host on the initial wiring

```
version: "2"
services:
  someapp:
    volumes:
      - fullexample-sync:/var/www
```

to

```
version: "2"
services:
  someapp:
    volumes:
      - fullexample-sync:/var/www:nocopy
```

Environment variables support

Docker-sync supports the use of environment variables from version 0.2.0.

The support is added via implementation of https://github.com/bkeepers/dotenv.

You can set your environment variables by creating a .env file at the root of your project (or form where you will be running the docker-sync commands).

The environment variables work the same as they do with docker-compose.

This allows for simplifying your setup, as you are now able to change the project dependent values instead of modifying yaml files for each project.

Tip: You can change the default file using `DOCKER_SYNC_ENV_FILE`, e.g. if .env is already used for something else, you could use `.docker-sync-env` by setting export `DOCKER_SYNC_ENV_FILE=.docker-sync-env`

```
# contents of your .env file
WEB_ROOT=/Users/me/Development/web
API_ROOT=./dir
```

The environment variables will be picked up by docker-compose

```
services:
  api:
    build: ${API_ROOT}
```

and by docker-sync as well.

```
# WEB_ROOT is /Users/me/Development/web
syncs:
  web-rsync:
    src: "${WEB_ROOT}"
```

For a detailed example take a look at https://github.com/EugenMayer/docker-sync-boilerplate/tree/master/dynamic-configuration-dotnev.

1.1.3 Commands

Sync commands (`docker-sync`)

Generally you can just list all the help in the cli by:

```
docker-sync help
```

Start

```
docker-sync start
```

Tip: See *Sync stack commands (docker-sync-stack)* on how `docker-sync-stack start` works to start sync / compose at the same time.

This creates and starts the sync containers, watchers and the sync itself. It blocks your shell and you should leave it running in the background. When you are done, just press `CTRL-C` and the containers will be stopped (not removed).

Running start the second time will be a lot faster, since containers and volumes are reused.

Tip: You can use `-n <sync-endpoint-name>` to only start one of your configured sync-endpoints.

Sync

```
docker-sync sync
```

This forces docker-sync to sync the host files to the sync-containers. You must have the containers running already (`docker-sync start`). Use this as a manual trigger, if either the change-watcher failed or you try something special / an integration

Tip: You can use `-n <sync-endpoint-name>` to only sync one of your configured sync-endpoints.

List

```
docker-sync list
```

List all available/configured sync-endpoints configured for the current project.

Clean

After you are done and want to free up space or switch to a different project, you might want to release the sync containers and volumes by

```
docker-sync clean
```

This will not delete anything on your host source code folders or similar, it just removes the container for sync and its volumes. It does not touch your application stack.

Sync stack commands (`docker-sync-stack`)

With docker-sync there comes docker-sync-stack (from 0.0.10). Using this, you can start the sync service and docker compose with one single command. This is based on the gem docker-compose.

Start

```
docker-sync-stack start
```

This will first start the sync service like `docker-sync start` and then start your compose stack like `docker-compose up`.

You do not need to run `docker-sync start` beforehand!

This is very convenient so you only need one shell, one command to start working and CTRL-C to stop.

Clean

```
docker-sync-stack clean
```

This cleans the sync-service like `docker-sync clean` and also removed the application stack like `docker-compose down`.

Daemon mode

Docker-sync in daemon mode

Beginning with version **0.4.0** Daemon mode is now the default, just use `docker-sync start`. `docker-sync-daemon` is deprecated.

Beginning with version **0.2.0**, docker-sync has the ability to run in a daemonized (background) mode.

In general you now run *docker-sync-daemon* to start in daemonized mode, type `docker-sync-daemon <enter>` to see all options

Start

The *docker-sync-daemon start* command has the following options to help configure daemon mode:

- `--app_name` (`--name`), The name to use in the filename for the `pid` and `output` files (default: 'daemon')
- `--dir`, The directory to place the `pid` and `output` files (default: './.docker-sync')
- `--logd`, Whether or not to log the output (default: true)

Stop

The `docker-sync-daemon stop` command is available to stop the background process. It also takes the `--app_name` and `--dir` arguments.

Log

The `docker-sync-daemon logs` command is a handy shortcut to tail the logs from the daemonized process, in addition to the `--app_name` and `--dir` from above, it takes the following arguments:

- `--lines`, Specify the maximum number of lines to print from the current end of the log file (defaults to 100)
- `--follow` (`-f`), Whether or not to continue following the log (press ctrl+c to stop following)

Examples

Instead of docker-sync-stack start

The way `docker-sync-stack start` used to operate was to begin to sync the container(s) specified in the `docker-sync.yml` file, and then begin a `docker-compose up`. The simplest way to replace this command is to use:

```
docker-sync-daemon start
docker-compose up
```

This will start your sync in the background, and then start all services defined in your docker-compose file in the foreground. This means that your sync continues in the background, even if you exit your `docker-compose` session(s). You can then stop that background sync with:

```
docker-sync-daemon stop
```

This will show the logs for the daemon started above

```
docker-sync-daemon logs
```

Running commands before starting the docker-compose services

By having the sync run in the background, you can then use a single shell session to ensure that the sync is running, and then run a few commands before starting all your services. You may wish to do this if you would like to use volumes to speed up rebuilds for node modules or gem bundles - as volumes are not available while building the image, but are when building the container.

```
docker-sync-daemon start
docker-compose run --rm $service yarn install
docker-compose up -d
```

This will ensure that your sync containers are up and available so that commands utilizing the docker-compose file don't fail for not finding those containers. It will then run all services in the background.

Notes

New directory

This will now create a `.docker-sync` directory alongside wherever you invoke the command (if you're asking it to run in the background). You will likely want to add this directory to your `.gitignore` file (or equivalent). You can, of course, use the `--dir` option to specify an alternate directory to save these files, but be sure to pass the same argument to `stop`, and to use it consistently, or you may end up with multiple sync's running in the background...

Invoking with the –config option

I imagine most users will be invoking `docker-sync` without specifying an alternate path to the config file, but it's worth mentioning that if that's your current setup, you should also consider using the `app_name` option or the `dir` option to ensure that your `pid` file won't conflict with other invocations of docker-sync - otherwise you'll get a message saying that it's already running.

1.1.4 Upgrade

0.4.2-0.4.5

Nothing special, just be sure to pull the newest docker-images for your strategy and do not leave any older version behind

---

0.4.1

- `:nocopy` needs to be added to all named-volume mounts, see *Why :nocopy is important?*
- if you want to use native_osx with docker-machine (toolbox) + virtualbox, it will not work https://github.com/EugenMayer/docker-sync/issues/346

The `:nocopy` issue has been there for a while, but nobody really recognized it.

0.4.0

Attention: Ensure you run docker-sync clean after the upgrade. Do not reuse the old containers!

The default strategy is now native_osx

Read more at *native_osx (OSX)*.

Background by default

`docker-sync start` does now background by default, use `--foreground` to run in foreground.

sync_user now removed

Being deprecated in 0.2, it's now throwing an error if still present

docker-sync-daemon is now deprecated

It's deprecated and will be removed in 0.5

0.3.0

Reinstallation of unox

Due to a lot of issues and inconvenience with the installation of unox and the lack of versioning of unox, i took the step to create a homebrew formula myself, while working with the unox author hand in hand. This way we can ease up the installation and also be able to avoid issues as https://github.com/EugenMayer/docker-sync/issues/296 The installer will take care of everything for you in this regard

Scaffolding usages needs to be migrated

If you scaffolded / scripted with docker-sync using it as a ruby lib, you will now need to change your implementation due to the changes to preconditions and config. Important new/replacing calls are. Please see the updated example at *Scripting* for how to load the project config, how to get its path and how to call the preconditions

Dest has been removed

After making dest deprecated in the 0.2 release, since we introduce named volume mounts, it is now removed. Just remove it, the destination is set in the docker-compose-dev.yml file like this

```
version: "2"
services:
  someapp:
   volumes:
     - fullexample-sync:/var/www:rw # will be mounted on /var/www
```

So here, the destination will be `/var/www`

0.2.0+

Versioning of docker-sync.yml

From 0.2.0 you need to add a new setting to your docker-sync.yml `version: "2"` - this describes the project version you are using. This is needed so later we can easily detect old/incompatible configuration files and warn you to migrate those. This is now mandatory. See this change on the boilerplates / examples, which may explain it even better.

Unison exclude syntax is Name by default now - migrate your entries

Prior to 0.2.0 the exclude default syntax of the unison strategy was "Path" - since we decided that this is counter-intuitive in most cases, we have changed the default to `Name` - please see the unison documentation for more - mostly you would have expected the `Name` behavior anyway, so you might want to stick with it. TLTR: `Path` math matches the exact path (not sub-directories) while name just matches string on the path - no matter which "nesting level". You could go back to `Path` by setting sync_exclude_type to 'Path'.

See this issue: Make Name the default exclude_type in 0.2.0.

rsync trailing slash changes

Prior to 0.2.0 the trailing slash was automatically added - but now you have to do this explicitly If you define an rsync sync, you most probably want to sync the inner folder into you destination, without creating the parent folder / syncing it. This trailing slash `./your-code/` ensures exactly that, so `your-code` will not be created on your destination, but anything inside it will be synced.

Default sync is now unison (from rsync to unison)

If you did not provide the sync_strategy setting prior 0.2.0 - rsync was used. Starting with 0.2.0 unison(dual sided) is the new default, so a 2 way sync. Beside its just being better, faster after the initial sync and also offers 2-way sync, it has a new Exclude-syntax. With 0.2.0 the `Name` exclude syntax is used, ensure you adjust your rsync ones to fit those.

See this issue: Migration Guide from rsync to unison as default.

volumes_from: container: syntax is no longer used

The `volumes_from: container:app-sync:rw` syntax is no longer used as a volume mount for the sync container, but rather `volumes: app-sync:/var/www:rw`

See this issue: Rework the way we mount the volume.

–prefer is now built in - remove it from sync_args

If you have used sync_args for unison and defined `--prefer`, please consider removing it. Without doing anything, docker-sync will now use `--prefer <srcpath> --copyonconflict` and also help you keep the src dynamic (depending on the developer).

The option sync_user no longer exists

`sync_user` has been removed, since it does not add any useful stuff, but spreads a lot of confusion. Please use `sync_userid` solely to define the user-mapping, no need to manually set the `sync_user` anymore.

Remove the old unison:unox image

Since the name was misleading anyway, please remove the old unison image: `docker image rm eugenmayer/unison:unox`.

The rsync / unison images have been remade and aligned

To share more code and features between the rsync / unison images, we aligned those images to share the same codebase, thus they have been renamed. The ENV variables have changed and some things you should not even notice, since it is all handled by `docker-sync` - all you need to know is, you need to pull the new versions if you have disabled the auto-pull (which you should not).

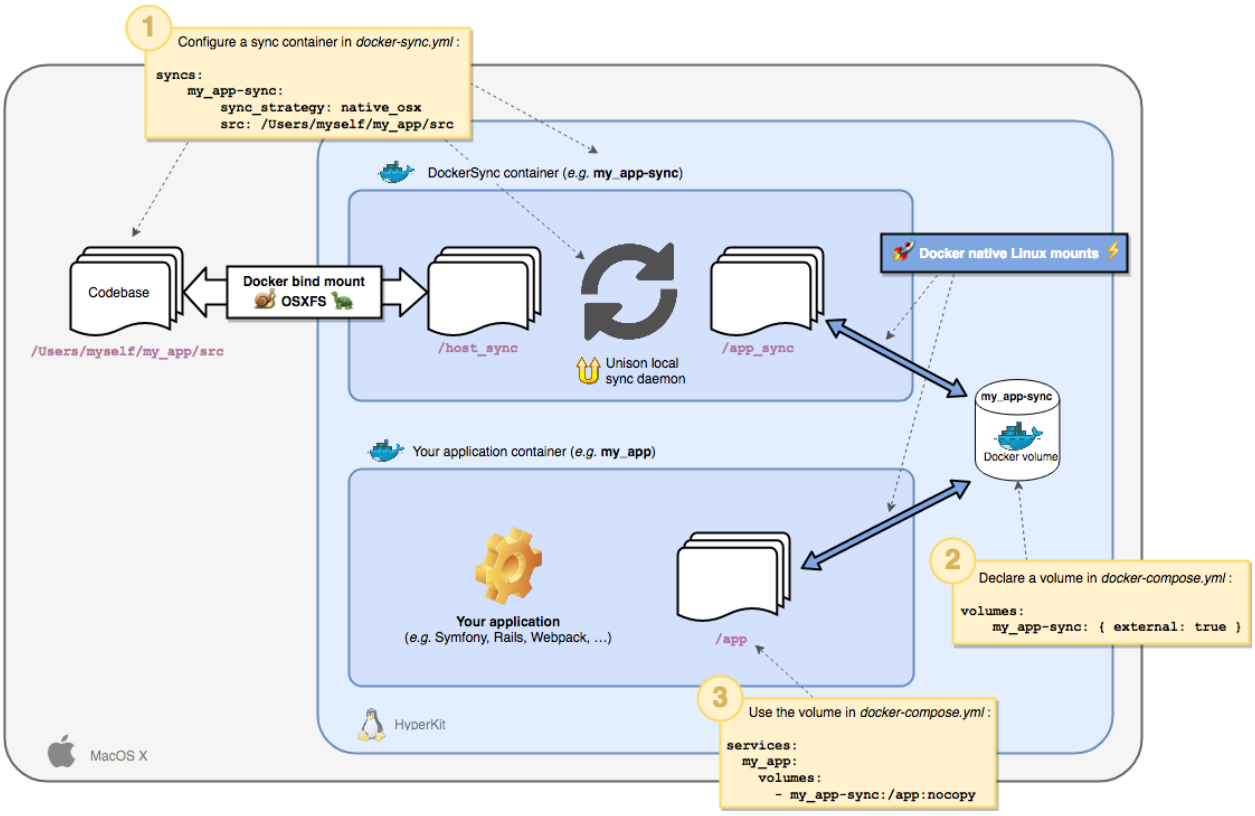
1.1.5 Sync strategies

The sync strategies depend on the OS, so not all strategies are available on all operating system

- OSX: native_osx, unison, rsync
- Windows: unison
- Linux: native_linux, unison

---

native_osx (OSX)



For advanced understanding, please read *native_osx in depth*.

Native-OSX is a combination of two concepts, OSXFS only and Unison together. We use OSX to sync the file-system into a sync-container to /host_sync. In that sync container we sync from /host_sync to /app_sync using Unison. /app_sync is exposed as a named volume mount and consumed in the app. You ask yourself, why? Its fairly simple.

By having this extra layer of unison on linux, we detach the actual write/read performance of your application from the actual OSXFS performance - running at native speed. Still, we using OSXFS, a about up to 1 second delayed, to synchronize changes both ways. So we have a 2-way sync.

What is different to plain Unison you might ask. The first big issue with Unison is, how bad it performs under OSX due to the bad FS-Events of OSX, implemented in macfsevents and alikes. It uses a lot of CPU for watching files, it can lose some events and miss a sync - but also, it adds extra dependencies on the OSX hosts.

All that is eliminated by native_osx - we use Unison in the Linux container, where it performs great due to inotify-event based watchers.

Pros

- Far more reliable due to a low-level implementation of the sync using OSXFS only
- Uses far less CPU to sync files from the host to the sync container - will handle a lot more files
- No daemon to run, control, restart and sync on the OSX host. Makes sleep/hibernate/reboot much more robust
- No dependencies on the OSX-Host at all
- A lot easier installation since we just need `gem install docker-sync` and that on runs under system ruby. Anything else is in containers
- It performs at native speed
- It will be much easier to support Windows this way

Cons

- Initial start can take longer as on unison, since the first sync is more expensive. But this is one time only
- It works under Docker for Mac only - missing file system events under vbox/fusion. See native_osx does not work with docker-machine vbox / fusion

---

unison (OSX/Windows/Linux)

This strategy has the biggest drive to become the new default player out of the current technologies. It seems to work very well with huge codebases too. It generally is build to handle 2 way sync, so syncs back changes from the container to the host.

Pros

- Offers 2 way sync (please see unison-dualside why this is misleading here)
- Still very effective and works for huge projects
- Native speed in for the application

Cons

- Can be unreliable with huge file counts (> 30.000) and bad hardware (events gets stuck)
- The daemon on OSX needs extra care for sleep/hibernate.
- Extra dependencies we need on OSX, in need to install unison and unox natively - brew dependencies

Initial startup delays with unison

On initial start, Unison sync needs to build a catalog of the files in the synced folder before sync begins. As a result, when syncing folders with large numbers of relatively large files (for example, 40k+ files occupying 4G of space) using unison, you may see a significant delay (even 20+ minutes) between the initial `ok Starting unison` message and the `ok Synced` message. This is not a bug. Performance in these situations can be improved by moving directories with a large number of files into a separate `rsync` strategy sync volume, and using unison only on directories where two-way sync is necessary.

---

rsync (OSX)

This strategy is probably the simplest one and probably the most robust one for now, while it has some limitations. rsync-syncs are known to be pretty fast, also working very efficient on huge codebases - no need to be scared of 20k files. rsync will easily rsync the changes (diff) very fast.

Pros

- Fast re-syncing huge codebases - sync diffs (faster then unison? proof?)
- Well tested and known to be robust
- Best handling of user-permission handling (mapped into proper users in the app container)

Cons

- Implements a one way sync only, means only changes of your codebase on the host are transferred to the app-container. Changes of the app-container are not synced back at all
- Deleting files on the host does yet not delete them on the container, since we do not use –delete, see #347

Example: On the docker-sync-boilerplate

---

native_linux

Native linux is actually no real implementation, it just wraps docker-sync around the plain native docker mounts which do work perfectly on linux. So there are no sync-containers, no strategies or what so ever. This strategy is mainly used just to have the whole team use docker-sync, even on linux, to have the same interface. If you use default sync_strategy in the docker-sync.yml, under Linux, native_linux is picked automatically

---

Sync Flags or Options

You find the available options for each strategy in *Configuration*.

1.1.6 Scripting

We use docker-sync as a library in our own docker-stack startup script. It starts the docker-compose stack using a Ruby gem EugenMayer/docker-compose all this wrapped into a thor task. So:

- Start docker-sync

- Start a docker-compose stack based on some arguments like –dev and load the specific docker-compose files for that using xeger/docker-compose

docker-sync-stack is actually an example already, just see here:

1. You run the sync manager with run : https://github.com/EugenMayer/docker-sync/blob/master/tasks/stack/stack.thor#L37
2. But you do not call .join_threads after that like her https://github.com/EugenMayer/docker-sync/blob/master/tasks/sync/sync.thor#L36
3. Then you just continue doing what you want to script, in my case, i start a new blocking task - docker-compose. But you could do anything.

Simple scripting example

```
require 'docker-sync/sync_manager'
require 'docker-sync/dependencies'
require 'docker-sync/config/project_config'

# load the project config
config = DockerSync::ProjectConfig.new(config_path: nil)
DockerSync::Dependencies.ensure_all!(config)
# now start the sync
@sync_manager = Docker_sync::SyncManager.new(:config_path => config_path)
@sync_manager.run() # do not call .join_threads now

#### your stuff here
@sync_manager.watch_stop()
system('my-bash-script.sh')
some_ruby_logic()
system('other-tasks.sh')
@sync_manager.sync()
@sync_manager.watch_start()
### debootsrapping
begin
  @sync_manager.join_threads
rescue SystemExit, Interrupt
  say_status 'shutdown', 'Shutting down...', :blue

  @sync_manager.stop
rescue Exception => e

  puts "EXCEPTION: #{e.inspect}"
  puts "MESSAGE: #{e.message}"

end
```

1.1.7 How it works

Architecture

On the host, a thor based ruby task is started, this starts:

- Every sync will start an own docker-container with a rsync/unison-daemon watching for connections.
- The data gets pre-synced on sync-start

- A fswatch cli-task gets setup, to run rsync/unison on each file-change in the source-folder you defined

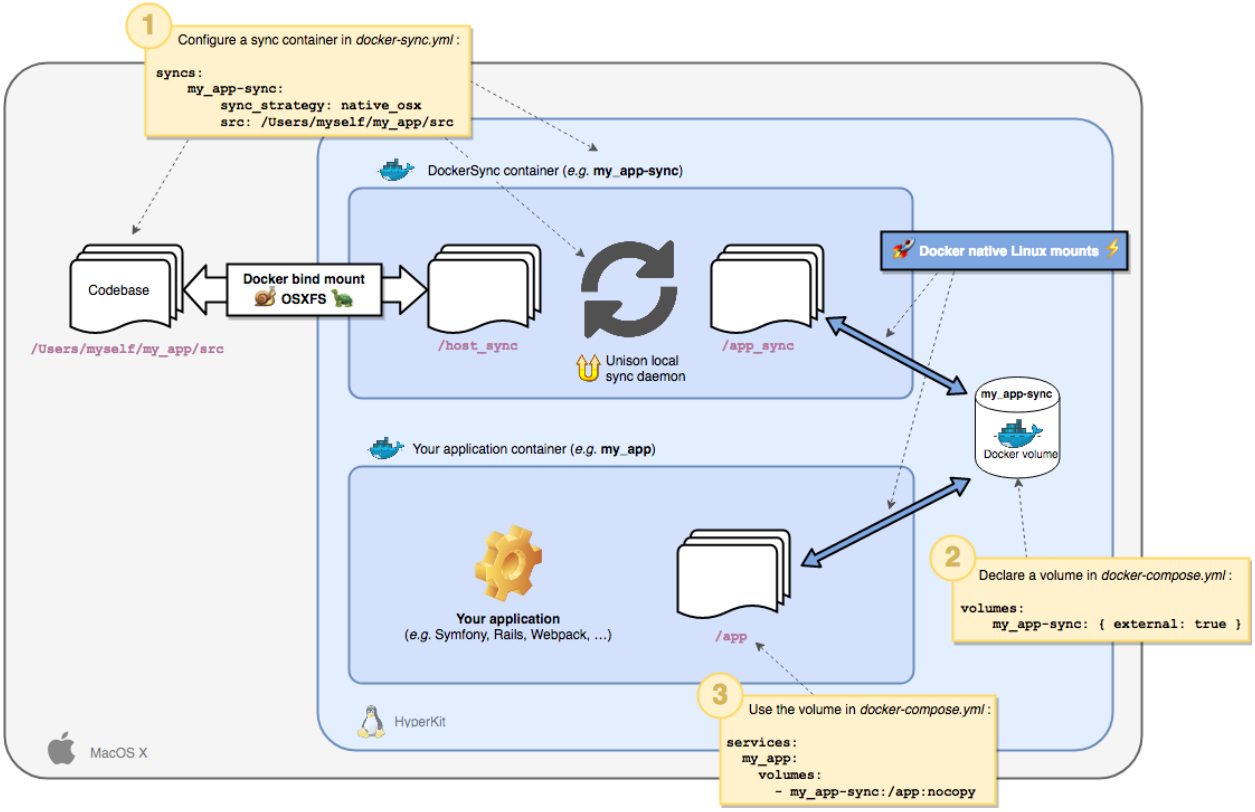
Done. No magic. But its roadrunner fast! And it has no pre-conditions on your actual stack.

---

native_osx in depth

Under The Hood

First, take a look at this diagram:



There are some important keypoints to notice here:

1. We use OSXFS to mount your local host folder into the sync-container.
2. We do not mount this in the app-container directly, since this would lead to infamously horrible performance.
3. Instead of directly mounting `/host_sync` in the app-container we setup a **2-way-sync** inside the sync-container using Unison. This ensures that the actual READ/WRITE performance on the `/app_sync` folder is native-speed fast.
4. This makes all operations on `/app_sync` be asynchronous with `/host_sync`, since writing and reading on `/app_sync` does not rely on any OSXFS operation directly, **but shortly delayed and asynchronous**.
5. We mount `/app_sync` to your app_container - since this happens in hyperkit, it's a **Docker LINUX-based** native mount, thus running at native-speed.
6. Your application now runs like there was no sync at all.

FAQ

Why use OSXFS in the first place (instead of the `unison` strategy) to sync from the host to the sync-container

There are several reasons, one of the most important being the performance. Since MacOS/OSX has very bad filesystem events support on HFS/APFS, watching the file-system for changes using `unox` or `fswatch` was causing a heavy CPU load. This CPU load is very significant, even on modern high-end CPUs (like a i7 4770k / 3.5GHz).

The second issue was dependencies. With native_osx you do not need to install anything on your host OS except the docker-sync gem. So no need to compile unox or install unison manually, deploy with brew and fail along the way - just keeping you system clean.

Is this strategy absolutely bullet proof?

No, it is not. But it has been pretty battle proven already - the main issue is https://github.com/EugenMayer/docker-sync/issues/410 - so sometimes OSXFS just stops triggering FS events in Hyperkit, thus in the sync-container. This leads to an issue with our sync, since the `unison` daemon inside the app-sync container relies on those events to sync the changes (it does not have the ability to poll, which would be disastrous performance-wise, anyway).

Advanced Monitoring for `native_osx`

Background

Monit is a utility which can be used to monitor the health of the `unison` process which runs in the container for the `native_osx` strategy. If it detects that `unison` is unhealthy, Monit automatically restarts `unison`. This improves the stability of the `native_osx` container in cases where the `unison` process is misbehaving but does not necessarily crash. Currently, there is only one check for CPU usage implemented, but in the future more checks may be added, such as memory usage. It is currently turned off by default and can be turned on in the configuration:

https://github.com/EugenMayer/docker-sync/blob/master/example/docker-sync.yml#L120-L126

Monitoring CPU usage

One instance which `unison` has been seen to misbehave is when quickly creating and deleting a file while it is processing it. `unison` may hang, using a high amount of cpu time: https://github.com/EugenMayer/docker-sync/issues/497. `monit` detects this high cpu usage (>50%) and automatically restarts `unison` to recover it. By default this happens within 10 seconds, but the tolerance can be configured in case there are normal spikes in cpu usage during successful syncs.

1.1.8 Tips and tricks

HTTP Proxy and DNS

The HTTP Proxy and DNS used in dinghy is available as a standalone project dinghy-http-proxy. The proxy is based on jwilder's excellent nginx-proxy project, with modifications to make it more suitable for local development work. A DNS resolver is also added. By default it will resolve all `*.docker` domains to the Docker VM, but this can be changed.

SSH-Agent Forwarding

If you need to access some private git repos or ssh servers, it could be useful to use have a ssh-agent accessible from your containers. whilp/ssh-agent helps you to do so easily.

Running composer or other tools like if they were on the host

If you run composer and other tools directly in containers, you could use a combination of the autoenv zsh plugin and a simple wrapper script to run it easily directly from the host. In your project folder, create a `.autoenv.zsh` file with the name of your container:

```
autostash COMPOSER_CONTAINER='project_fpm_1'
```

then create a simple function in your `.zshrc`:

```
composer () {
    if [ ! -z $COMPOSER_CONTAINER ]; then
      docker exec -it ${COMPOSER_CONTAINER} /usr/local/bin/composer --working-dir=/
↪src -vv "$@"
    else
      /usr/local/bin/composer "$@"
    fi
}
```

Ignore files in your IDE

It's a good idea to add the temporary sync files to your IDE's ignore list to prevent your IDE from indexing them all the time or showing up in search results. In case of unison and PHPStorm for example just go to Preferences -> File Types -> Ignore files and folders and add `.unison` to the pattern.

Don't sync everything

You should only sync files that you really need on both the host and client side. You will see that the sync performance will improve drastically when you ignore unnecessary files. How and which files to ignore depends on your syncing strategy (rsync/unison/...) and your project.

Example for a PHP Symfony project using unison:

```
# docker-sync.yml
syncs:
  appcode-unison-sync:
    ...
    sync_args:
        - "-ignore='Path .idea'"          # no need to send PHPStorm config to␣
↪container
        - "-ignore='Path .git'"           # ignore the main .git repo
        - "-ignore='BelowPath .git'"      # also ignore .git repos in subfolders such␣
↪as in composer vendor dirs
        - "-ignore='Path var/cache/*'"    # don't share the cache
        - "-ignore='Path var/sessions/*'" # we don't need the sessions locally
        - "-ignore='Path node_modules/*'" # remove this if you need code completion
        # - "-ignore='Path vendor/*'"     # we could ignore the composer vendor␣
↪folder, but then you won't have code completion in your IDE
```

1.1.9 Sync stopping

Npm / webpack / gulp based projects

Npm, webpack, gulp, and any other tooling that watches, deletes and/or creates a lot of files may cause sync to stop.

In most cases, this has nothing to do with docker-sync at all, but with OSXFS getting stuck in the FS event queue, which then also stops events for unison in our docker image (linux, so inode events) and thus breaks syncing.

- Run `npm i`, `composer install`, and the likes **before** `docker-sync start`. This way we avoid tracking unnecessary FS events prior to start.
- Sync only necessary folders, e.g. `src/` folders. Restructure your project layout if needed.

Other reported solutions

1. Run `docker-sync stop && docker-sync start`
2. Run `docker-sync stop && docker-sync clean && docker-sync start`
3. Manually going to the unison docker container and executing `kill -1 [PID]` on the unison process (suggested by @grigoryosifov)
4. Sometimes, the OSXFS itself gets stuck. Docker for Mac restart maybe the only option. Sometimes, an OS restart is the only option.

1.1.10 Other common issues

Incorrect mount location

Docker-sync uses a named container/volume for synchronizing. There is a chance you may have a conflicting sync name. To verify this, run:

```
docker container inspect --format '{{(index .Mounts 1).Source}}' "$DEBUG_DOCKER_SYNC"
```

If you do not see the path to your directory, this means your mount location is conflicting. To fix this issue:

1. Bring your containers down
2. Perform `docker-sync clean`
3. Bring your containers back up again.

1.1.11 Advanced troubleshooting for `native_osx` strategy

Note: This document is a work in progress. Each time you encounter the scenario below, please revisit this document and report any new findings.

The osx_native sync strategy is the fastest sync strategy for docker-sync under Docker4Mac. Unfortunately a recurring issue has emerged where the sync strategy stops functioning. This page is to guide you on how to debug this situation to provide information so that it can be solved.

Step 1 - Prepare: Identify the docker-sync container involved

First, open your docker-sync.yml file and find the sync that has to do with the code that appears to be failing to sync.

For example, if you have two docker-sync mounts like so:

```
syncs:
  default-sync:
    src: './default-data/'
  fullexample-sync:
    src: './data1'
```

And your file that is not updating is under `default-data`, then your sync name is `default-sync`.

Run this in your terminal (substitute in your sync name) for use in the remaining steps: `DEBUG_DOCKER_SYNC='default-sync'`

Step 2 - Prepare: A file path to check

Next we're going to assign a file path to a variable for use in the following steps.

1. Change into your sync directory (in the example `cd default-data/`)
2. Prepare the relative path to your file that does not appear to be updating upon save, example `some-dir/another-dir/my-file.ext`
3. Run the following command with your path substituted in: `DEBUG_DOCKER_FILE='some-dir/another-dir/my-file.ext'`

Step 3 - Reproduction: Verify your host mount works (host_sync)

Run this to verify that your file changes have been synced by OSXFS to the sync-container

```
diff -q "$DEBUG_DOCKER_FILE" <(docker exec "$DEBUG_DOCKER_SYNC" cat "/host_sync/
↪$DEBUG_DOCKER_FILE")
```

Usually this should never get broken at all, if it does, you see one of the following messages, the so called host_sync is broken:

```
Files some-dir/another-dir/my-file.ext and /dev/fd/63 differ
diff: some-dir/another-dir/my-file.ext: No such file or directory
```

Step 4 - Reproduction: Verify your changes have been sync by unison (app_sync)

Run this to verify that the changes have been sync from host_sync to app_sync on the container (using unison)

```
diff -q "$DEBUG_DOCKER_FILE" <(docker exec "$DEBUG_DOCKER_SYNC" cat "/app_sync/$DEBUG_
↪DOCKER_FILE")
```

If you see a message one of the messages, this so called app_sync is broken:

```
Files some-dir/another-dir/my-file.ext and /dev/fd/63 differ
diff: some-dir/another-dir/my-file.ext: No such file or directory
```

If you do not see a message like one of these, then the issue you are encountering is not related to a sync failure and is probably something like caching or some other issue in your application stack, not docker-sync.

Step 5 - Reproduction: Unison log

If one of the upper errors occurred, please include the unison logs:

```
docker exec "$DEBUG_DOCKER_SYNC" tail -n70 /tmp/unison.log
```

And paste those on Hastebin and include the link in your report

Step 6 - Reproduction: Ensure you have no conflicts

Put that into your problematic sync container docker-sync.yml config:

```
sync_args: "-copyonconflict -debug verbose"
```

Restart the stack

```
docker-sync-stack clean
docker-sync-stack start
```

Now do the file test above and see, if next to the file, in host_sync or app_sync a conflict file is created, its called something like conflict

Also then include the log

```
docker exec "$DEBUG_DOCKER_SYNC" tail -n70 /tmp/unison.log
```

And paste those on Hastebin and include the link in your report

Step 7 - Report the issue

If the issue still persists, post the results from the above steps in a new Github issue (see an example at *issue #410*)_.

1.1.12 Development & testing

Development

You do not really need a lot to start developing.

- A local Ruby > 1.9 (I think we need that)

```
git clone https://github.com/eugenmayer/docker-sync
cd docker-sync
bundle install
gem uninstall -a docker-sync
```

Important: To properly develop, uninstall docker-sync as a gem so it is not used during the runs: `gem uninstall -a docker-sync`.

Now you can:

```
cd example
thor sync:start
```

or

```
thor stack:start
```

So you see, what is separated in to binaries in production `docker-sync` and `docker-sync-stack` is bundled under one namespace here, but prefixed.

General layout

Check libs folder.

- SyncManager: Main orchestrator to initialise the config, bootstrap ALL sync-endpoint-processes and start/stop those in threads
- SyncProcess: Does orchestrate/a manage ONE sync-endpoint. Selects the strategy on base of the config
- Strategies: See below, specific implementations how to either sync or watch for changes.

Sync strategies

1. To add a new strategy for sync, copy one of those https://github.com/EugenMayer/docker-sync/tree/master/lib/docker_sync/sync_strategy here as your
2. Implement the general commands as they are implemented for rsync/unison - yes we do not have an strategy interface and no abstract class, since its ruby .. and well :)
3. Add your strategy here: https://github.com/EugenMayer/docker-sync/blob/master/lib/docker_sync/sync_process.rb#L31

Thats it.

Watch strategies

1. To add a new strategy for watch, copy one of those https://github.com/EugenMayer/docker-sync/tree/master/lib/docker_sync/watch_strategy here as your
2. Implement the general commands as they are implemented for fswatch
3. Add your strategy here: https://github.com/EugenMayer/docker-sync/blob/master/lib/docker_sync/sync_process.rb#L46

Thats it.

Testing

Automated integration tests

```
bundle install
bundle exec rspec --format=documentation
```

Manual Tests (sync and performance)

Tip: You can also use the docker-sync-boilerplate.

Pull this repo and then

```
cd docker-sync/example
thor stack:start
```

Open a new shell and run

```
cd docker-sync/example
echo "NEWVALUE" >> data1/somefile.txt
echo "NOTTHEOTHER" >> data2/somefile.txt
```

Check the docker-compose logs and you see that the files are updated.

Performance write test:

```
docker exec -i -t fullexample_app time dd if=/dev/zero of=/var/www/test.dat bs=1024␣
↪count=100000
```

1.1.13 Performance

Performance

OSX

Setup	Native (out of the box)	Docker-Sync	
		Unison	native_osx
Docker Toolbox - VMware Fusion	12.31s	0.24s	n/a (issue)
Docker Toolbox - VirtualBox	3.37s	0.26s	n/a (issue)
Docker for Mac	20.55s	0.36s	0.28s
Docker for Mac Edge	18.12s	0.27s	0.19s
Docker for Mac Edge + APFS	18.15s	0.38s	0.37s
Docker for Mac Edge :cached	17.65s	0.21s	0.22s

Setup and details below at *Performance Tests 2017*.

Windows

Coming soon.

Linux

Coming soon.

Performance Tests 2017

Results

Test: writing 100MB

Setup	Native (out of the box)	Docker-Sync	
		Unison	native_osx
Docker Toolbox - VMware Fusion	8.70s	0.22s	n/a (issue)
Docker Toolbox - VirtualBox	3.37s	0.26s	n/a (issue)
Docker for Mac	18.85s	0.24s	0.28s
Docker for Mac Edge	18.12s	0.27s	0.19s
Docker for Mac Edge :cached	17.65s	0.21s	0.22s

Those below is how the tests were made and how to reproduce them:

Setup

Test-hardware

- i76600u
- 16GB
- SSD
- Sierra

Docker Toolbox VMware Fusion machine:

```
docker-machine create --driver vmwarefusion --vmwarefusion-cpu-count 2 --vmwarefusion-
↪disk-size 50000 --vmwarefusion-memory-size 8000 default
```

Docker for Mac

- 8GB
- CPUs

Native implementations

Those tests run without docker-sync or anything, just plain what you get out of the box.

VirtualBox - Native

```
docker-machine create --driver virtualbox --virtualbox-cpu-count 2 --virtualbox-disk-
↪size 20000 --virtualbox-memory "8000" vbox
```

```
docker run -it -v /Users/em/test:/var/www alpine time dd if=/dev/zero of=/var/www/
↪test.dat bs=1024 count=100000
100000+0 records in
100000+0 records out
real  0m 3.37s
user  0m 0.00s
sys 0m 2.09s
```

3.37s

VMware Fusion - Native

```
docker run -it -v /Users/em/test:/var/www alpine time dd if=/dev/zero of=/var/www/
↪test.dat bs=1024 count=100000
100000+0 records in
100000+0 records out
real  0m 12.32s
user  0m 0.14s
sys 0m 2.22s
```

12.31s

Docker for Mac - Native

- 8GB Ram
- 2 CPUs

```
docker run -it -v /Users/em/test:/var/www alpine time dd if=/dev/zero of=/var/www/
↪test.dat bs=1024 count=100000
100000+0 records in
100000+0 records out
real  0m 18.85s
user  0m 0.11s
sys 0m 1.06s
```

20.55s

Docker-sync - Strategy: Native_osx

Get this repo and this boilerplate project

```
git clone https://github.com/EugenMayer/docker-sync-boilerplate
cd docker-sync-boilerplate/default
docker-sync-stack start
```

Vmware Fusion

```
docker exec -it nativeosx_app-unison_1 time dd if=/dev/zero of=/var/www/test.dat␣
↪bs=1024 count=100000
100000+0 records in
```

(continues on next page)

(continued from previous page)

```
100000+0 records out
real  0m 0.32s
user  0m 0.02s
sys 0m 0.24s
```

0.32s

Docker for Mac

```
docker exec -it nativeosx_app-unison_1 time dd if=/dev/zero of=/var/www/test.dat␣
↪bs=1024 count=100000
100000+0 records in
100000+0 records out
real  0m 0.28s
user  0m 0.02s
sys 0m 0.25s
```

0.26s

Docker-Sync - Strategy: Unison

Get this repo and this boilerplate project

```
git clone https://github.com/EugenMayer/docker-sync-boilerplate
cd docker-sync-boilerplate/unison
docker-sync-stack start
```

VirtualBox

```
docker exec -it unison_app-unison_1 time dd if=/dev/zero of=/var/www/test.dat bs=1024␣
↪count=100000
100000+0 records in
100000+0 records out
real  0m 0.26s
user  0m 0.00s
sys 0m 0.23s
```

VMware Fusion

```
docker exec -it unison_app-unison_1 time dd if=/dev/zero of=/var/www/test.dat bs=1024␣
↪count=100000
100000+0 records in
100000+0 records out
real  0m 0.24s
user  0m 0.01s
sys 0m 0.23s
```

Docker for Mac

```
docker exec -it unison_app-unison_1 time dd if=/dev/zero of=/var/www/test.dat bs=1024␣
↪count=100000
100000+0 records in
100000+0 records out
real  0m 0.24s
user  0m 0.04s
sys 0m 0.16s
```

0.36s

1.1.14 Alternatives

This is a list of alternatives grouped by technology. Feel free to add the missing ones.

Docker native

Transparent, consistent, dual-sided (host -> container, container -> host) synchronization. Performance is here a trade-off for consistency. Can be 2-100 times slower than nfs and even more as compared with rsync.

- docker-toolbox: virtualBox / fusion VM (horribly slow)
- docker for mac: uses osxfs. See osxfs-caching for optimization ideas. As of October 2017, they aren't proven to be effective yet.

OSXFS + unison

Dedicated container mounts a local directory via osxfs and runs Unison to synchronize this mount with a Docker volume. - docker-magic-sync - docker-sync implements osxfs+unison-based sync when 'native_osx' is used as a strategy, being the default since 0.4.x. We use a special technique to achieve better performance, we sync with osxfs but the container still runs at native speed, let's call it decoupled sync.

Unison

Unison runs both on the host and in a Docker container and synchronizes the macOS directory with a Docker container with Unison. **osxfs + unison** is a preferred alternative, because it's simpler and more reliable (bad FSEvents performance).

- docker-sync - unison can be used with docker-sync as well as a strategy, just set *sync_strategy: unison*
- Hodor (should be as fast as rsync?)

Tip: You can choose to use Unison with docker-sync by adding sync_strategy: 'unison' to a sync-point too

Rsync

Performance: Exactly the performance you would have without shares. Downside: **one-way sync**.

- docker-sync - rsync can be used with docker-sync as well as a strategy, just set *sync_strategy: rsync*

- docker-dev-osx (rsync, vbox only) - Hint: If you are happy with docker-machine and virtual box, this is a pretty solid alternative. It has been there for ages and is most probably pretty advanced. For me, it was no choice, since neither i want to stick to VBox nor it has support for docker-for-mac

NFS

Performance: In general, at least 3 times slower than **rsync**, often even more.

- Dinghy (docker-machine only, no docker for mac)
- DLite (docker-machine only, no docker for mac)
- Dusty (docker-machine only, no docker for mac)

1.1.15 Changelog

Attention: This is legacy/deprecated. Newer changelogs are now part of the releases. See https://github.com/EugenMayer/docker-sync/releases.

0.5.0

Note: This release has no breaking changes, so it is a drop-in-replacement for 0.4.6 without migration.

Features/Improvements

- Integrations tests - huge credits to @michaelbaudino
- FreeBSD support
- print sync time so you can see potential stalls earlier #431
- be able to set max_attempt in global configuration #403
- added support for d4m edge #478
- added diagram to explain how native_osx works #465
- upgraded terminal-notifier to 2.0.0 #486
- upgraded docker-compose gem to 1.1 #486
- upgraded thor to 0.20 #486

Bugfixes:

- default ip detection fixed
- fix several typos and docs #432 #409 #404 #396
- fix exceptions thrown #406
- unison mount destination #433
- fix issues with spaces in folder / paths when using unison #426

Special thanks to @michaelbaudino who has done a incredible job

---

0.4.6

Fixes:

- Fixed Issue introduced with 0.4.5: #367

Nothing else - most probably the last 0.4.x release.

0.4.5

See https://github.com/EugenMayer/docker-sync/milestone/22?closed=1 for the bugfixes

Windows and Linux support now got documented and the documentation has been made more cross-platform.

Mot probably last 0.4.x maintenance release - with 0.5.x a rewrite of the Config/Dependency/Env/Os handling is supercharged by @michaelbaudino. This will help improving the overall quality of the codebase and reduce the clusterfuck when we do cross-platform implementation / splits. Be tensed.

0.4.2

- Implement proper selective sync default: native_osx for d4m and unison for docker-machine, see https://github.com/EugenMayer/docker-sync/issues/350
- When you run –foreground with native_osx you now see the unison logs running in the container, see https://github.com/EugenMayer/docker-sync/issues/341
- Properly pull new unison:hostsync image to fix 0.4.0 bugs

0.4.1

- Fixing issue with sync_userid and native_osx
- Fixing different new and old issues with native_osx and unison / installation and upgrade issues
- More at https://github.com/EugenMayer/docker-sync/issues?q=is%3Aclosed+milestone%3A0.4.1

0.4.0

- **New Sync Strategy native_osx.** See *native_osx (OSX)*
- Daemon mode is now the default mode
- No need for unison/unox by default when using native_osx
- Better performance when using native_osx
- Fixed auto-ip guessing
- More at https://github.com/EugenMayer/docker-sync/milestone/17?closed=1

0.3.6

- **Finally removed any support of 'dest' which has been deprecated since 0.2.x.**
- Linux support: docker-sync can now also be used under Linux, were it does a fallback to native volume mounts automatically.
- Introducing auto-guessing of the sync_host_ip for simultaneous usage of the same `docker-sync.yml` using d4m, docker-toolbox and others. Just set `sync_host_ip: 'auto'`
- Fixed spaces in ./src lead to issues with unison
- Fixed various issues with the installation of 0.3.x
- Fixed issues with the new configuration model
- Overall making docker-sync more robust and verbose if things are not as intended
- More at https://github.com/EugenMayer/docker-sync/milestone/16?closed=1

0.3.1 - 0.3.5

- Bugfixes

0.3.0

- You can now chose the dotenv file to be used by docker-sync using setting DOCKER_SYNC_ENV_FILE
- The configuration has been rewritten, huge thank you to @ignatiusreza for his effort. This was done to support better scaffolding (inline configuration loading), prepare linux support (or windows cygwin) and to simplify the code / reduce its madness factor
- The precondition checks have been reworked to be simpler and more convinient
- Unox has now been packaged using brew, which makes the installation of unox/unison easier
- Unox has been upgrading to use watchdog instead of macfsevents, which should improve performance
- Several installation issues have been fixed
- Stopping docker-sync now runs synchronously, avoiding accidental race conditions

Thank you a lot for the contributions guys, a lot of team effort in this release!

0.2.3

- Smaller Bugfixes and minor features: https://github.com/EugenMayer/docker-sync/releases/tag/0.2.3

0.2.1

- Smaller bugfixes https://github.com/EugenMayer/docker-sync/milestone/15?closed=1

0.2.0

- You can now start docker-sync in daemon mode `docker-sync-daemon`. See *Daemon mode*.
- The default sync strategy is now unison, no longer rsync. Check *Upgrade*.
- Unison sync now starts slightly faster
- New default setting for `--prefer`: `--prefer <src> --copyonconflict`. Check *Upgrade*.
- Detection of macfsevents installation including some edge cases does properly work now #243.
- You can now run `docker-sync start --version` to see your version
- You can now use spaces in the src/dest path #211.
- unison:onesideded sync has been entirely removed. Check *Upgrade*.
- `sync_user` option has been removed (use `sync_userid` only), since it only spread confusion. Check *Upgrade*.
- Better way of mounting sync-volumes. Check *Upgrade*.
- sync_exclude 'type' for unison is now *Name*, not `Path` by default. Check *Upgrade*.
- You can now use environment variables in your docker-sync.yml using `dotenv`. See *Environment variables support*.
- unison using `--testserver` now to avoid startup issues and also speedup the startup
- Check for updates only for the actually strategy picked, not all
- Add support for `--abort-on-container-exit` for docker-compose #163.
- To share more code and features between the rsync / unison images, we aligned those images to share the same codebase, thus they have been renamed. The ENV variables are changed and some things you should not even notice, since it is all handled by docker-sync. Check *Upgrade*.
- Fix dynamic port detection with unison / make it more robust #247.
- New and more robust unison/rsync images

0.1.2

- Adjustments and bugfixes
- Full changelog at: https://github.com/EugenMayer/docker-sync/releases/tag/0.1.2

0.1.1

- Small bugfixes

0.1.0

- **Unison-Unox strategy for transparent 2-way sync introduced.**
- Full changelog at: https://github.com/EugenMayer/docker-sync/releases/tag/0.1.0

0.0.15

- **Notifications, cli mode**
- cli-mode selection https://github.com/EugenMayer/docker-sync/pull/66
- Notifications on sync https://github.com/EugenMayer/docker-sync/pull/63, thank you midN

0.0.14

- **Welcome unison-dualside for real 2-way-sync**
- unison-dualside strategy introduced for real 2 way syncing, thank you mickaelperrin. See *Sync strategies*.
- New image for rsync based on alpine (10MB), thank you Duske.
- Optimize fswatch to watch only useful events (better performance), thank you mickaelperrin
- Different fixes with filepaths, symlinks and some minors
- Detailed list at https://github.com/EugenMayer/docker-sync/milestone/5?closed=1

0.0.13

- **docker-compose-dev.yml make docker-compose.yml portable**
- By moving all changes initially made to your docker-compose.yml into docker-compose-dev.yml, your production docker-compose.yml stays portable #41
- Fixing a bug when docker-sync / docker-sync-stack has been symlinked #44 by mickaelperrin

---

0.0.12

- **Unison slim image, docker-compose path and fswatch disabling**
- You can no configure were you docker-compose file is located at. See *Configuration*.
- You can now disable the filewatcher using watch_strategy. See *Configuration*.
- docker-compose gem is now part of the gem
- gem / lib was re-layouted to fit the library usage better
- tons of requires have been fixed for the script usage. See *Scripting*.
- A alpine based, slim unison image was created by onnimonni. Thank you!

- You can now customize which unison/rsync image you want to use (experts only please!)

0.0.11

- **docker-sync-stack is here**
- **You can now start sync and docker-compose in one go** - See *Sync stack commands (docker-sync-stack)*.
- rsync image is now checked for update ability to avoid issues with outdated images

---

0.0.10

- Yanked, broken release

0.0.9

- **Adresses further unison issues, minor features**
- Missing stdout pipe and wrong color, thank you @mickaelperrin
- More verbose outputs on unison runs with verbose,, thank you @mickaelperrin
- Adding update-checker to ensure, that you run the newest docker-sync

---

0.0.8

- **Fix unison startup**
- Fixed issue during unison startup

0.0.7

- ** Convenience / Bugfixes**
- **Add the possibility to map user/group on sync**
- Fixed container-re-usage issue
- Add preconditions to properly detect if fswatch, unison, docker, and others are in proper state
- Better log output
- Do no longer enforce verbose flag
- Remove colorize
- Be less verbose in normal mode

- Fixed source code mapping when using test
- Renamed test to example

0.0.6

- **Critical issue in sync**
- Fixing critical issue where sync has been called using the old sync:sync syntax - not syncing at all

---

0.0.5

- **Added unison support**